Content

1. What is operating System?
2. Why OS?
3. Functions of operating system?
4. Operating system goals.
5. Types of operating system.
6. Multitasking vs Multithreading.
7. Component of OS. (Kernel & User Space)
8. System Calls.
9. What happen when you turn on your PC/computer?
10. 32 bit vs 64 bit OS.
11. What are various memory present in computer system?
12. Introduction to Process.
13. Process States
14. Process Queues
15. Swapping.
16. Context switching
17. Orphan process
18. Zombie Process.
19. Process scheduling
20. CPU scheduler.
21. Non-Preemptive scheduling
22. Preemptive scheduling.

(23) Goal of Scheduling.
(24) FCFS
(25) SJF (NP)
(26) SJF (P)
(27) priority Scheduling (NP).
(28) priority Scheduling (P).
(29) Round Robin.
(30) Multi level Queue Scheduling.
(31) Multilevel feedback scheduling.
(32) Concurrency
- threads
- Thread content switching
- Benefit of Multithreading.

(33) Critical Section Problem.
(34) Conditional variables Semaphores
(35) Producer - consumer problem.
(36) Reader - writer Problem.
(37) The Dining philosopher problem.
(38) Deadlock.
(39) Deadlock Avoidance.
(40) Memory Management.
(41) Defragmentation.
(42) Paging
(43) Segmentation
(44) Virtual Memory
(45) Page Replacement Algorithm, Thrashing.

Operating System

1. What is an Operating System?

- Application Software performs specific task for the user.

- System Software operates & control the computer system & provide 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 & software, & provides an environment in which the user can execute his programs in a convenient & efficient manner by hiding underlying complexity of the hardware and acting as a Resource Manager.

2. Why OS?

① what if there is no OS?

- Bulky & complex app. { Hardware interaction code must be in app's code base }

| Memory allocation & many more Hardware things

- Resource Exploitation by 1 app. { Basically take control over other apps Memory Place }

- No memory Protection.

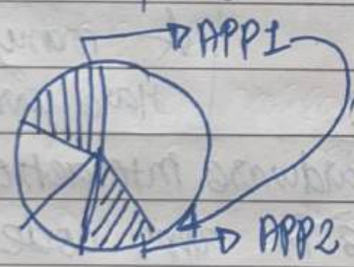
__/__/____

② what is an OS made up of ?
- Collection of system softwares.

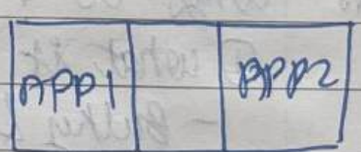
3. Functions of Operating System

- Access to the computer Hardware. {allocation of best Memory for app} }
- Acts as interface between the user & computer Hardware.
- Resources Management (Arbitration)
  { Memory, File, Process, Security } → Resources.
- Hides the underlying complexity of the Hardware (Abstraction).
- Facilitates the execution of application Programs by providing Isolation & protection.

Example {Isolation & Protection}



attack for other app storage



(without OS)

(with OS)

Operating System provides the means for proper use of resources in the operation of Computer System.

4. Operating System Goals

- Maximum CPU utilization. { OS will give processes to CPU in ideal time }
- Less Process Starvation. { Process waits indefinitely to get CPU or resources }
- High Priority Job Execution.

5. Types of Operating System

Single Process OS	MS-DOS
Batch Processing OS	ATLAS
Multiprogramming OS	THE
Multitasking OS	C
Multi-Processing OS	WindowsNT
Distributed OS	LOCUS
Real Time OS	ATCS

① Single Process OS, only 1 process executes at a time from the Ready Queue.

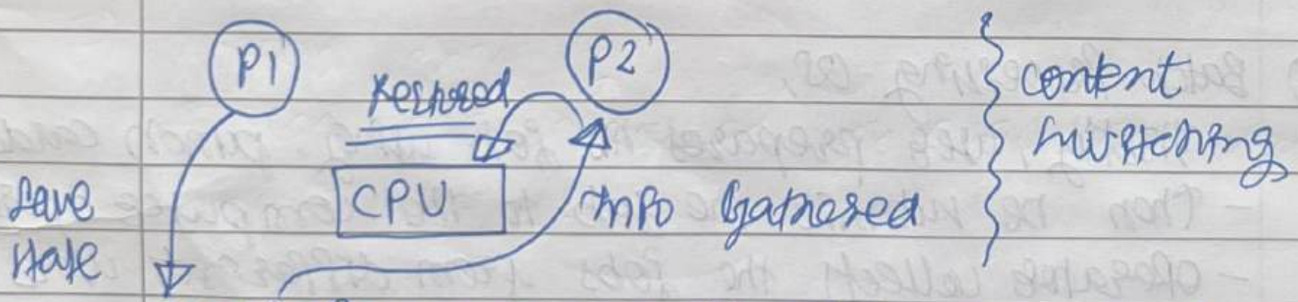
② Batch Processing OS,
- Firstly, user prepares his job using punch cards.
- Then he submit the job to the computer operator.
- Operator collects the jobs from different users & sort the jobs into batches with similar needs.

- Then, operators submits the batches to the processor one by one
- All the jobs of one batch are executed together.

Disadvantages
- Priorities cannot be set, if a job come with higher priority.
- Lead to starvation.
- CPU may become idle in case of I/O operation.

③ Multi Programming OS,
Multiprogramming increases CPU utilization by keeping multiple jobs in the memory so that the CPU always has one to execute in case some job gets busy with I/O.

- Single CPU.
- Switch happens when current Process goes to wait state.
- CPU Idle time Reduced.
- Content switching for processes.



(PCB) Process Control Block is data structure store Process Info

④ Multitasking is a logical extension of multiprogramming.

- Single CPU.
- Able to exec more than one task simultaneously.
- Context switching & Time sharing is used.
- CPU Idle time is reduced.

⑤ Multiprocessing OS, more than 1 CPU in a single comp.

- Increase Reliability, 1 CPU fails other can work.
- Better throughput
- Lesser Process starvation.
  { Process can work multiple CPUs }

⑥ Distributed OS { Loosely Coupled OS }

- OS manages many bunches of resources, >= 1 CPUs, >= 1 memory, >= 1 GPU, etc.

Example: Leetcode Judge

- Interconnected computer nodes.
- Collection of independent, networked nodes.

⑦ Real Time OS,

- Real time error free, computations within tight time boundaries.
- Air traffic control system, ROBOTS etc.

6. Multi-Tasking vs Multi-Threading

① Program: A Program is an executable file which contains a certain set of instruction written to complete the specific job or operation on your computer.
- It is a Compiled code, Ready to execute
- Stored in DISK

② Process: Program Under execution, resides in RAM.

③ 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.
- Example, Multiple tabs in browser.

④ Thread Scheduling
Threads are scheduled for execution based on their priority.

	Multi-Tasking	Multi-Threading
①	The execution of more than one task simultaneously is called as "multitasking".	① The process is divided into several different sub-tasks called threads, which has its own path of execution, concept called multithreading.
②	concept of more than 1 process being content switched.	② concept of more than 1 thread. Thread are content switched.
③	NO. of CPU 1.	③ NO. of CPU >= 1.
④	Isolation & memory Protection apply.	④ No Isolation & memory Protection

	Thread Content Switch	Process Content Switch
①	OS saves current state of thread & switches to another thread of same Process.	① OS saves current state of Process & switches to another Process by restoring its state.
②	Doesn't include switching memory address space.	② Include switching address space
③	Fast switching	③ Slow switching.

__/__/__

7. Components of OS

① Kernel, A Kernel is that part of the operating system which interacts directly with the hardware & performs the most crucial tasks.
- Heart of OS
- very first part of os to load on start-up.

② User Space, where application software runs, apps don't have privileged access to the underlying hardware. It interacts with Kernel
- GUI
- CLI.

③ Functions of Kernel
a. Process Management
- Scheduling processes & threads on the CPUs.
- Creating & deleting both user & system process.
- Suspending & Resuming Processes.
- Provide mechanisms for Process communication.

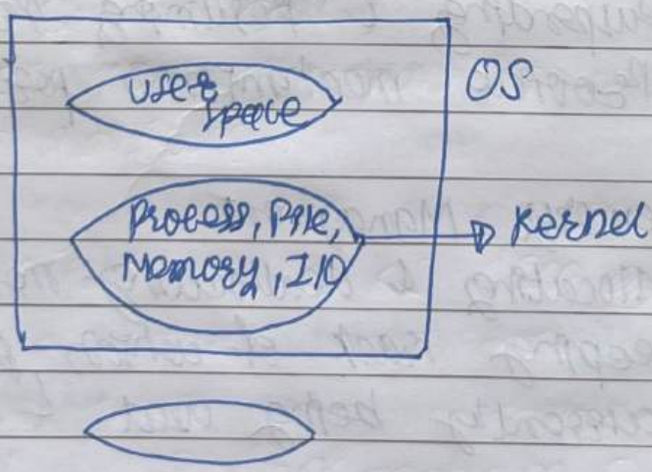
b. Memory Management
- Allocating & deallocating memory space as per need.
- Keeping track of which part of memory are currently being used & by which process.

c. File Management
- Creating & deleting files.
- Creating & deleting directories to organize.
- Mapping files into secondary storage.
- Backup support onto a stable storage media.

d. I/O Management, to manage & control I/O operations & I/O devices.

④ Types of Kernel

a. Monolithic Kernel
- All function are in kernel itself.
- Bulky in size.
- Memory required to even is high.
- Less reliable.
- High performance as communication is fast.
- Eg Linux, Unix, MS-DOS.



b. Micro kernel
- Only major function are in kernel
  Memory } Management.
  Process
- File } Management are in User-space.
  IO
- Smaller in size
- More Reliable
- More Stable.
- Performance is slow.
- Overhead switching betn kernel & user space
- e.g. L4 Linux, Symbian OS, MINIX, etc.

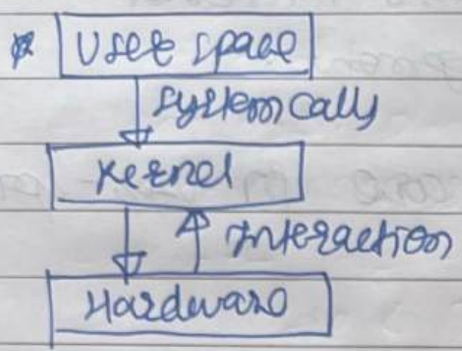
} disadvantage }

How will communication happen betn user space & kernel ?
→ IPC (Inter Process Communication)
→ Shared Memory
→ Message Passing

c. Hybrid kernel
- Advantage of both worlds
  File → User space
  Rest → Kernel
  are
- Combined approach,
- Speed & design of monolithic
- IPC happens but less overheads
- e.g. MacOS, Windows NT.

__/__/____

8. System Call



① How do apps interact with kernel? Using system calls?
e.g. Mkdir xyz
- mkdir indirectly call kernel and asked the file management module to create a new directory.
- mkdir is just wrapper of actual system calls.
- mkdir interact with kernel using system call.

e.g Creating a Process.
- User executes a process. (User)
- gets System call
- Execution (kernel)
- Return (user)

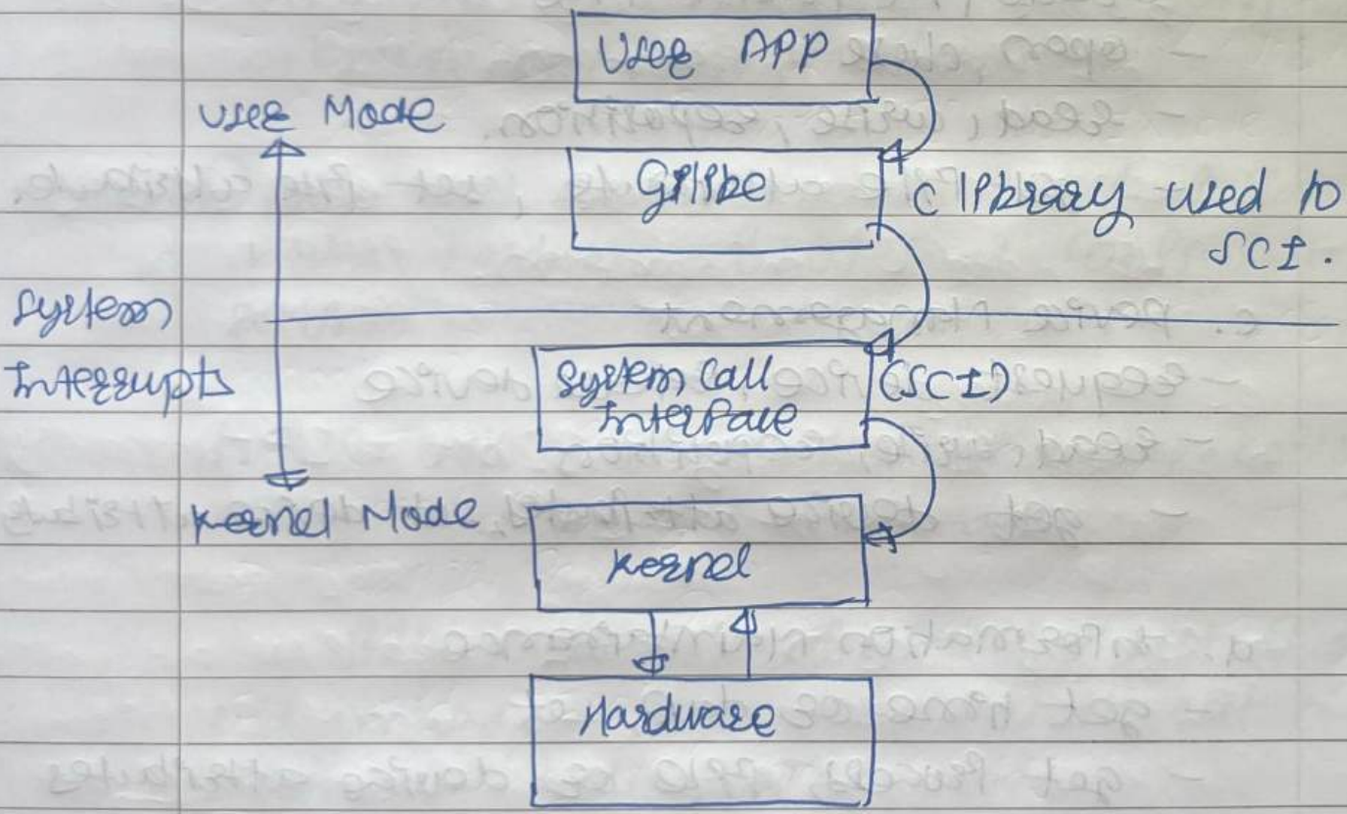
+ Transition from US to KS done by software interrupts.

+ System calls are implement 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 program typically don't have permission to perform operation like accessing I/O devices.

System Calls are the only way through which program can go into kernel mode from user mode.



__/__/____

② Types of System Calls :

a. Process Control
- end, abort
- load, execute
- create process, terminate process
- wait for time.
- allocate & free memory.

b. File Management
- create file, delete file
- open, close
- read, write, reposition.
- get file attribute, set file attribute.

c. Device Management
- request device, release device
- read, write, reposition
- get device attributes, set device attributes

d. Information Maintenance
- get time or date, set
- get process, file or device attributes

e. Communication Management
- create, delete communication
- send, receive message
- transfer status information
- attach or detach remote devices.

9. What happens when you turn on your computer?

(a) PC ON.

(b) CPU initialize itself & look for a firmware program (BIOS) stored in BIOS chip (Basic input output System chip is a ROM chip that allow access & setup computer system at basic level)

In modern PC, CPU loads UEFI (Unified extensible firmware interface).

(c) CPU runs the BIOS which tells it initialize system hardware. BIOS load configuration settings.

(d) BIOS will handoff responsibility for booting your PC to your OS's bootloader.

BIOS looked for MBR (master boot record), MBR contains the code that load the rest of OS. known as "bootloader".

(e) The bootloader is a small program that has large task of booting the OS. basically boots (user & kernel)

windows Bootloader	→	Bootmgr.exe
Mac	→	boot.efi
Linux	→	GRUB.

10. 32 bit vs 64 bit OS.

a. A 32-bit OS has 32 bit registers, it can access 2^{32} unique memory address i.e 4GB of physical Memory.

b. A 64 bit ---
17179869184 GB of physical memory.

c. 32 bit CPU architecture can read 32 bits of data & information

d. 64 bit CPU architecture can read 64 bits of data & information.

e. Advantages

→ Addressable Memory → 32 bit = 2^{32} memory addresses, 64 bit = 2^{64}.

→ Resource usage → Installing more RAM on system with a 32 bit OS doesn't impact performance. However, upgrade that system with excess RAM to the 64 bit version of windows increase performance.

→ Performance → Having larger registers allow you to perform larger calculation at the same time.
32 bit processor can execute 4 byte of data in 1 instruction cycle.
64 bit processor can execute 8 byte of data in 1 instruction cycle.

→ Compatibility → 64 bit CPU can run both 32 bit & 64 bit OS.
32 bit can only run 32 bit OS.

→ Better graphic performance.

11. Memory present in the computer system?

a. Register, smallest unit of storage it is a part of CPU itself.
b. Cache, stores the frequently used instruction, data.
c. Main Memory, ROM.
d. Secondary Memory, store media, data & Program

Comparison

Cost — Primarily storage are costly.
Registers are expensive due to expensive semiconductor.

Access speed — Primary has higher access speed than secondary memory.
e.g. Registers

Storage size [secondary has more space

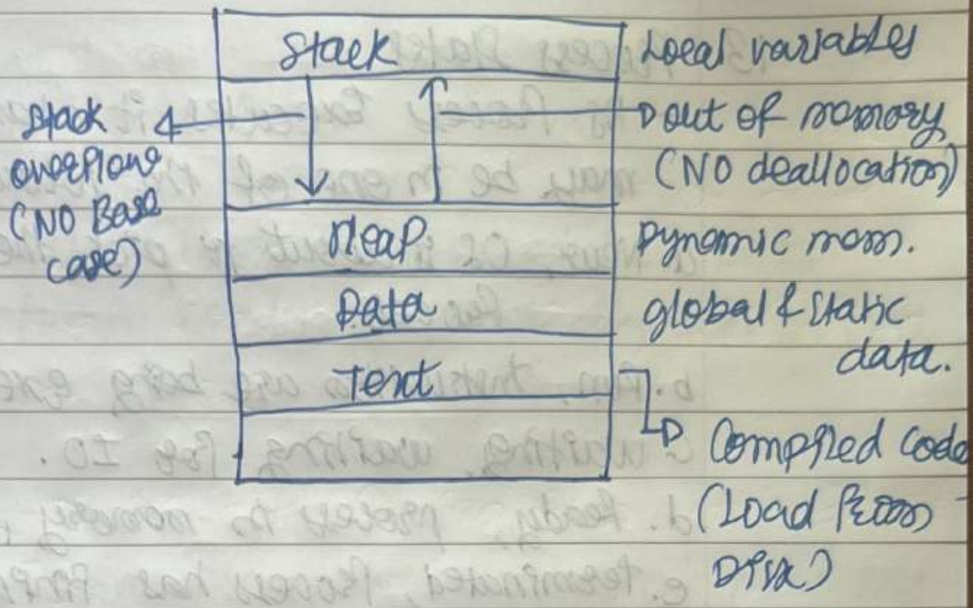
Volatility [Primary memory is volatile.
secondary is non-volatile.

12. Introduction to Process

what is Program? Compiled code Ready to execute
what is Process? Program under Execution.
How OS creates a Process? Converting Program into a Process.

Steps:
- Load the Program & Static data into memory. (Give Name to Process.)
- Allocate runtime stack. (Part of memory used for local variable, function arg, etc)
- Allocate Heap. (assigned dynamic data/ram)
- IO tasks
- OS hands to control main().

① Architecture :



② Attributes of Process

a. Features that allow identifying process uniquely.

b. Process Table

- All processes are being tracked by OS using table (PS).
- Each entry in the table is PCB (Process Control Block).

PCB structure

Process ID	Unique identifier
Program Counter	Next instruction address
Process state	Store Process state
Priority	Priority (High / low)
Registers	Save the CPU Registers.
List of open files	File descriptor
List of open device	& Device Descriptor

{New, wait, Run}

SP, Stack Pointer

BP, Base Pointer

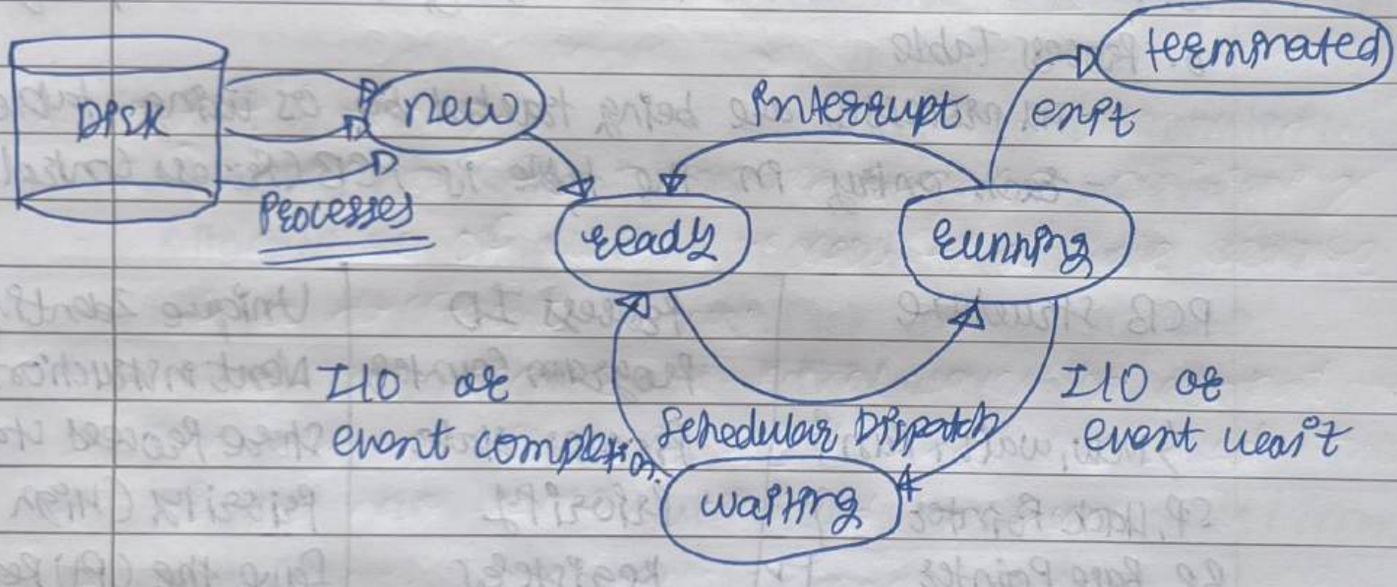
CR, Control Register

} → Registers

List down the file & Device version will be helpful to run the process.

Register in the PCB is like data structure, when Process is Running & its Time Slice expires the current value of specific register stored in PCB, & when the Process is scheduled the Register value Read from PCB & written on CPU, Registers.

13. Process States
- As Process Executes, it change state. Each process may be in one of the following.
a. New, OS is about to pick the program & convert it into Process.
b. Run, Instructions are being executed, CPU is allocated
c. Waiting, waiting for IO.
d. Ready, process in memory, waiting to be assigned to a {processor.
e. Terminated, Process has finished execution. PCB entry removed from table.}



14. Process Queues,
a. Job Queue
- Processes in new state, present in Secondary Mem.
- Job Scheduler (Long term Scheduler), picks processes from pool & load them into Ready Queue or Memory.

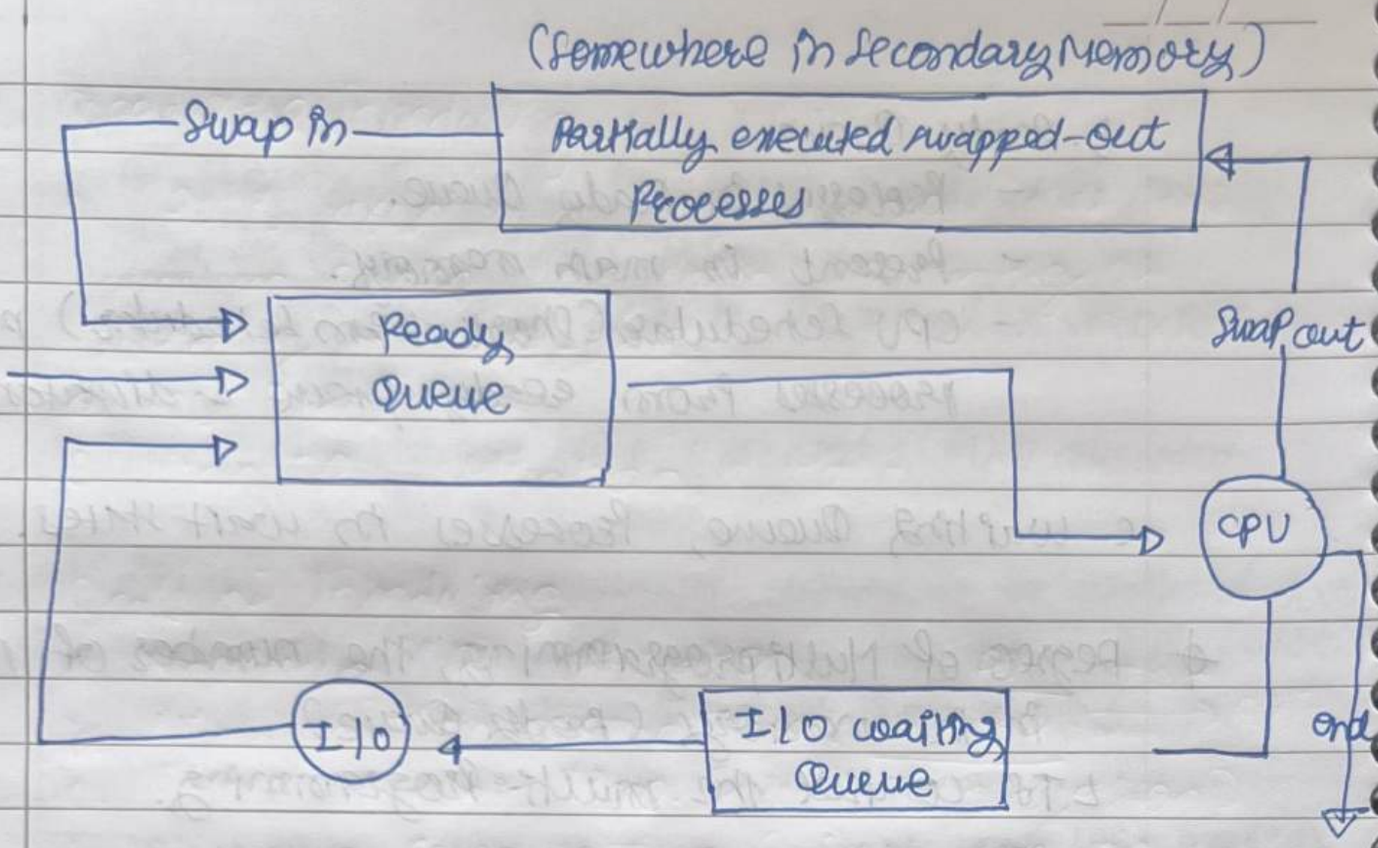
b. Ready Queue
- Processes in Ready Queue.
- Present in main memory.
- CPU Scheduler (Short-term Scheduler) picks processes from ready queue & dispatch in CPU.

c. Waiting Queue, Processes in wait states.

⊕ Degree of Multiprogramming, the number of processes in the memory (Ready Queue)
LTS control the multi-programming.

⊕ Dispatcher, Module of OS that give control of CPU to Process selected by STS (Short-term Scheduler)

15. Swapping (MTS)
a. Time-sharing system may have medium term scheduler
b. Remove process from Ready Queue to Reduce degree of multiprogramming.
c. These removed processes can be re-introduced in memory & its execution can be continued where it left off. called swapping.
d. Swap in & swap out is done MTS.
e. Swapping is a mechanism in which process mix or because a change in memory requirement has overcommitted available memory, requiring memory to be freed up.



16. Context Switching
a. Switching to CPU to another requires performing state save of current process & a state restore of a different process.
b. When this occurs, the kernel saves the context of the old process in its PCB & loads the saved context of the new process scheduled to run.

17. Orphan process
a. The process whose parent process has been terminated & it's still running.
b. Orphan processes are adopted by Init process.
c. Init Process is the first process of the OS

Fork() ⇒ Command is responsible creating child Processes

18. Zombie Process
a. A zombie process whose execution is completed but it still has an entry in the process table.
b. Zombie processes usually occure for child processes, as the parent process still need to read its child's exit status.
c. Parent Process call wait() on child process for a longer time duration & child process got terminated much earlier.
d. Hence, the child process remains zombie till it is removed from process table.

19. Process Scheduling
a. Basic of Multiprogramming OS.
b. By switching the CPU among the processes, the OS can make the computer more productive.

20. CPU Scheduler
a. whenever the CPU become Idle, OS must select one process from the ready queue to executed.
b. Done by STS.

21. Non-Preemptive Scheduling
a. Once CPU is allocated to a Process, then the Process keeps the CPU until it release CPU either by terminating or switching to wait state
b. Starvation.
c. Low CPU utilization.

22. Preemptive Scheduling

a. CPU is taken away from a process after time quantum expire along with terminating or shifting to wait-state.

b. Less Starvation.

c. High CPU utilization.

23. Goals of Scheduling

a. Maximum CPU utilization.

b. Minimum Turn around Time. P1 [Ready Q] TAT → exit

c. Minimum wait time. { Leads to starvation }

d. Minimum Response Time { First time on which CPU is scheduled }

e. Maximum Throughput
   └ No. of Processes / time.

- Arrival Time :- Time when process is arrived at Ready Q
- Burst Time :- Time required for its execution.
- TAT : (CT - AT)
- wait Time :- Time process spends waiting for CPU.
  WT = (TAT - BT)
- Response Time :- Time duration bet^n process in Ready Queue & Process getting in CPU for first time.
- Completion Time :- Time taken till process get terminated.

24. FCFS,
a. whichever Process comes first in the Ready Queue will be given CPU first.
b. In this, if one process has longer BT, it will have major effect on average waiting time of different processes, called as 'Convoy effect'.
- This cause poor resource Management.

25. SJF (non-preemptive), First.
a. Process with least BT will be dispatched to CPU^
b. Run lowest time process for all time than, choose job having lowest BT at that instance.
c. This will suffer from convoy effect as if the very first process which came in Ready state is having a large BT.
d. Process starvation may happens.
e. criteria, AT + BT.

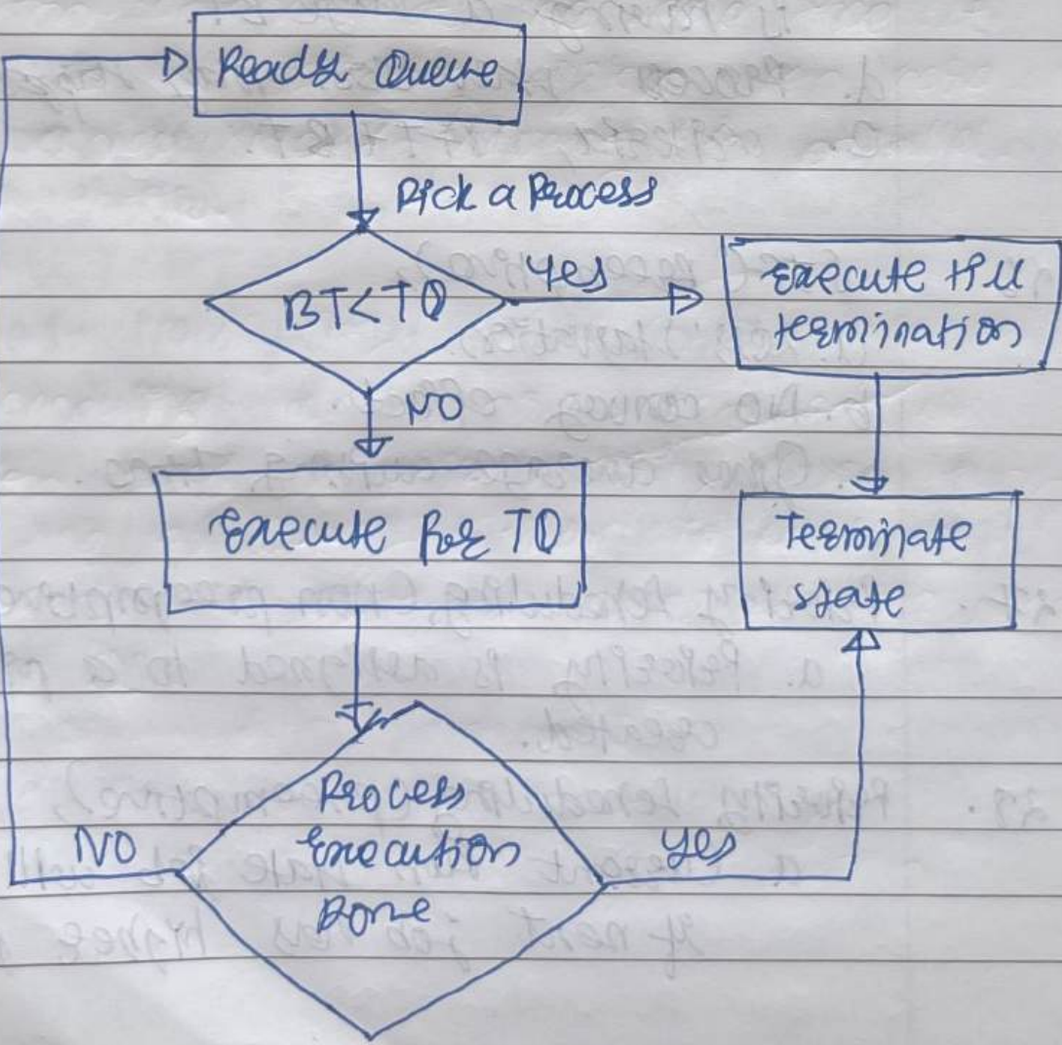
26. SJF (Preemptive),
a. Less starvation.
b. No convoy effect.
c. Gives average waiting time.

27. Priority Scheduling (non-preemptive),
a. Priority is assigned to a process when it is created.

28. Priority Scheduling (preemptive),
a. Current Run state job will be preempted if next job has higher priority.

b. may cause indefinite waiting for lower priority job.
→ Solution (Ageing) →
Gradually increase priority of process that wait so long,
Increase priority by 1 every 15 min.

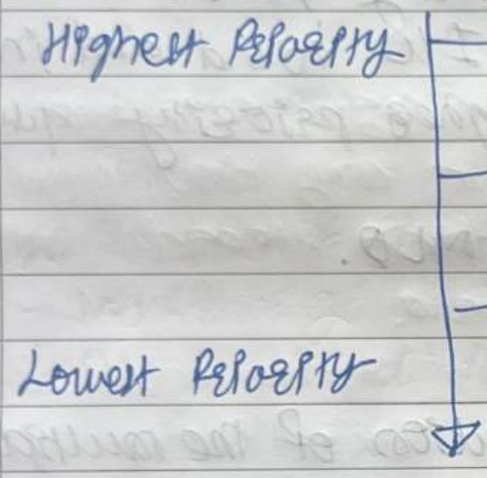
29. Round Robin,
a. FCFS preemptive
b. Criteria → AT + Time Quantum
c. NO process will wait forever, low starvation,
d. Easy to implement.
e. If TQ is small, more will be context switch (more overhead).



30. Multi-level queue Scheduling

a. Ready Queue is divided into multiple queues depending upon priority.

b. A Process is permanently assigned to one of the queues based on some property of process, memory, size, etc.



c. System Process (created by OS)
Interactive Process (need user I/O)
Batch Process (Runs silently).

d. Scheduling among different sub-queries is implemented as fixed priority preemptive scheduling.

e. If an interacting process comes & batch process is currently executing. Then batch process will be preempted.

f. Convoy effect is present.

31. Multi-level feedback queue scheduling (MLFQ),
a. Multiple sub-queues are present.
b. Allows the process to move between queues.
The idea is to separate processes according to the characteristic of there BT.
If 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.

c. Less starvation than MLQ.
d. It is flexible.

32. Concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process thread running in parallel.

- Thread
- 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 OS.

- Thread Context switching

- How each thread get access to the CPU?
  - Each thread has its own program counter.
  - Depending upon the thread scheduling algorithm, OS schedule these thread.

- I/O or TQ, based context switching is done, TCB (Thread control Block)

- Single CPU would gain multithreading?
  - Never
  - As 2 thread have to context switch for that single CPU.

- Benefits of Multithreading,
  - Responsiveness
  - Efficient Resource sharing.
  - Thread allow utilization of multiprocessor.

33. Critical Section Problem
- Process synchronization techniques play a key role in maintaining the consistency of shared data.

- Critical Section (C.S)
The critical section refers to the segment of code where Processes / threads access shared memory such as common files, variables, etc. & Perform write operation on that.

since, threads execute concurrently that leads to be interruption mid-execution.

- Major thread scheduling issue
  - Race Condition

A race condition occurs when two or more threads can access shared data & they try to change it at same time.

Because, the thread scheduling algorithm can swap between threads at any time you don't know the order in which threads will attempt to access the shared data

e.g
ADDNOR
DDTD

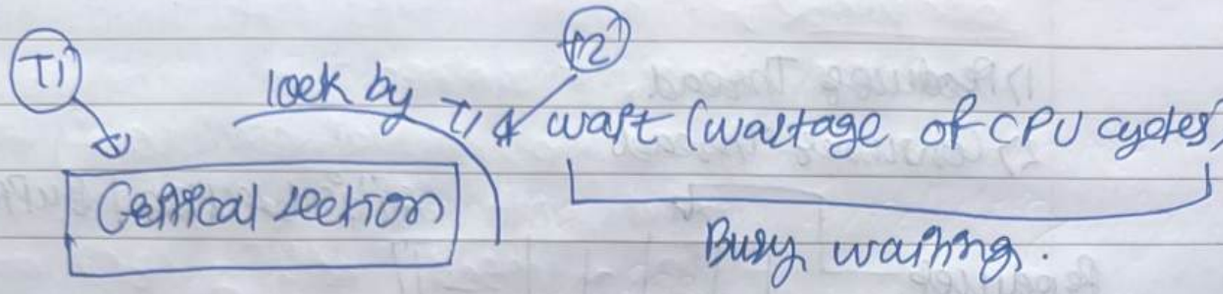
- Solution for Race Condn
  - Atomic Operation
  - Mutual Exclusion Locks (Mutex)
  - Semaphores.

- Can we use simple flag variable to solve Race Condn?
  → NO

- Peterson's soln can be used to avoid race condn but holds good for only 2 process / threads.

- Locks can be used to implement mutual exclusion & avoid race condition by allowing only one thread process to access critical section.

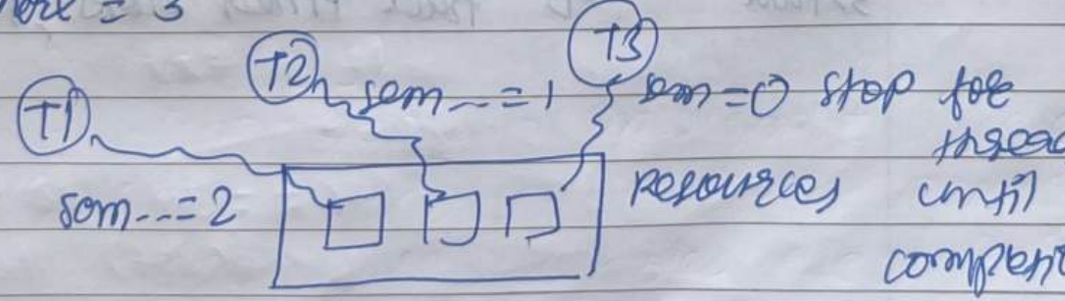
34. Conditional variables, Semaphores, for Thread



Conditional variables
- the condition variable is a synchronization primitive that lets the thread wait until a certain condition occurs.
- works with locks.
- Why we conditional variable?
  to avoid Busy waiting.

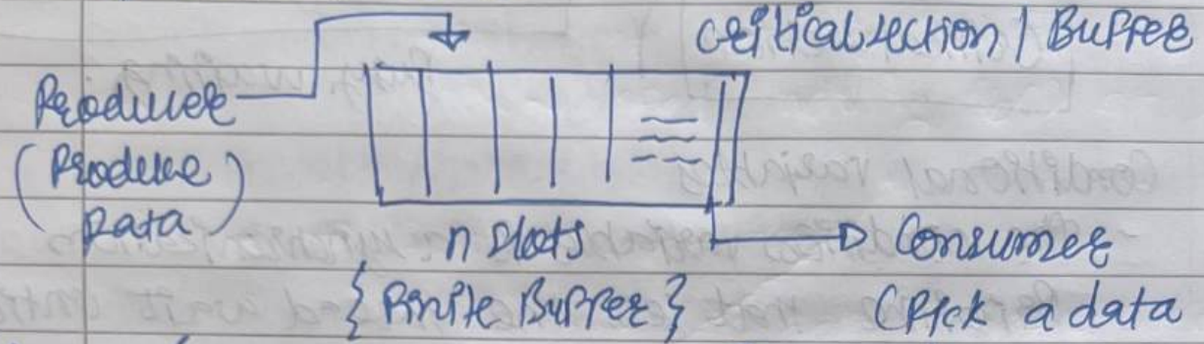
Semaphores
- Synchornization Method.
- An Integer that is equal to the number of resources.
- Multiple thread can go & execute C.S concurrently.
- Allows multiple program threads to access the finite instance of resources whereas mutex allow multiple threads to access a single shared resource at one time.

e.g
Semaphore = 3



35. Producer - Consumer Problem (Bounded Buffer Problem)

1) Producer Thread
2) Consumer Thread



Problem statement → { No 2 process interfere each other. }

→ How to get access of data synchronously?
→ Producer must not insert data when buffer is full?
→ Consumer must not remove data when buffer is empty?

Solution → Semaphores
1) m, mutex → Binary semaphore used to acquire lock on Buffer.
2) Empty → A counter semphore, initial value n tracks empty slots
3) Full → track filled slots, initial = 0

(mutual exclusion)
Achieve synchronization with using of mutex (variable).

__/__/____

Producer	Consumer
do {	do {
wait(empty); //wait until // empty > 0 then, // empty → value--	wait(full); //wait until full > 0, // then full--;
wait(mutex); //Locked	wait(mutex);
// CS add data to buffer	//remove data from buffer
signal(mutex);	signal(mutex);
signal(full); //increment full → value++	signal(empty); //increment empty
} while(1)	} while(1)

36. Reader-writer Problem

1) Reader thread
2) writer thread

Problem ① >1 Reader are allowed
② ~~>1~~ writer not needed only 1 writer at a time.

>1 writer or 1 writer & some other thread (R/W), Parallel,
→ Leads to Race Condition
→ Data Inconsistency

__/__/____

Solution

Semaphore :-

1) mutex, Binary Semaphore
   - to ensure mutual exclusion when read Count (rc) is updated.
   - NO. 2 thread modifying (rc) at same time

2) wrt → Binary Semaphore
   - common for both Reader & writer.

3) read count (rc) → Integer
   tracks how many readers are Reading in C.S.

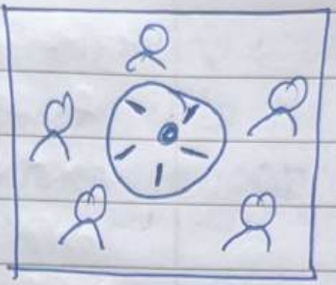
Writer

```
do {
  wait(wrt); //locked
  //do write operation
  signal(wrt);
} while(1);
```

Reader

```
do {
  wait( mutex); // to mutex readcount variable
  rc++;
  if(rc==1)
    wait(wrt); //ensure no writer can
               //enter if there is even
  signal(wrt);  // one reader
  // C.S Reader is Reading
  wait(mutex)
  rc--;
  if(rc==0) //no reader is left
    signal(wrt) ; // writer can write
} while(1);
```

37. The Dining Philosophers Problem



- we have 5 philosophers.
- Spend there life just being in 2 state
  - Eating
  - Thinking

- Thinking state, when philosphee thinks, he doesn't interact with other.
- Eating state, when philosphere gets hungry he tries to pick up 2 forks adjacent to him.
- one can't pick up a fork if it already taken.
- when philo. has both fork, he eats.

- Solution (semaphores),
  Each fork is a binary semaphore.
  A philo. calls wait() operation to get fork.
  Release fork by calling signal().
  Semaphore fork[5]={1}

- Although the semaphore solution makes sure that no two neighbour are eating simultaneously but could still create 'deadlock'.

- Avoid Deadlock,
  - allows at most 4 philo. to be sitting down.
  - allow philo. to pick his fork if they are available.

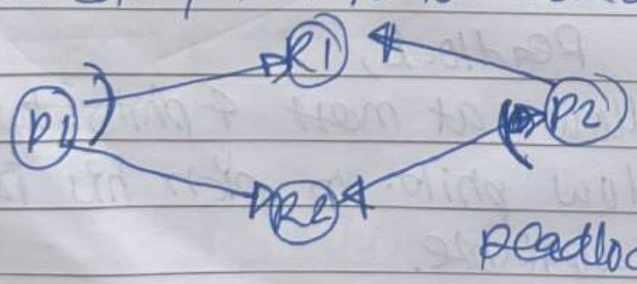
o odd-even Rule
an odd philo. Picks up first his left fork and then his right fork, whereas an even philosophee picks up his right fork then his left fork.

Hence, only semaphores are not enough to solve this problem.
Some enhancement needed like deadlock free solution.

38. DeadLock,
- In multiprogramming environment, we have several process competing for finite number of resources.

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

Conside P1 & P2 need Resource R1 & R2 at same time but any one resource is not available or locked, then it is called deadlock.



Deadlock

- Two or More processes are waiting on some resource availability, which will never be available as it is also busy with some other process. This Process called as 'Deadlock'.

- Example of Resources, Memory space, CPU cycles, files, locks, socket, etc.

⊕ 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 the resource instance.

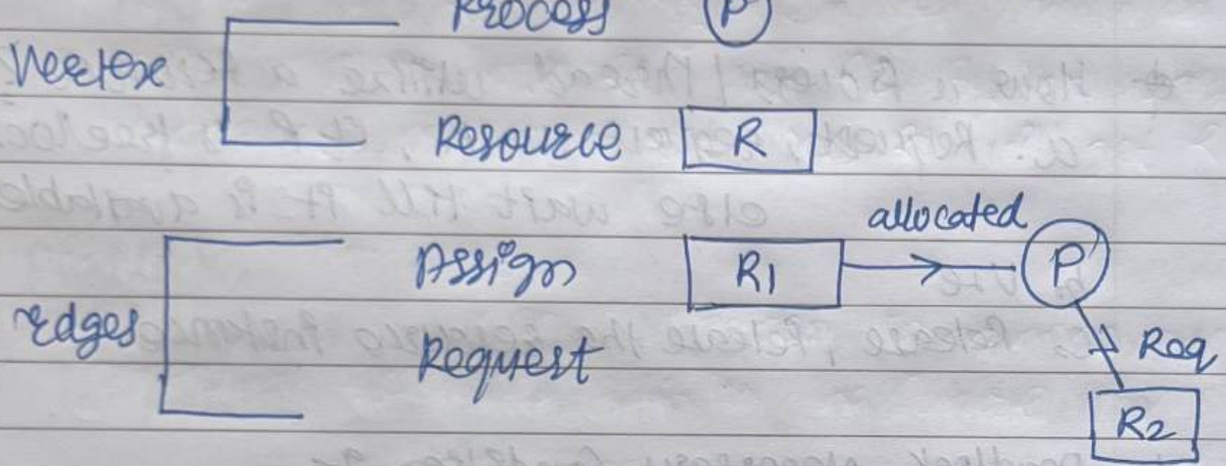
⊕ Deadlock Necessary Condition :-
a. Mutual Exclusion
Only 1 process at a time can use the resource, if another process request that resource, the requesting process must wait until the resources has been released.

b. Hold & wait
A Process must be holding at least one resource & waiting to acquire additional resource that are currently being hold by other ~~resource~~ process.

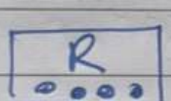
c. No Preemption
Resource must be completed work for holded Process, after completion then resources can be allocated to new Processes.

d. Circular wait
A set {P0, P1, Pn} of waiting Process must exist such that P0 is waiting for resources held by P1, P1 is waiting for a resource held by P2 and so, on.

⊕ Resources allocation Graph (RAG)

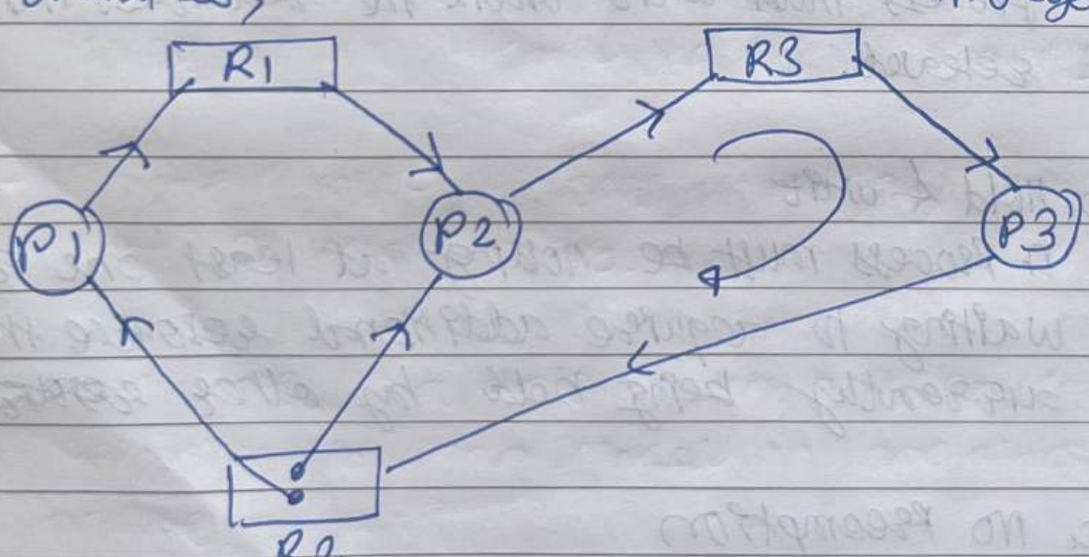


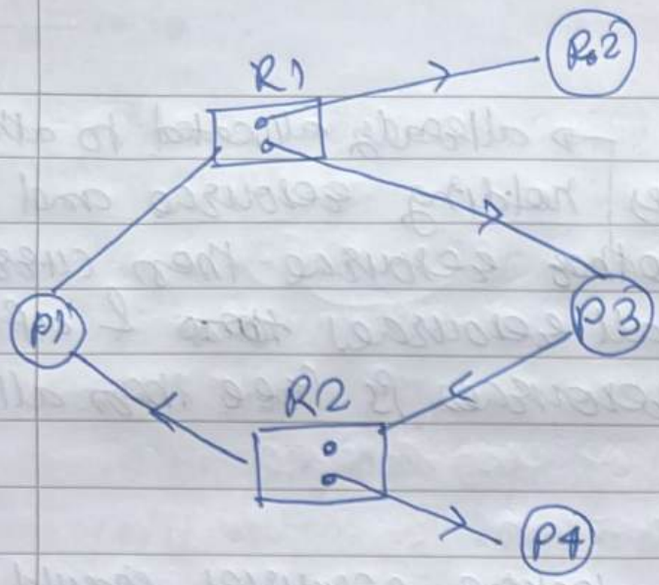
Multiple Instance



RAG | cycle (May be DL)
| No cycle (No DL)

Examples,





NO DL

- Method to handle DL
  - Use a protocol to prevent & avoid Deadlock.
  - Allow system to enter DL, detect it, recover it.
  - Ignore the problem altogether & pretend that deadlock never occures in system (Ostrich Algo).

- Deadlock prevention,

a. Mutual Exclusion
  - Use locks only for non-sharable Resource.
  - Sharable resource like Read only files can be accessed by process.
  - we can't prevent DL by denying the mutual exclusion condition because some resource are non-sharable.

b. Hold & wait.
  - To ensure H & W cond^n never exist in system, we must guarantee that whenever a process requests a resource, it doesn't hold any other Resource.

c. No preemption,
- If some Process holding resource and Request for another resource then current will release all resources & after when both resources is free then allocated. → already allocated to other

d. Circular wait,
P1 & P2 need R1, locking resources should be like both try to lock R1 then R2. By this way whichever process first locks R1 will get R2.

39. Deadlock Avoidance
- Kernel will be known information in advance about concern of resources we in the future processing.

- To decide whether the current request ~~whether the Process~~ can be satisfied or delayed, the system must consider the resources currently available, resources currently allocated to each Process in the system & the future Requests.

- Allocate Process & resources accordingly such way that DL never occur.

- Safe state, is a state in which system can allocate resources to each process in some order so DL not occur.

- In an Unsafe state, the operating system cannot prevent processes from requesting resources in such a way that any deadlock occur.

- The main key of the deadlock avoidance method, whenever the request is made for resource it will allow only if the resulting state is safe state.

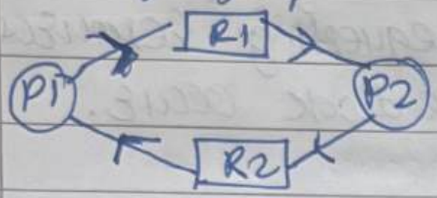
- If the system does not give resource then state is called Unsafe.

- Scheduling algorithm use to find safe state (Banker's Alg)

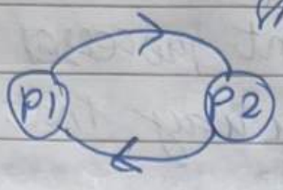
⊕ Banker Algorithm
When the Process request for resources, the system must determine the resulting state is in safe state then the Resources will be provided otherwise process goes in wait state.

⊕ Deadlock Detection
If the system have Implement Deadlock Prevention or Deadlock avoidance techniques. Then employ DL detection.
a. Single Instances of Resources
A deadlock exits in the system if and only if there is cycle in wait for graph.
To avoid DL, system need to maintain wait - for graph & invokes algorithm for searching cycle in graph.

wait for graph



shows the dependency.



wait for graph

b. Multiple instances for each resource
Use Banker Algorithm.

4 Recovery from Deadlock

a. Process Termination
- Terminate all Process
- Abort one process until DL cycle is eliminated.

b. Resource Preemption
Preempt the some Resources to eliminate DL.

40. Memory Management Techniques

a. In multiprogramming environment, we have multiple processes in the main memory to keep CPU utilization high.

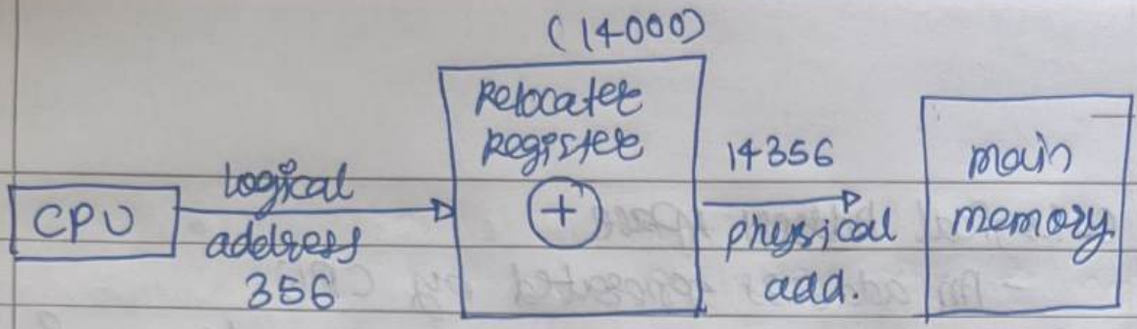
b. To fullfil process required, we must share the main memory, so we need to manage main memory for all different Processes.

c. Logical Address space
- An address generated by CPU.
- logical address is basically the address of an instruction of a process.
- User can access logical add. of Process. (Laddre.)
- User have indirect access of physical address through logical address.
- Logical address does not exist physically so known as virtual address.
- Range : 0 to max.

d. Physical Address space
- An address loaded into the memory address register of the physical Memory.
- User can never access physical address of Proces.
- Physical address is in the main Memory.
- It is computed by MMU.
- Range : (R+0) to (R+max), for base=R.

e. Runtime mapping from virtual to physical address is done Hardware device called MMU.

f. User program mainly generate the logical address, & the user thinks that the program run on the logical address, but actually program need main memory.

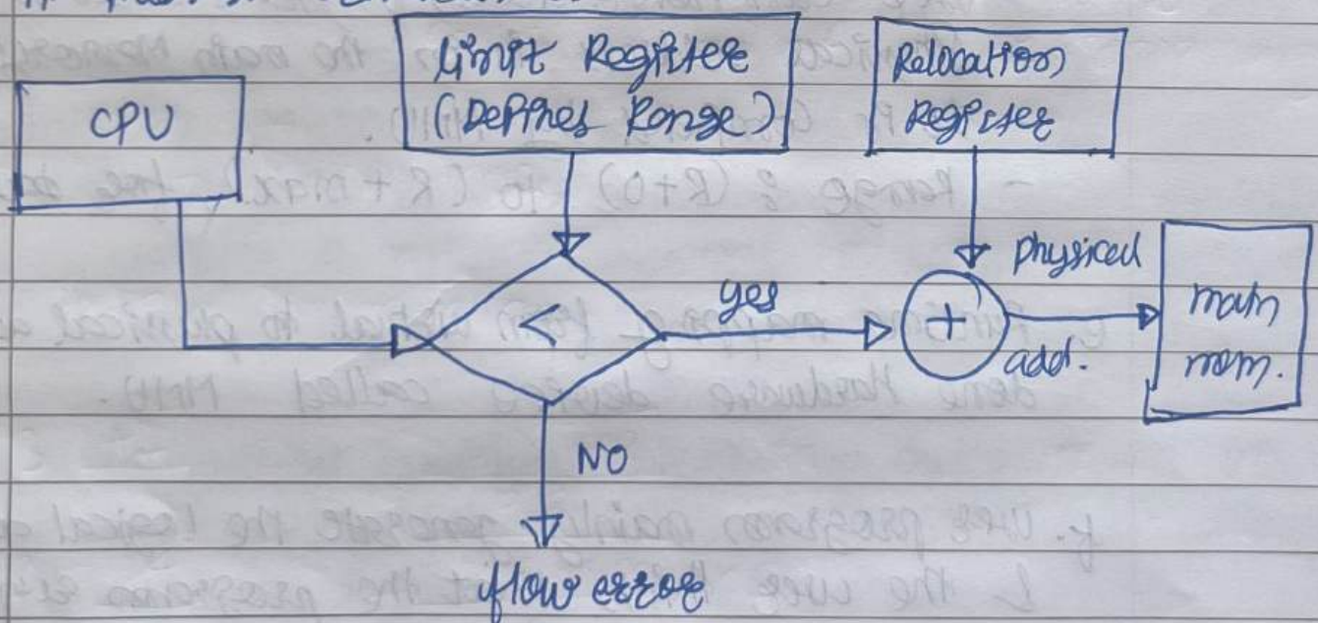


g. How OS manage Relation & protection of memory?

- OS provide virtual address space (VAS).
- to seperate memory space, we need ability to determine the range of legal address that the process may access to make sure that process only access legal addresses.

- the relocate & register contains smallest value of Base value (R) Basically defined Range of the respective processes.

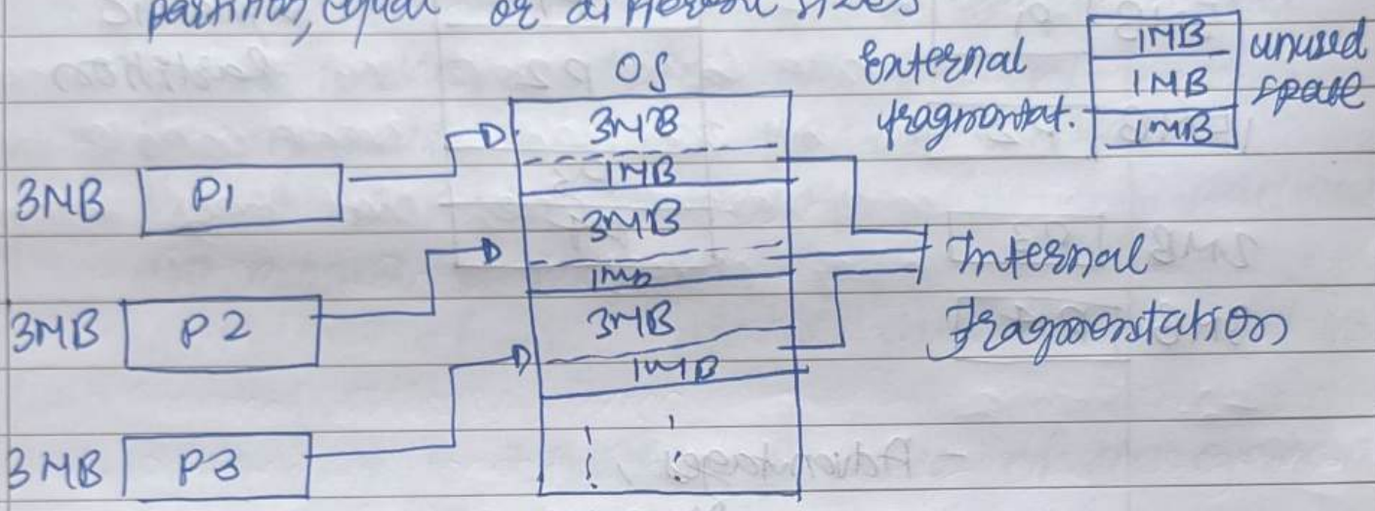
h. Address Translation



n. Allocation Methods

a. Contiguous Allocation

- In this scheme, each process is contained in a single contiguous block of memory.
- Fixed Partitioning, main memory is divided into partition, equal or different sizes



. Fixed Partitioning

- Limitation

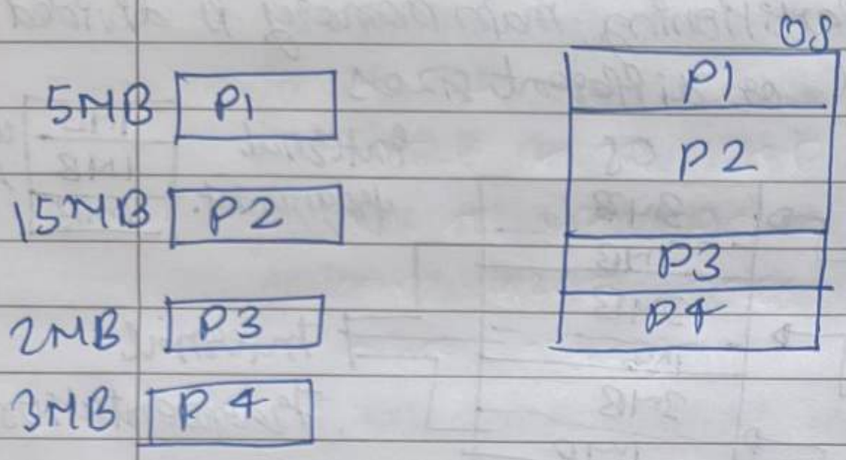
→ Internal Fragmentation, If the size of the process is lesser than the ~~big~~ total size of the partition then some size got wasted.

→ External Fragmentation,

→ If Process size is larger, partition can lead to take entire RAM (main memory.

b. Non-Contiguous allocation

- In this technique, partition size is not declare initially. It is declared at the time of process loading.

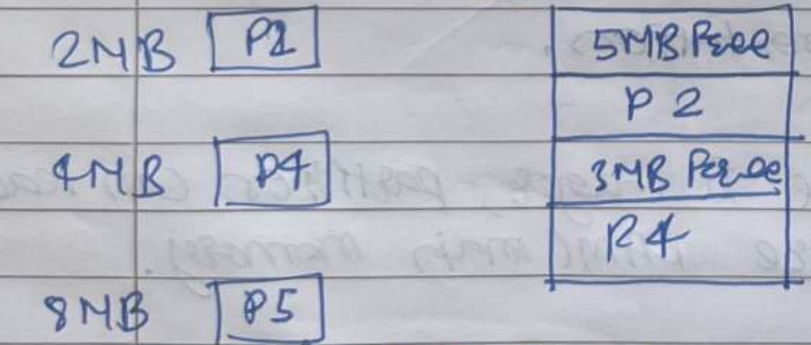


Dynamic Partition

- Advantages,
  - No internal fragmentation
  - No limit on size of process.
  - Better degree of multi programming.

- Disadvantage,

External Fragmentation



↳ Process can't be loaded in the memory even though there is 8MB space free (external fragmentation)

41. Reframentation / Compaction

- Dynamic Partitioning suffer 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 apply this technique, we can store the bigger processes in memory. The free partition are merged which can now be allocated according of new process called defragmentation.
- efficiency is decreased because of the overhead.

⊕ how free space is stored in OS,
by maintaining free list to calculate free holes in the memory.

⊕ To allocate free holes properly various algorithms are implemented,

a. First fit,
Allocate the first hole that is big enough.
Simple & easy to implement.
less time complexity.

b. Next fit
Enhancement of first fit but starts search always from last allocated hole.

c. Best fit

Allocate smallest hole that is big enough.
Leads internal fragmentation.
Slow; large time complexity.

d. Worst fit

Allocate largest hole that is big enough
Slow, as required to create whole free list.
Leaves larger holes that may taken be other process.

42. Paging

a. The main disadvantage of dynamic partitioning is External Fragmentation.

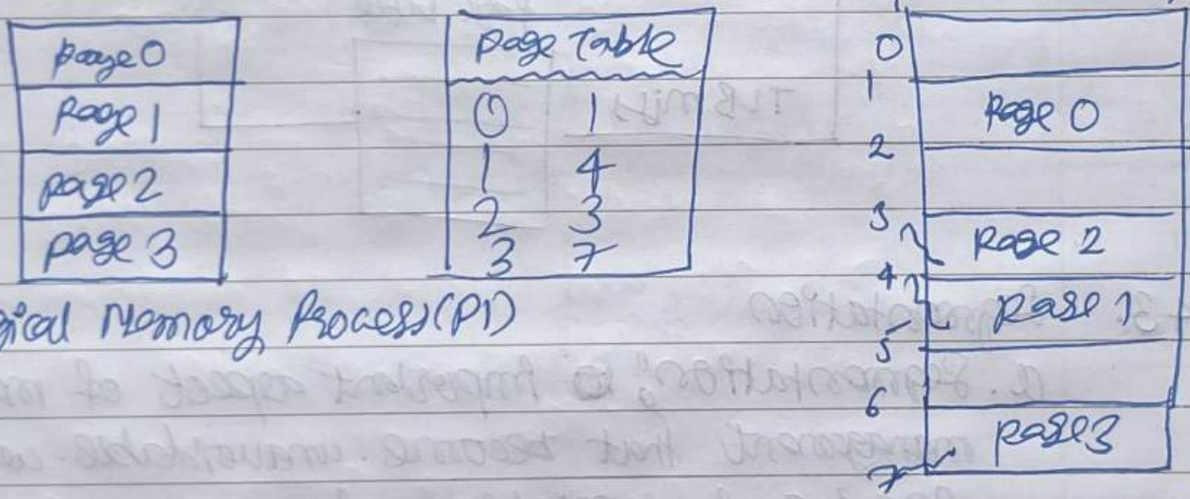
- Can be removed by compaction, with overhead.

b. Idea behind Paging,

- If we have only 2 small non-contiguous free holes in the memory, 1KB. each.
- OS wants to allocate RAM to a process 2KB in contiguous allocation, it is not possible as we must have contiguous memory space available 2KB.

c. Paging
- Paging is a memory-management scheme that permit the physical address space of a process to be non-contiguous.
- It avoids external fragmentation.
- Idea behind is to divide the physical memory into fixed-size blocks called frames, along with divide logical memory into blocks of same size called pages.

- Page table, a data structure that which page is mapped to which frame.

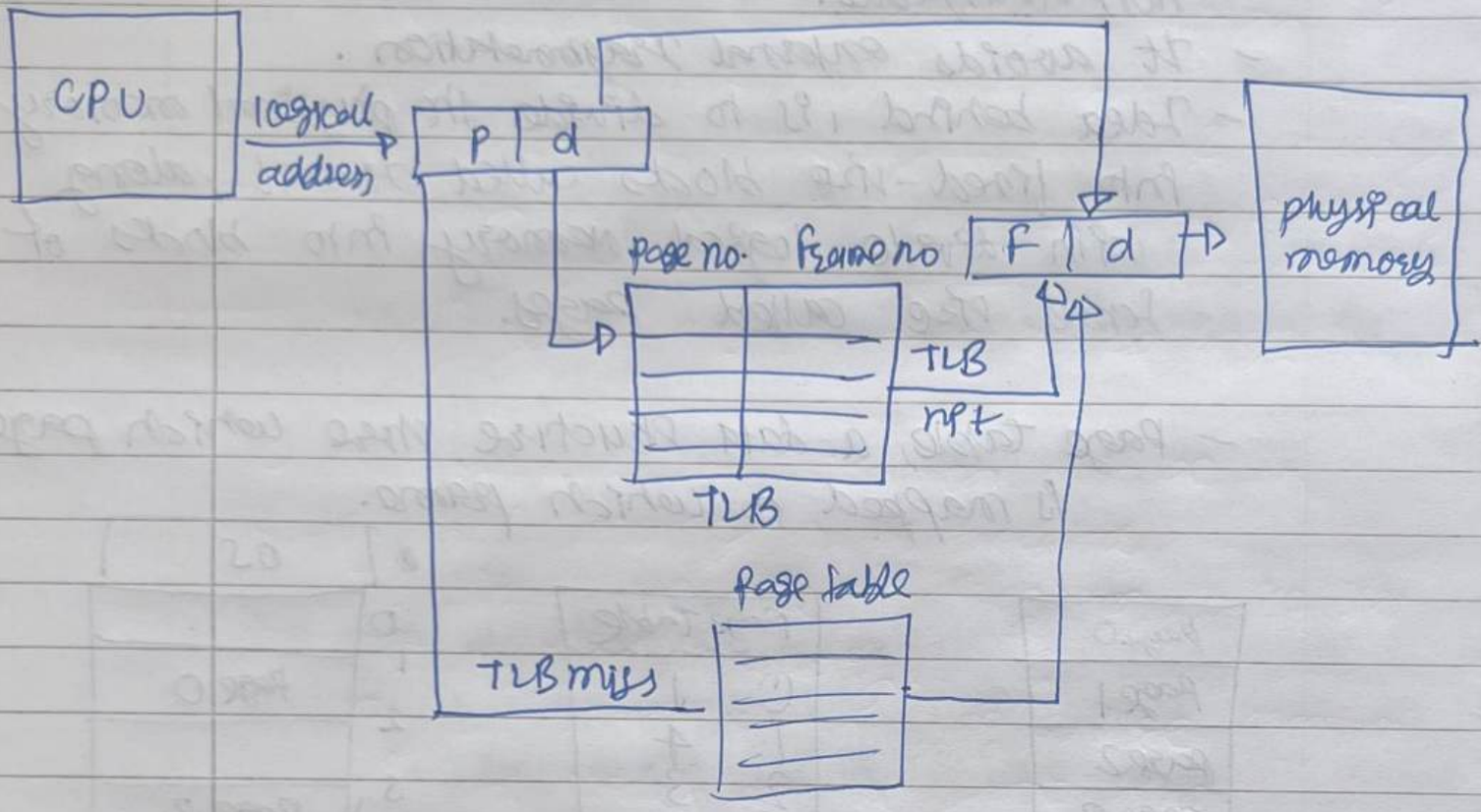


logical Memory Process (P1)

d. How Paging avoid fragmentation?
non-contiguous allocation of the pages of the Process.

e. Why Paging is slow?
There are too many memory references (Overhead).

f. Translation Look-aside Buffer
- Hardware support to speed up paging
- TBL has key & value.



43. Segmentation

a. Segmentation is important aspect of memory management that become unavoidable with paging is seperation of user's view of memory from actual physical memory.

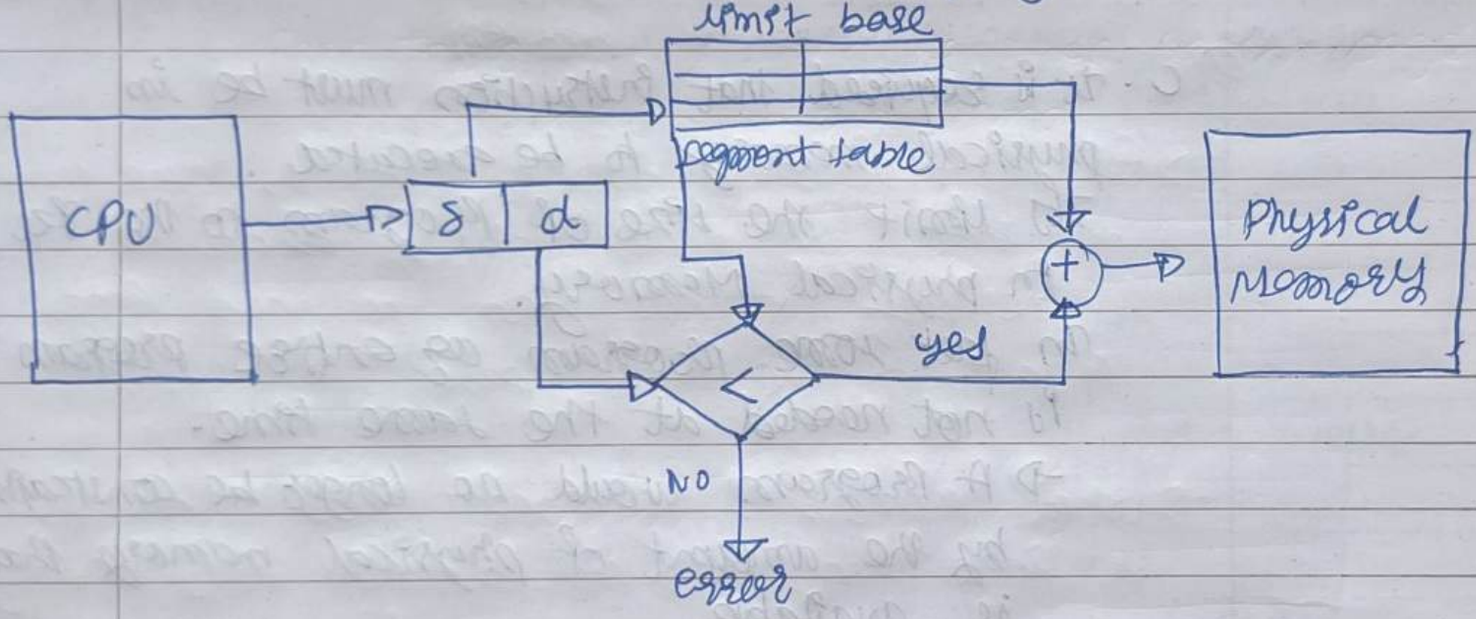
b. Segmentation is memory management technique that support view of memory.

c. logical address space is collection of segment,

seg → {segment-number, offset} {s, d}

Advantages ⇒ No Internal Fragmentation
- one segment has continuous allocation, efficient working with segment.
- size of segment less than size of Page table.

Disadvantages ⇒ External Fragmentation,
- Different size of segment is not good that time of swapping.



44. Virtual Memory

a. Virtual Memory is a techniques that allows the execution of processes that are not completely in the memory. It provides 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).

b. Advantage of this, Program can be larger than RAM.

c. It is required that instructions must be in physical memory to be executed.
To limit the size of Program to the size in physical Memory.
In fact some program or entire program is not needed at the same time.
→ A Program would no longer be constrained by the amount of physical memory that is available.
→ Because, each user Program could take less physical memory. more Program could run at same time (CPU utilization).

d. Programmer provide very large virtual memory when only a smaller physical memory is available.

e. Demanding / Demand Paging is a popular method of virtual memory management.

f. In demand Paging, the pages of a Process which are least used, get stored in the secondary memory.

g. Page is copied to the main memory when it's demand made. (Lazy swapper used to load the demanded Page in main memory).

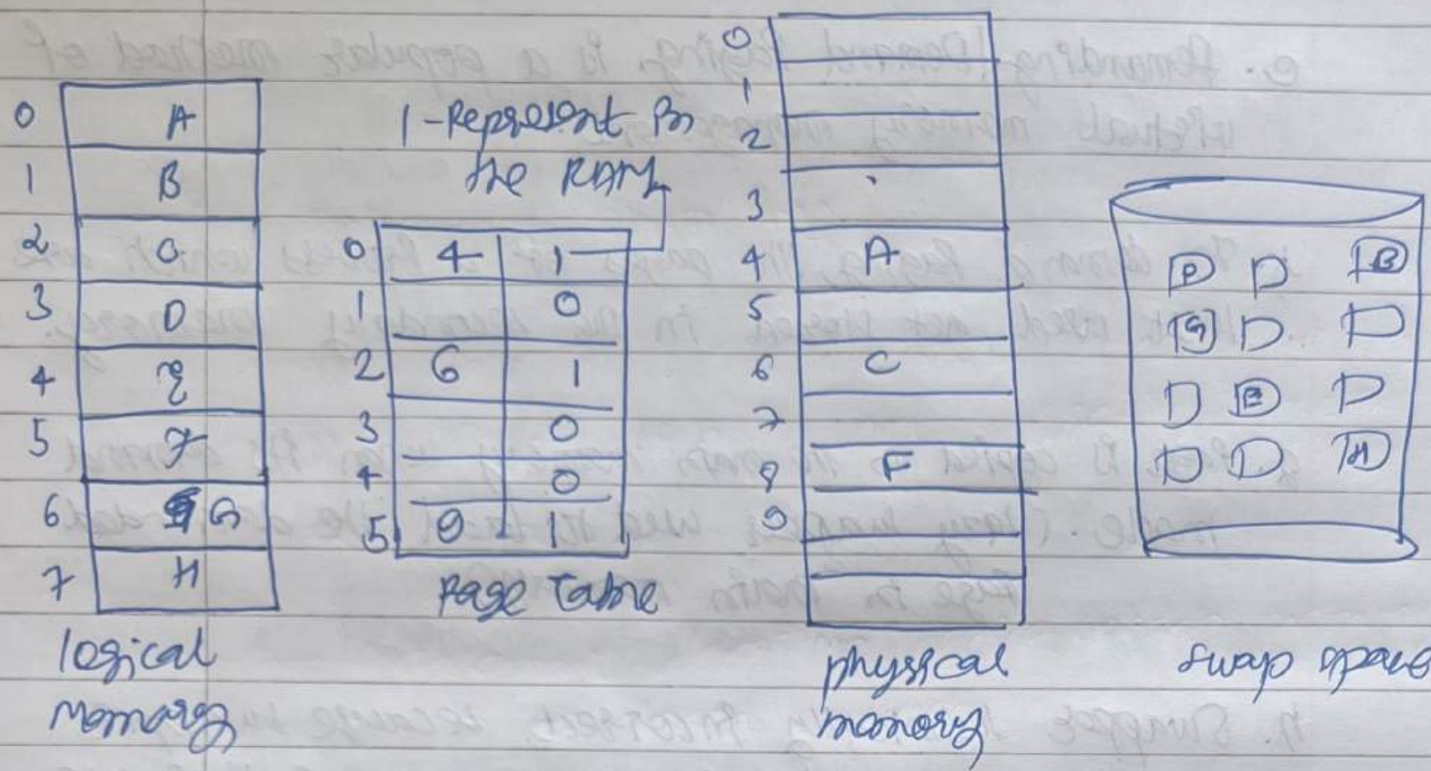
h. Swapper technically incorrect, because swapper help swap whole Process memory, Either we need some memory block of Process to use called 'Pager'.

* How demand Paging work?

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

→ Instead of swapping in a whole Process, Pager brings those Pages.

→ Above way, OS decreases the swap time & the amount of physical memory needed.



→ 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

→ what happens if the process tries to access a page that was not brought into memory, access to a page marked invalid cause Page fault.

~~Procedure to handle Page Fault:~~

Advantages of virtual memory	Disadvantages
- Degree of multiprogramming increase.	The system can become thrashing as swapping take time.
- User can run large app by using less physical memory	

Procedure to handle Page Fault,

① - Check an internal table to determine whether the reference was valid or invalid memory access.

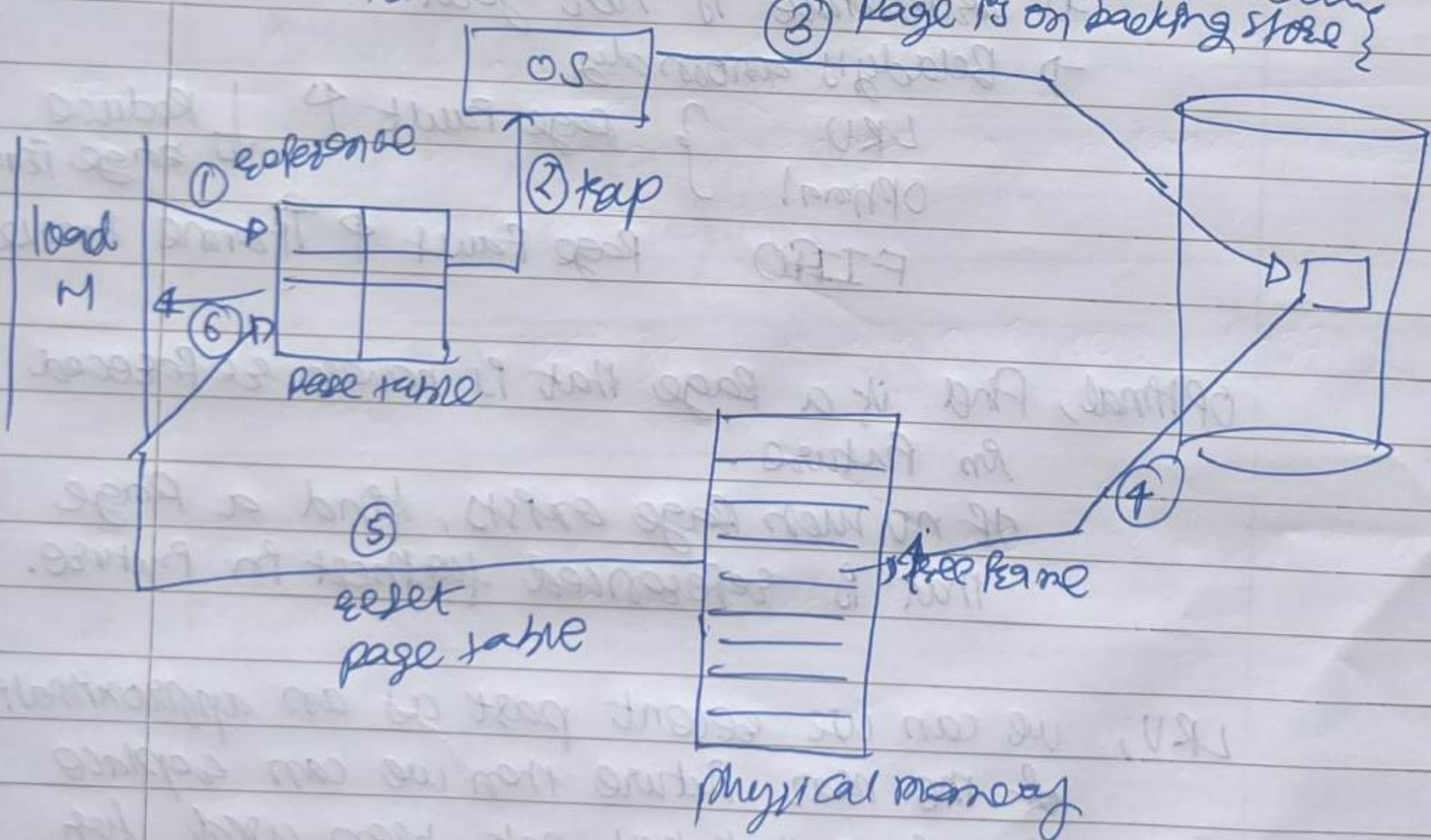
② - If ref. was invalid Process throws error
If ref was valid Page swap in the Page.

③ - Find a free frame.

④ - Schedule disk operation to read desired Page.

⑤ - When disk read is complete modify Page table.

⑥ - Restart the instruction that was interrupt by trap.



45. Page Replacement Algorithms

a. Whenever Page Fault occure that is Process tries to access a page which is not currently present in a frame.

b. Page Replacement algorithm decide which memory page is to be placed/replaced.

FIFO, → Allocate frame to the Pages as it comes into memory by replacing the oldest Page
→ Performance is not good.
→ Belady's anomaly

LRU, Optimal } Page Fault ↑ ↓ Reduce Page Frame
FIFO Page Fault ↑ ↑ Frame Increase

Optimal, Find if a page that is never referenced in future.
If no such Page exists, find a Page that is referenced farthest in future.

LRU, we can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.

LFU & MFU

46. Thrashing

a. If the Process does not 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,
Since all its Pages are in active use it must replace a page that will be needed again right away.

b. High Paging activity called Thrashing

c. System is Thrashing when it spend more time servicing the page faults than executing Processes.

d. How to Handle Thrashing,
Page fault frequency.

If the frequency is high we need more frames