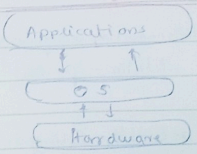
Main function of operating system must be
- provide maximum throughput (more no of process in limited time)
- provide convinience (user friendly)

DELTA
Date / /
Pg No.

~ operating systems ~

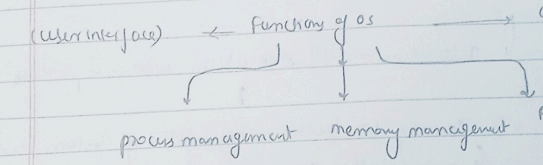
→ What is OS and some types of OS:-



↓

it is a software acting as an interface b/w computer, hardware and software (user applications)

↳ • manages resources • manages incoming user commands
• provides services for successful execution of programs



→ some types of OS- • Windows • macOS • Linux • Unix • Android
• IOS

→ RTOS (Real time Operating System) - Used for devices requiring deterministic and real-time response
↳ used in embedded systems, control systems, IOT devices

→ multi ~~programming~~ processing - Execution of multiple processes on a system with multiple CPU's or CPU cores, each process is an instance of a running programme and multiple processes can execute concurrently, each process having its own memory spaces and resources
↳ Aims to increase system's throughput and provides faster execution by dividing workload across multiple processes

* free multiple processes communicate with each other via IPC (inter-process communications)

→ Multithreading = • Involves executing multiple threads within single process

• Threads is a light weight unit of execution ↳ can run concurrently with other threads within same process

↓

(share same memory space and resources)

↳ allows parallel execution within a process

↓

leading to better utilization of system resources ↳ potentially improving performance by dividing tasks → into smaller units

↳ executed concurrently

(file handles and network connections)

→ Multiprogramming

↳ multiple programs are loaded into memory simultaneously

↳ CPU switches between them to execute instructions

↓

multiprogramming → leads to our CPU utilization

↓

keeping CPU busy by quickly switching between different programs when one is waiting for I/O or other operations

every program → having its own memory space

* multitasking - Allows multiple tasks or processes to run concurrently on a single CPU

↳ CPU time → divided among tasks, giving illusion of parallel execution

OS → switches between tasks rapidly, giving each task a time slice or quantum to execute

↳ used in modern OS to provide responsiveness and ability to run multiple applications simultaneously

* what is a program, process, thread?

(i) Program - Set of instructions written in programming language that performs a specific task or set of tasks

↳ Stored in a file on a disk → represents an executable entity

they can be compiled or interpreted

↳ Serve as a blueprint for execution of tasks on a computer system

(ii) Process - • An instance of a program

↳ when a program is loaded into memory and executed → process it becomes

process is an independent entity having its own memory space, resources and execution context

↳ has its own Program counter, Stack, variables

Provides isolation and protection between different instances of a program

* processes can be concurrent and communicate with each other through interprocess communication mechanisms

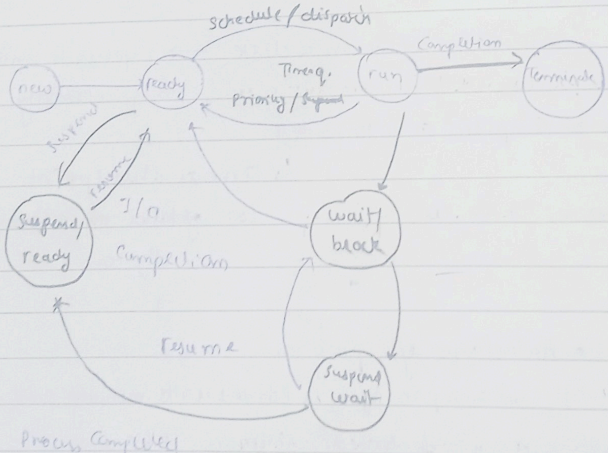
* Thread - It is a unit of execution within a process
  ↳ represents a sequence of instructions that can be scheduled and executed independently

Threads share same memory space and resources within a process. Since these process can run concurrently, they allow parallel execution of tasks

each thread is having its own stack and program counter

efficiency of using threads > than IPC (inter process communication)

Process states in operating systems -

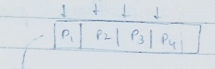


7 - State block diagrams for process is

Process completed I/O but still in suspend

→ From our new state, process gets created and is been forwarded to ready state via long term scheduler

Running state being a queue format accepts the request that has been first in put and removes as first output



↓

this removal (dispatch) of processes is known as Scheduling

FIFO

* CPU allocated to the running stage → now there are 2 conditions

(i) If process gets completed, it goes to the termination stage and process gets over

(ii) If process demands for I/O operation, it goes to wait / block, process is on hold, after completion of I/O request → this process is sent back to our ready state for execution

→ running state works on the theme of multitasking, process in running state is provided a fixed amount of time (time quantum) → since we are having a unique processor and no parallel execution possible

* moreover, the concept of priority also matters for a process, if any other process is been currently executed, it would execute prioritized process first

* Case of premption in running state:- when a process takes time T_1 for execution and if this time quantum $< T_1$ → execution stops, process gets forcefully removed and repeats itself for same time quantum after (wait) block

↳ if CPU runs a process without looking care of time

↳ non premptive process

Since the memory been used for ready / running State is our main volatile memory (RAM) if it runs out of memory to accomodate the process → it would send the process in suspend / ready state and fetch these process when there is enough space to accomodate those processes.

(suspend / ready) → is in ROM memory

Now since our running state as well as waiting / blocked is also in RAM memory

↳ If all the processes scheduled are been used for I/O requests read our secondary memory

↳ (waiting / blocked) gets filled

↓

a wait queue gets created also known as suspend / wait state

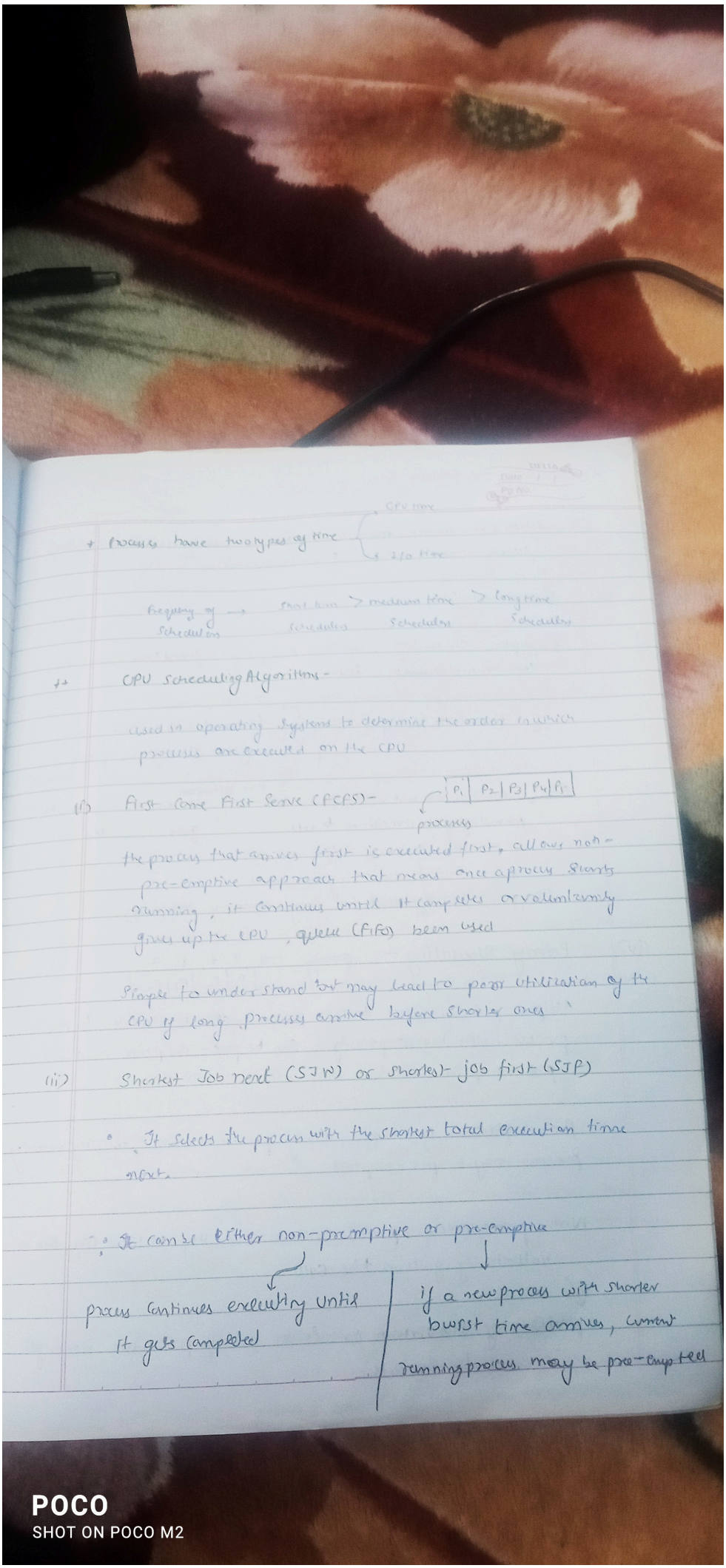
↓

processes can be taken or suspended to / from this block

+ Long term Scheduler - picks process from new state and put it in our ready state also deciding degree of multiprocessing / multiprogramming for long term scheduling (ready)

+ Short term Scheduler - for scheduling of our processes from ready state to running state, we use it

+ Medium term Scheduler - handles (wait / block) → (suspend / wait) states as well as (suspend / ready) and (ready state)



* Process have two types of time → CPU time, I/O time

Frequency of Scheduling → Short term Scheduler > medium time Scheduler > long time Scheduler

CPU Scheduling Algorithms -

used in operating systems to determine the order in which processes are executed on the CPU

(i) First Come First Serve (FCFS) - | P1 | P2 | P3 | P4 | P5 | (P1 = process)

the process that arrives first is executed first, allows non-pre-emptive approach that means once a process starts running, it continues until it completes or voluntarily gives up the CPU, queue (FIFO) been used

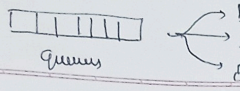
Simple to understand but may lead to poor utilization of the CPU if long processes arrive before shorter ones

(ii) Shortest Job next (SJN) or shortest job first (SJF)

- It selects the process with the shortest total execution time next.
- It can be either non-preemptive or pre-emptive

non-preemptive: process continues executing until it gets completed

pre-emptive: if a new process with shorter burst time arrives, current running process may be pre-empted



(v) Multilevel queue scheduling - dividing ready queue into multiple priority queues, each with its own scheduling algorithm

processes are initially kept (placed) in highest priority queue and can move between queues based on pre-defined criteria

→ This approach allows for the differentiation of processes based on their priority or characteristics

every queue can individually use a different scheduling algorithm, such as FCFS, SJF or RR suitable for processes within queue

(vi) Shortest remaining time (SRN) - / SRTF

A pre-emptive version of the Shortest Job first which is where the processor is allocated to the job closest to completion

↳ process with smallest amount of time remaining until completion is selected to execute.

(vii) Longest remaining time first (LRTF) / LRN

↳ a pre-emptive version of longest job first scheduling algorithm, used by operating system to program incoming processes for use in systematic way → schedules those processes first which have longest processing time remaining for completion

(viii) Highest response ratio next (HRRN)

↳ non-preemptive CPU scheduling algorithm, finding response ratio of all available process and select the one with highest response ratio.

** burst time → execution time

SJN/ SJF aims to minimize the average waiting time and is suitable when burst times are known in advance

(iii) Round Robin (RR) –

premptive scheduling algorithm assigning a fixed time quantum (eg 10 milliseconds) to each process in circular manner

if execution time of process
- > time quantum → moved back to the ready queue allowing next process to execute
- ≤ time quantum → gets executed

RR →
- providing fair execution to all processes but may suffer high context switching overhead
- (may not be efficient for long-running processes)

(iv) Priority Scheduling – priority → 4 3 1 2

P_1	P_2	P_3	P_4

Assigns priority value to each process, CPU is allocated to the process with highest priority, can be pre-emptive and non-pre-emptive

Pre-emptive – if a higher priority process arrives, currently running process may be pre-empted

Non-preemptive – process continues executing until it completes or voluntarily gives up the CPU

↓

problem of starvation for lower priority processes never get chance to execute

(PCB)

* What is a process control block -

It is a data structure used by an operating System (OS) to manage and control execution of processes. It contains all the necessary information about a process, including the process state, program counter, memory allocation, open files and CPU scheduling information

↳ Created by the OS when a process is created and is used to manage and control execution of that process

→ primary purpose of a PCB is to enable the OS to manage multiple processes efficiently by keeping track of the state of each process and allocating system resources accordingly

Process → Created → OS Creates PCB

↓

Stores all process info.

↓

OS then uses info. in PCB to manage process and ensure that it runs efficiently

** Roles of PCB

- Process Scheduling
- Process Synchronization
- Inter-process Communication

DELTA
Date / /
Pg No.

** Critical - Section problem :-

Critical Section can be said as a part of a program are where the shared memory is to be accessed → portion of code or block where process or thread accesses a shared resource

Shared resource can be - variable, file, database

↳ needs to be executed in an atomic or mutually exclusive manner

In order to maintain data integrity and avoiding conflicts
↳ only one process or thread should be allowed to enter Critical Section at a time

↳ ensuring no 2 processes interfere with each other, modify shared resources simultaneously preventing inconsistencies or incorrect results

```
while(true)
{
    entry section
    Critical Section
    exit section
    remainder section
}
```

* Solution to avoid Critical section problem

(Software)

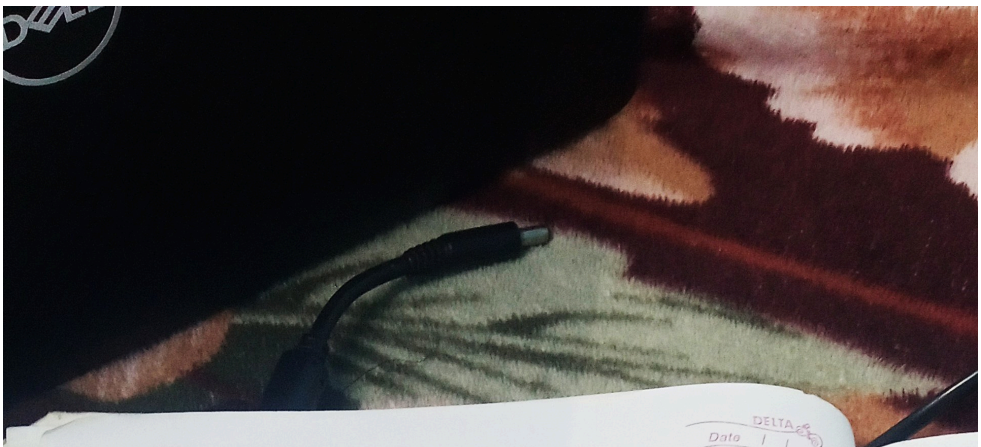
1. S/w Solution
   ↳ peterson's solution

(Hardware)

2. H/w Solution
   ↳ Synchronization hardware

Three problems that need to be satisfied to solve critical section problem

- mutual exclusion
- Progress
- Bounded waiting



* Remainder Section- remaining portions of the program excluding Critical Sections

* Race around condition- When our final o/p of the code depends on the order in which variables are accessed

↳ Race around condition

→ Process Synchronization ——→ (management of shared resources)

It is like a traffic signal helping to regulate the flow of vehicles at an intersection

↳ for computing stuff - refering to techniques and mechanisms used to coordinate the execution of processes and threads so that they can work together

↓ ensures multiple process / threads cooperate and communicate with each other to avoid conflicts and proper order of execution

↔ Avoiding conditions of race condition, deadlock, data inconsistency

* Solution for Critical section problem or say process Synchronization requirements:-

1. Mutual exclusion- If a process is already being executed in its Critical Section to access its shared resources, no other process is allowed to enter into the Critical Section

DELTA
Date / /
Pg No.

* Progress - If any process is not busy executing presently in its critical section, it can only only with the condition to use shared resources and enter the critical section after that

* Bounded waiting - There is a bound on the number of times other processes can enter into the critical section after a process has made a request to access the critical section and before request is granted

↳ every process would get a chance to enter the critical section without any chance of starvation

> Process synchronization mechanisms -

(i) Locks / mutexes - they provide a simple and effective way to achieve mutual exclusion

↳ allow only 1 process or thread to acquire a lock at a time, ensuring exclusive access to a shared resource or critical section

locks can be implemented using → hardware instructions
→ software constructs

(ii) Semaphores - They are synchronization objects that can be used to control access to shared resources. They can be implemented as binary semaphores or counting semaphores

→ Binary semaphore (0 or 1) - Used to implement mutual exclusion and synchronize concurrent processes

(ii) Counting Semaphore - A counting semaphore is an integer variable whose value can range over an unrestricted domain

↳ allows a specific number of processes or threads to access a shared resource simultaneously.

↳ provide mechanisms for mutual exclusion, signaling and coordination

(iii) Read-write locks - They provide synchronization mechanisms for scenarios where multiple readers can simultaneously access a shared resources without conflicts, but exclusive access is required for writers

~Types of operating systems~

(i) Batch O.S - Similar set of jobs stored in main memory for execution, a job gets assigned to CPU when execution for previous job completes

(ii) Multiprogramming O.S - main memory consisting of jobs waiting for cpu time. The os selects one of the processes and assigns it to CPU

↳ when executing process need to wait for any operation (like I/O)

↓

OS selects another process from job queue and assigns it to CPU

↳ CPU is never kept idle and user gets flavor of multiple tasks done

(iii) Multitasking O.S - Combines the benefits of multiprogramming OS and CPU scheduling to perform quick switches between jobs

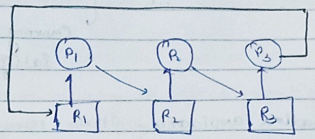
(iv) Time Sharing OS- these systems require interaction with the user to interact the OS to perform various tasks
↳ given through key board

→ OS responds with an o/p

(v) RTOS (Real time OS)- usually built for dedicated systems to accomplish a specific set of tasks within deadlines

* Deadlocks

A situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process



+ Conditions for deadlock -

(i) Mutual exclusion- A resource can only be shared in a mutually exclusive manner
↳ implies that two process cannot use same resource at same time

(ii) Hold and wait → process waits for some resources while holding another resource at same time.

(iii) No pre-emption — process once scheduled will be executed till the completion. No 2 process can be scheduled by scheduler.

(iv) Circular wait- processes must be waiting for resources in a cyclic manner so that the last process is waiting for the resource being held by first process

* Deadlock handelling techniques-

(i) Deadlock prevention- Aims to eliminate one or more of the necessary conditions for deadlock to occur. These conditions involve mutual exclusion, hold and wait, no preemption, circular wait

↳ ensuring that one or more of these conditions are not satisfied

(ii) Deadlock avoidance- Involves resource allocation algorithms and resource request protocols to avoid situations that may lead to deadlocks

↓

usage of RAG (resource allocation graph), banker's algorithm

↳ can predict whether granting a resource request will lead to potential deadlock → if yes then it is delayed for sometime

(iii) Deadlock detection- It involves periodically examining resource allocation state to determine if a deadlock has occurred

↳ can be achieved by resource allocation graph (RAG) and banker's Algo.

✓

if deadlock detected → system can take appropriate actions to resolve it-

↳ terminating processes, resource preemption, rolling back system to a safe state

(v)

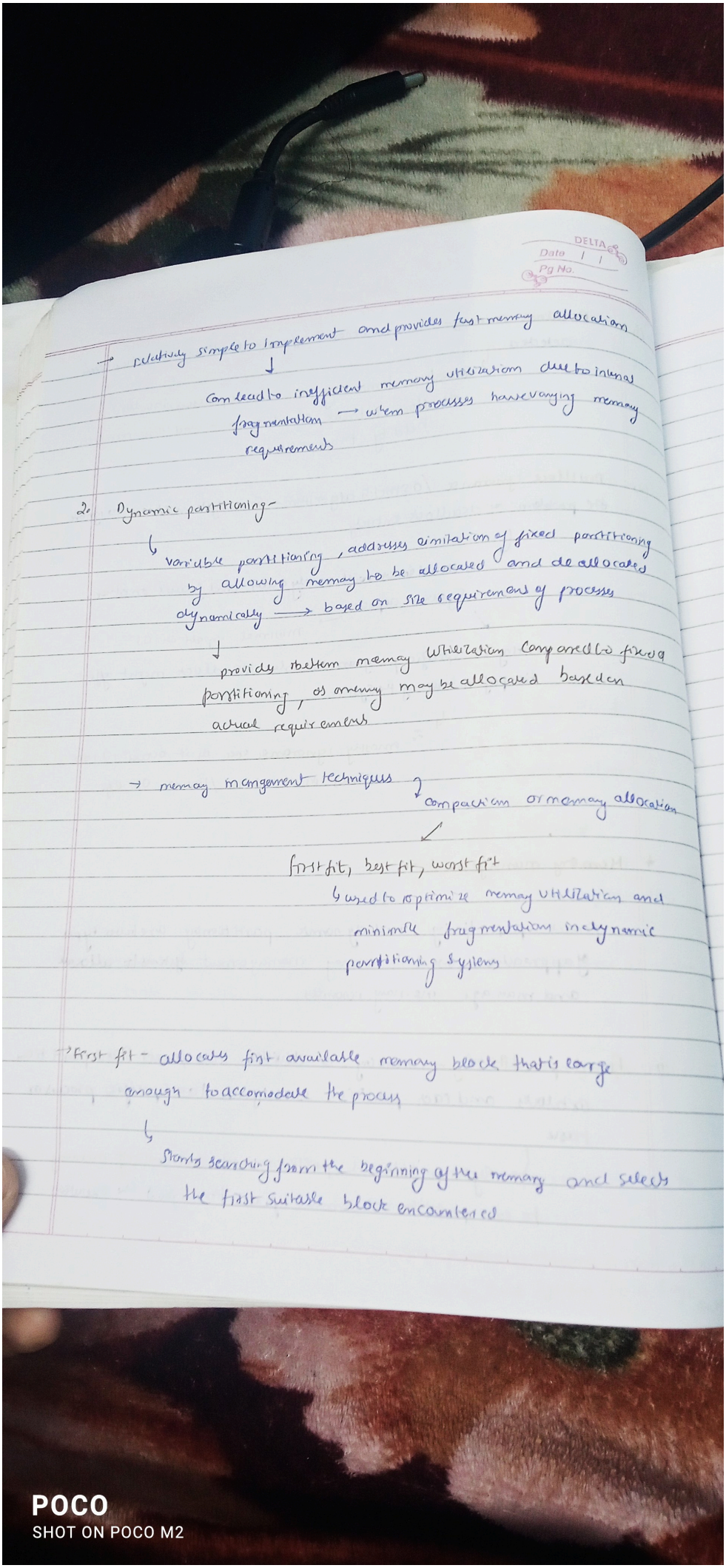
(iv) Deadlock recovery - used to recover from a deadlock after it has been detected
↳ involves terminating one or more processes involved in deadlock to break circular wait and release resources held by those processes

(v) Deadlock ignorance / OS with algorithm - Some systems choose to ignore the problem of deadlock entirely
↳ as deadlocks will rarely occur and focuses on other system aspects → minimal impact on OS
writing a specific program to solve deadlock might affect the performance of the system
↳ ∴ mostly ignorance due as it occurs 1/100 times in windows / Linux OS e.g.

* Memory management -

fixed partitioning and dynamic partitioning are two types of approaches used in memory management systems to allocate and manage memory resources

i) fixed partitioning - memory is divided into fixed sized partitions or blocks and each partition is assigned to a specific process or task
↳ system allocates predetermined amount of memory to each partition → remains fixed throughout the execution



→ relatively simple to implement and provides fast memory allocation
↓
can lead to inefficient memory utilization due to internal fragmentation → when processes have varying memory requirements

2. Dynamic partitioning-
↳ variable partitioning, addresses limitation of fixed partitioning by allowing memory to be allocated and de allocated dynamically → based on size requirement of processes
↓
provides better memory utilization compared to fixed partitioning, as memory may be allocated based on actual requirements

→ memory management techniques → compaction or memory allocation
↓
first fit, best fit, worst fit
↳ used to optimize memory utilization and minimize fragmentation in dynamic partitioning systems

→ First fit - allocates first available memory block that is large enough to accomodate the process
↳
Starts searching from the beginning of the memory and selects the first suitable block encountered

(ii) Best-fit – Searches for the smallest available memory block that is large enough to accomodate the process.

(iii) worst fit – Ordering process is allocated the hole of memory in which it leaves maximum gap.

* Paging

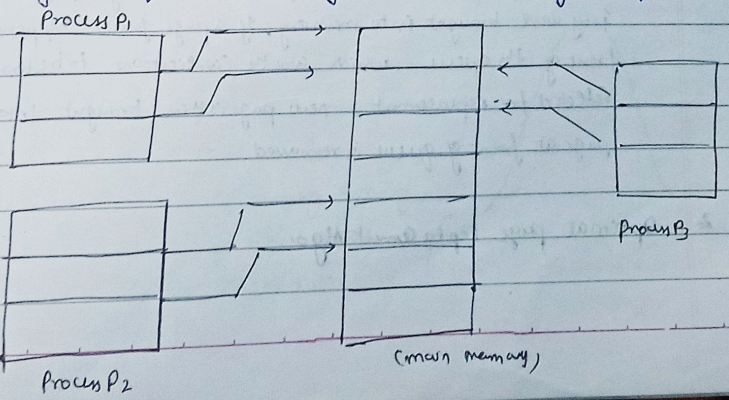
It is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages

main idea behind paging → divide each process in form of pages the main memory will also be divided in form of frames (pages)

One page of process is stored in one of the frames of memory

↳ the pages can be stored at different locations of the memory but priority is always given to contiguous holes or frames

Pages of the process are brought into main memory only when they are required or else they remain in secondary storage



* Virtual memory - It's a concept that lets computer use more memory than it actually has

↳ Creates imaginary memory space by combining physical memory (RAM) and secondary storage (like a hard disk)

↳ when programs needs memory > available RAM

↓

temporarily moves some data to secondary memory allowing longer programs and handlle multiple tasks at once

↳ provides memory protection → ensuring programs can't access each other's

* Using this degree of multiprogramming ↑s memory directly and CPU utilization ↑sed

--Page replacement Algorithm--

1. First in First out Page replacement Algo.-

Operating System maintains a queue or list of pages in the order they were brought into memory. If a page fault happens, page at front of the queue, which was the earliest one to be brought in, is selected for replacement. new page is then brought into memory and page at front of queue is removed

2. Optimal page replacement Algo.

Pages are replaced which would not be used for longest duration of time → providing best possible performance by reducing no of page faults

↳ practically not possible in real world scenarios → Knowledge of prior references also necessary hence impossible

(LRU)

3. Least recently used -

Page will be replaced which is least recently used

LRU works on principle that pages that have been recently accessed are more likely to be accessed again and again in near future

↓

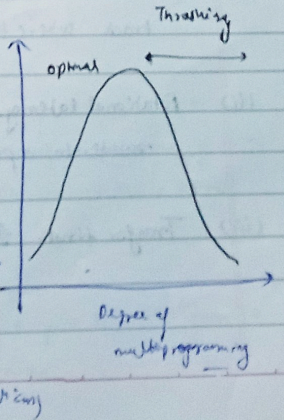
helps to retain frequently accessed pages in memory.

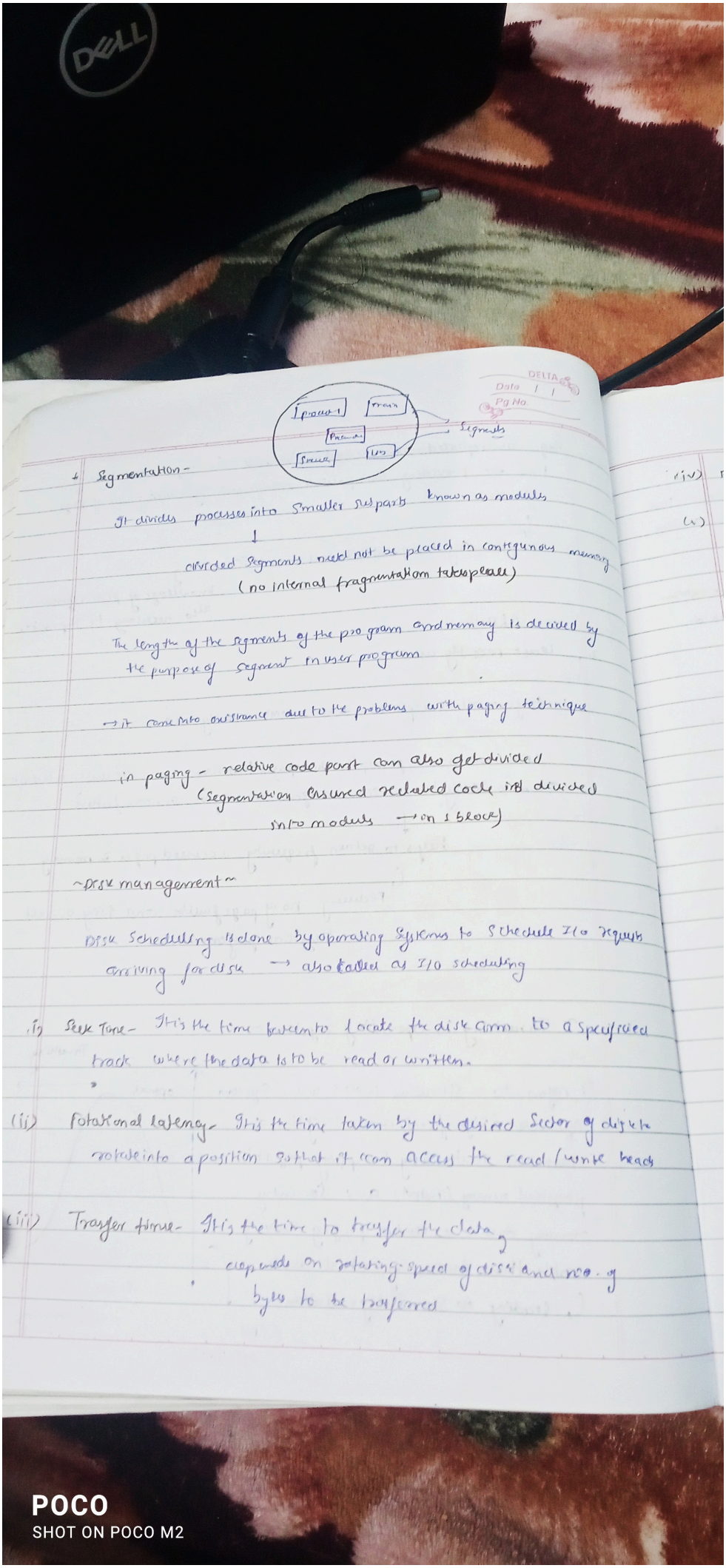
↳ reducing no of page faults and ↑ing overall system performance

~ Thrashing in Operating Systems ~

It refers to a situation in OS where system spends a significant amount of time and resources continuously swapping pages b/w physical memory (RAM) and secondary memory due to excessive paging activity rather than executing actual work

↳ leading to severe degradation





* Segmentation -

It divides processes into smaller subparts known as modules

↓

divided segments need not be placed in contiguous memory
(no internal fragmentation takes place)

The length of the segments of the program and memory is decided by the purpose of segment in user program

→ It came into existance due to the problems with paging technique

in paging - relative code part can also get divided
(segmentation ensured related code is divided into modules → on 1 block)

~ Disk management ~

Disk scheduling is done by operating systems to schedule I/O requests arriving for disk → also called as I/O scheduling

(i) Seek Time - It's the time between to locate the disk arm to a specified track where the data is to be read or written.

(ii) Rotational latency - It is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads

(iii) Transfer time - It's the time to transfer the data
depends on rotating speed of disk and no. of bytes to be transferred

(iv) Disk access time – Seek time + rotational latency + transfer time

(v) Disk response time – Average time spent by a request waiting to perform its I/O operation

↳ this is response time of all requests

– Disk Scheduling algorithms –

It is a key component of disk management that determines the order in which I/O (input/output) requests are processed and serviced by the disk controller

↳ minimizes the seek time and rotational latency and maximize overall disk performance

(i) FCFS (First Come First Served) – Simplest disk scheduling algorithm that processes requests in order they arrive

↳ does not considers position of disk head or proximity of requests,
→ can result to potential delays if a long seek time between requests.

(ii) SSTF (Shortest Seek time first) – this algorithm selects the request request with shortest seek time from current position of disk head → minimizes average seek time and reduces overall disk access time

↳ may lead to starvation of requests located farther away from current position

3- SCAN - also called elevator algorithm → moves the disk head in one direction (from outermost back → innermost or vice versa), and services requests along way, as it reaches the end → it changes its direction and continues same process

↳ provides fair distribution of service and prevents starvation
↳ may result in longer response times for requests at far ends of disk

4 C-SCAN (Circular Scan) → It also moves disk head in one direction but instead of changing (reversing) direction, it jumps to other end of disk and starts again

↳ Ensures more consistent response time for all requests
↳ it may cause delays for requests that arrive after head has passed their location

5 Look - It is a variant of SCAN, only goes as far as last request in its current direction, once there is no more requests in that direction, it reverses direction

↳ reduces unnecessary traversal of entire disk and improves response time for requests

6 C-Look (Circular Look) - Similar to C-SCAN, C-Look jumps to the other end of disk without servicing requests along the way

↳ reduces seek time and improves disk throughput