Bit vector
Bitmap where each bit represents a block, a bit value of 0 indicates the block is free, and 1 means it is allocated.

Linked list
Each free block points to the next available block.

Grouping
Blocks are grouped together, with each block in the group pointing to the next.

Directory Implementation
Linear list
↓
A simple list of file names which can be slow for large directories

Hash table
↓
uses a hash function to quickly find files, providing efficient search operations.

File System Structure
Framework the OS uses to organize & manage files.

1) User Interface Layer →
Provides user commands (eg. "Open-File")

2) Logical File System →
Manages metadata like file names, sizes, & permissions

3) Physical File System →
Manages storage blocks on physical disk.

Allocation Methods
how OS stores files on a disk :-

1. Contiguous Allocation →
files occupy consecutive blocks
adv. → fast access disadv. → leads to fragment.

2. Linked Allocation →
each block contains a pointer to the next.
adv. no fragmentation disadv. slow for large file

3. Indexed Allocation →
uses an index block to list all file blocks.
adv. → Random access disadv. → req. extra space for the index.

Free Space Management →
Free space on a disk needs to be efficiently managed.
Common techniques → bit vector
→ Linked List
→ Grouping.

File types

Text files	Binary files	Executable files
Human-readable data	machine-readable data	program files that can run on the system
(.txt, .csv)	(.exe, .jpg)	(.exe, .sh)

File Operations

1. Create - making a new file
2. Open - preparing a file for access.
3. Read / Write - Accessing / modifying file data
4. Delete - Removing a file from storage.
5. Rename - Changing the name of a file
6. Copy / Move - Duplicating / relocating a file.

Directory Structure

Directories (folders) Organize files into a hierarchical str. for easy management.

1. Single-level directory → all files stored in a single directory.
   → simple but inEffi for large systems
2. Two-level directory → each user has their own directory.
   → improves file organization
3. Tree-str. directory → files & directories are organized like a tree with folders & sub-folders.
   → most-common system.
4. Acyclic graph directory → allows sharing of files b/w users or directories.
5. General Graph directory → similar to acyclic but allows cycles (can lead to complexities).

Boot Blocks → contains necessary ~~data~~ code to store the OS, typically located at the beginning of the disk.

Bad Blocks → areas on the disk that are damaged & can't reliably store data. Disk management sw can mark bad blocks to avoid their usage.

② File Management –

- critical function of OS
- deals with storage, retrieval, organizⁿ. of files on comp.
- ensures data is stored efficiently, securely, easily accessible when needed.

Concept of a file

File - collection of related data stored on a storage device (hard drive, SSD, USB)

Purpose - files store data permanently for later retrieval, even after the computer is powered off.

ex - text files (.txt, .sv)
multimedia (eg .mp4, .mp3)
binary files (eg .exe, .bin)

File Access Methods

Sequential Access
data is read or written in order, one record at a time.
ex - Reading a text file line by line.

Direct Access
data is accessed directly at any location using an index or address.
ex - Jumping to a specific song in a playlist.

Indexed Access
uses an index to locate data efficiently.
ex -
Accessing a specific chapter in eBook

File Types
Text files
Human - readable data
(.txt, .csv)

File Operation
1. create -
2. Open -
3. Read / Wri
4. Delete -
5. Rename -
6. Copy / Mo

Directory
Directories
Str. for

1. Single -
2. Two - l
3. Tree
4. Acyc
5. Gre

Disk scheduling Algorithm
- determines the order in which disk I/O operations are serviced.
- goal is to minimize seek time & improve throughput.

① FCFS → disk requests are processed in order they arrive. While simple, it can lead to inefficient disk usage.

② SSTF (Shortest Seek Time First) → disk arm moves to the closest request, minimizing seek time but potentially leading to starvation of far-off requests.

③ SCAN → disk arm moves in 1 direc", servicing requests until it reaches the end, then reverses direction. (Elevator Algo.)

④ C-SCAN (Circular SCAN) → similar to SCAN but after reaching the end of disk, arm returns to beginning without servicing requests along the way.

⑤ LOOK - similar to SCAN, but arm only moves as far as the last req. in current direc", reversing direc" when no more request exist.

③ Disk Management-
Disk Structure ✓
Disk Scheduling - FCFS, SSTF, SCAN, C-SCAN, disk reliability, Disk Formatting, Boot-block, Bad blocks.

formatting
disk sche~~duling~~ → ability of a disk to store data by dividing it into tracks & sectors. Also sets up file system str like file tables & directories.

reliability
disk formatting → ~~preparing a disk to store data by dividing it into tracks & sectors~~ ability of a disk to function properly without failure. Factors affecting it includes → hw quality, usage patterns & env. cond?

Device-independent I/O sw.
→ Goal is to provide a layer b/w hw devices & user-level sw, ensuring that
→ applications can operate without knowing the specifics of hw.

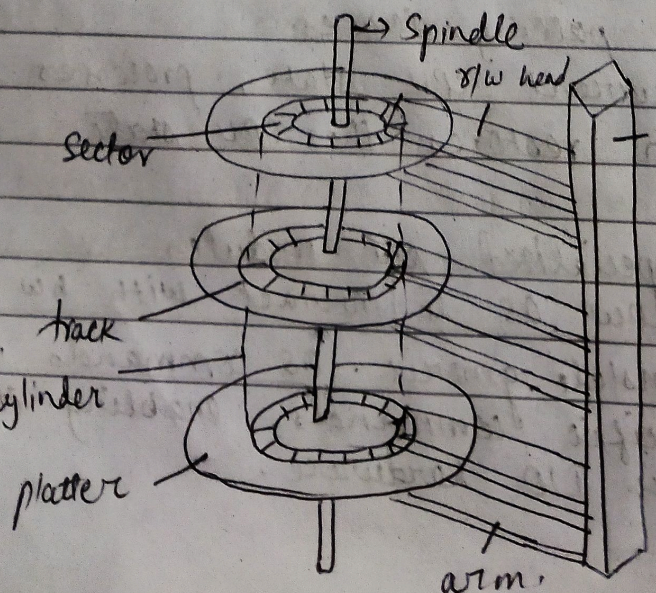
→ This layer includes -
I/O scheduler:- Decides the order in which I/O operations are executed.
Buffering:- stores data temporarily to optimize I/O performance.
Spooling:- Manages the execution of I/O operations (printer jobs) in a queued fashion.

Secondary Storage Structure
→ used for long-term storage
→ non-volatile
→ hard drives, SSDs.
→ slower than primary storage (RAM)
→ have much larger capacity.

Disk Structure

- disk consists of several platters, each containing tracks which are further divided into sectors.

- Platters are mounted on a spindle, each track contains a fixed no. of sectors where data is stored.



DMA controller → specialized hw component that facilitates DMA transfers by controlling memory access & notifying the CPU when the data transfer is complete.

Principles of I/O software →
- responsible for managing I/O devices & enabling the OS to communicate with hw.
- abstracts the hw complexity & provides a uniform interface for applications.

Goals → Efficiency → minimize delays & maximize throughput for I/O operations.
Reliability → ensure error-free comm? & data transfer
Flexibility → provide uniform interface for diverse I/O devices.
Device independence → hide the specifics of hw from applications, allowing sw to operate across diff. I/O devices.

Interrupt handler →
- signals sent by I/O devices to CPU indicating device is ready for comm? (I/O ready) or needs attention (error, completion).
- sp. routines that respond to interrupts.
- allow the system to efficiently manage I/O without constantly polling devices.
- saves current CPU state, processes the interrupt, and then restores the CPU state

Device drivers → specialized sw modules
→ allow OS to interact with hw devices.
→ translate generic OS commands into device-specific commands, enabling communication with I/O hardware.

Module (6)

(1) [I/O Hardware]

- refers to devices & controllers responsible for communi. DMA controlle
  b/w the computers & its external environments.
- these devices enables the user system to receive input
  from users & send outputs.

I/O device →

Input device - devices that allow users to input data into the system. (keyboard, microphones, scanners)
Output device - devices that allow the system to present data to the user, (monitors, printers, speakers).
Storage device - devices that store data (hard drives, USB drive

Device controllers →

i) hw component that manages the operation of I/O device and acts as interface b/w the device & computer system.

ii) converts High-level commands from CPU into specific signals that control I/O device.

iii) components → control registers - store info. like commands & status for the device.
Buffer - temporary storage for data being transferred b/w device & memory.

iv) ex. disk controller manages hard drives.

Direct memory access → (DMA)

→ allow certain hw subsystems to access main memory independently of the CPU.
→ used for high-speed data transfer b/w I/O devices & memory without CPU's intervention
→ Increasing system's performance.

Principles o
- respon
the O
- abstrac
interfa

Goals →

Devic

Interrup
↓
- signa
indic
atten

Devic

Page Replacement Algorithms (PRA)

When a page is needed but phy. memory is full, OS must decide, which page to replace

→ ① Optimal Page Replacement
② FIFO
③ Second chance (SC)
④ Not Recently used (NRU)
⑤ Least Recently used (LRU)

① best PRA
- replaces the page that will not be used for longest prd. of time in future.
- not feasible in practice.
- req. future knowledge of memory access.

② simplest PRA.
- oldest page in memory (which was loaded first) is replaced when new page is needed.

disadv. → lead to poor performance
Belady's Anomaly (↑ no. of frames ⇒ ↑ page fault)

③ - FIFO with modific".
- each page has a ref. bit.
- When page is selected for replac., the ref. bit is checked.
if it is set to 1 ⇒ given 2nd chance.
if it is 0 ⇒ page is replaced.

④ → based on ref. bit & modified bit.
→ principle of replacing pages that have not been recently referenced / modified.

⑤ → page not used for longest time - replaced.
→ principle - temporal locality → pages used recently are likely to be used again soon.

disadv. maintaining LRU list can be expensive in terms of processing time.

Locality of reference

→ processes tends to access a small portion of memory at any given time.

→ 2 types ⟷ Spatial → tendency of a prog. to access memory locations that are physically close to each other in a short time period.

⟷ Temporal → tendency of a prog. to access the same memory locations repeatedly within a short time period.

→ allows OS to keep frequently accessed pages in phy. memory & swap out the less frequently used pages.

Page fault

→ occurs when a prog. accesses a page that is not currently in physical memory.

→ when this happens, OS must load the page from disk into memory.

→ slow down the system - high latency of disk access compared to memory access.

Working set - Set of pages, the process is currently using

↓ if it's too large for available memory

page faults will occur frequently.

Dirty page / Dirty bit -

Page that has been modified but has not yet been written to disk.

→ flag that indicates whether a page has been modified. If a page is marked as dirty, it needs to be written back to disk before it can be replaced.

Demand Paging

- pages are only loaded into memory when they are needed (on demand), reducing memory usage.
- When a page is accessed, & the page is brought into memory. page fault occurs

Protection and sharing

↓

OS ensures that a P can't access memory that belongs to other Ps by checking the page table.

↳ Multiple Ps can share pages of memory. For ex- code shared b/w Ps might reside in same phy. page frame.

Disadvantages of paging →

Overhead → it req. add. memory for page tables & time for add. translation.

Int. frag.m. → paging eliminates ext. fragn. but int. fragn. may still occur if a process doesn't fit perfectly into a page.

⑤ Virtual Memory.

→ allows execution of processes that may not be completely loaded into phy. memory.

→ uses the concept of paging to provide illusion of a large, contiguous memory space.

Basics of VM

- enables to access more memory than is physically available by swapping parts of programs (pages) in & out of disk storage.

- enables processes to execute in VAdd. Space & OS manages the translation of virtual addresses to phy. add.

Hardware & control structures

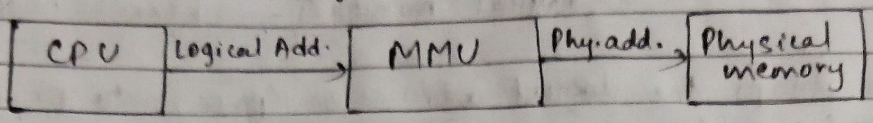
Page Tables

OS maintains page tables to map VA to PA

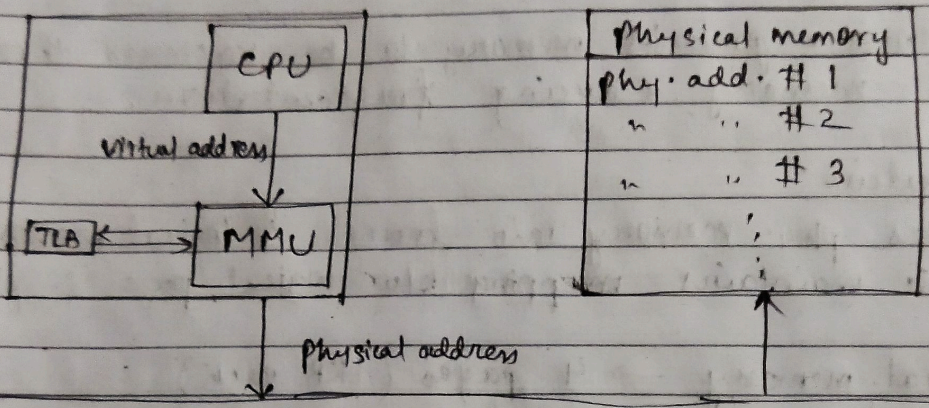
TLB

Cache used to speed up address translation.

Hardware support for paging
- MMU support paging.
- translates logical add. to phy. add. using page table.
- uses base registers & limit registers to keep track of boundaries of the page & the frame.



MMU:-



CPU → communicate with MMU to manage memory.
Virtual memory → abstract memory space that OS presents to processes (larger than phy. memory)
Physical memory (RAM) → actual hardware that stores data, prog., system info.
Page Tables → a DS used by MMU to map VA to PA
TLB (Translation lookaside Buffer) → cache that stores recent virtual-to-phy. add. translations to speed up memory access
Address Bus → commun. pathway that carries addresses b/w CPU, MMU & memory.
Memory Protection → a prog. can't access memory locations that are not allocated to it.

Protection
↓
OS ensures that can't access belongs to checking the

Disadvantage
overhead →

Int. frag.

Virtual
→ allows loaded
→ uses
contig

Basics
- enables
swapping

- enables
the

Hardware

Page
OS ma
map

Principle of operation :-

- In paging, phy. memory is div into fixed-size blocks called pages, & logical memory is div into blocks of same size called frames.

Paging → Physical memory → Pages (fixed size blocks)
→ logical memory → Frames (blocks of same size)

- Page table maps the logical addresses (pages) to physical addresses (page frames).

- This allows a process's memory to be scattered throughout physical memory, reducing fragmentation.

Page Allocation :-

OS divides phy. memory into frames & logical memory → pages. page table maintains mapping b/w logical pages & physical frames.

ex:- Virtual memory → 4 pages (1KB each)
Physical memory → 3 frames (1KB each).
Demonstrate how OS manages the mapping of virtual pages to physical frames.

⇒ ① Virtual address space (Pages)

Page 0 (0-1023)
Page 1 (1024-2047)
Page 2 (2048-3071)
Page 3 (3072-4095)

Physical address space (Frames)

frame 0 (0-1023)
frame 1 (1024-2047)
frame 2 (2048-3071)

② Page table

Virtual page	Physical frame
Page 0	frame 1
Page 1	frame 2
Page 2	frame 3
Page 3	(not allocated)

Memory Allocation

- process of assigning memory to processes.

① Contiguous memory allocation.

- each process is allocated a single contiguous block of memory
- Simple method
- suffers from fragmentation.

- Fixed Partitioning → memory is divided into fixed-size partitions, each partition holds exactly one process.
  - adv: simple to implement
  - disadv: wastes memory if process size is smaller than partition size.
- Variable Partitioning → memory is divided dynamically based on process req.
  - adv: more flexible & efficient memory usage.
  - disadv: leads to fragmentation.

Fragmentation

- Internal " → Occurs when allocated memory block is larger than memory requested by the process, leaving unused memory within the allocated block.
- External " → Happens when free memory is scattered in small blocks throughout the system, making it difficult to allocate large blocks of memory

Compaction

→ process of moving processes in memory to eliminate external fragmentation & create large contiguous free memory blocks.

→ costly in terms of time & resources.

Paging

- memory management scheme.
- eliminates the need for contiguous memory allocation.
- solves the issue of ext. fragmentation.

Principle of operation

→ In paging, pl
pages, & logical
frames.

Pa

→ Page table
addresses

→ This allows
physical

Page Allocatio
OS divides
page table

ex:- Virtual
Physica
Demonstrat
to physic

⇒ ① Virtual
Page
Page
Page
Pag

② Page

24/12/24

[Module - 5]

① Memory Management

- crucial aspect of an OS
- managing the computer's memory resources efficiently.
- ensures - process have sufficient memory space to execute
- provides mechanism for protecting the memory.
- optimizes memory usage.

[Basic concept of memory management]

① Allocating memory to processes.
② Tracking memory usage
③ ensuring that processes don't interfere with each other's memory (protection).
④ maximizing efficiency in memory usage
⑤ freeing memory when no longer needed.

[Logical & Physical Address map] → key concepts in MM.

① Logical Address
- generated by CPU during program execution.
- represents the address seen by the program
- not directly related to physical ~~address~~ memory.
- reference to a location in virtual memory.
- OS uses virtual memory management tech. to map logical add. to physical add., allowing programs to use more memory than is physically available.

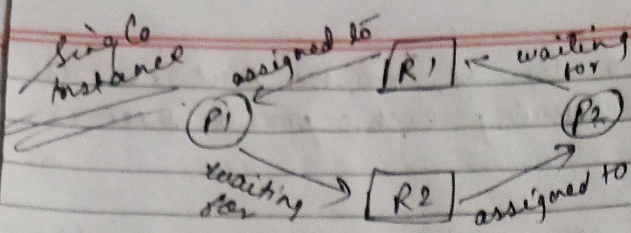
② Physical Address
- actual location in computer's physical memory (RAM)
- address that the memory hw (RAM) uses to access a specific memory location.

Address Translation (Virtual to Physical)
→ OS uses MMU (MM unit)

- Page Tables → os maintains page tables to keep track of which virtual addresses corresponds to which physical addresses.
- Segmentation & Paging → In syst. using paging, the virtual addess is divided into pages & phy. addr. div. into frames, each page is mapped to a frame in phy. memory.

Process represented as circle
Resources " as square.

24/12/24



• P1 → R2 → P2 → R1
↓
cycle is created.

Detection → resource allocation graph.
→ wait for graph.

Scenario:- Process P1 holds R1 & Req. R2
P2 holds R2 & Req. R3
P3 holds R3 & Req. R1

RAG ⇒ P1 → R2 → P2 → R3 → P3 → R1 → P1
WFG ⇒ P1 → P2 → P3 → P1

RAG
•) directed graph.
•) nodes represent processes & resources edges represent resource allocation & request relationships.

WFG :
•) simpler graph
•) nodes represent process & edge from process A to B rep. A is waiting for B to release a resource.

} if a cycle is detected in either of these graphs, deadlock arises

Deadlock Recovery:

(1) Process termination
. abort one or more processes to break the cycle.
•) done by choosing a process to terminate based on factors like process's priority & resources it holds.

. abort all processes involved in deadlock, though this is more drastic & can lead to data loss.

(2) Resources preemption
Preempt resources from processes involved in the deadlock & allocate them to other processes.
The preempted resources are then restarted.
This can lead to data inconsistency if resources are not properly saved before being taken away.

(3) Rollback - one/more process to a safe state where they were not involved in deadlock then restart process then resource allocation.

(1) Memory
- crucial aspect
- managing th
- ensures - pro
- provides ma
- optimizes

Basic con
(1) Allocating
(2) Tracking
(3) ensurin memory
(4) maximi
(5) freein

Logical
(1) Logical
- genera
- represe
- not
- refere
- OS logic use

(2) Phy
- ac
- ad a

Addre
• Pag vi
• S

1. Safe state → if there exists a seq. of processes that can all be executed to completion, assuming each process receives its max. req. resources.

2. Unsafe state → if no such seq. exists, meaning there is a potential for deadlock.

Algorithm →

Safety Algorithm

① Input - available, allocation, need
② Initialize → work = Available, Finish[i] = False.
③ Find a process → if work = work + Allocation[i], set Finish[i] = True.
④ Check → Finish[i] == True for all i, system is safe state, Otherwise unsafe

Resource request algo.

① Input → req., available, allocation, need.
② Check request validity → if Req[i] <= Need[P][i] ∀i, proceed
if Req[i] <= available[i] ∀i, proceed.
③ Allocate tentatively
④ Check system safety
↓
→ run safety algo. with updated values
→ if system remains in safe state, grant the request
→ otherwise, revert the alloc".

available = available - req.
allocation = alloc.[P] + req.
Need[P] = Need[P] - Req.

Deadlock detection and Recovery

Deadlock detection: OS → by Resource allocation graph.

1) Single instance
↓
if cycle is created.
↓
then deadlock detected.

2) Multiple instance.
↓
if cycle is created
↓
we can't surely say whether deadlock detected or not
↓
So, we check by safety algo

Deadlock prevention -
aims to eliminate one/more of necessary conditions for deadlock

1) Prevent ME :- It's difficult to prevent ME bcz many resources, like printers/files, are inherently non-sharable. It can be mitigated by allowing resources to be shared when possible. (eg. read-only access to resources)

2) Prevent hold & wait :-
- Req. processes to request all resources at once :- This prevents holding some resources while waiting for others
- Alternatively :- processes can release all resources they currently hold if they can't obtain all resources they need.

3) Prevent no preemption :-
- If a P is holding some resources & its requesting addl ones, the OS can prempt the process, forcibly taking resources away from it & reallocating them to other processes. Afterward, the preempted process can resume execution.

4) Prevent circular wait :-
- Impose a Total Ordering of Resources - Assign a unique no. to each resource type. A process can only request resources in an increasing order of their no. This prevents cycles in waiting graph.

Deadlock ~~Detection~~ Avoidance → ensures that system never enters an unsafe state where deadlock is possible.

(1) ~~Banker's Algo~~:

→ system must have knowledge about future req. to avoid entering unsafe states.

Banker's Algorithm → deadlock avoidance algorithm.
→ similar to bank's lending process. Where the system checks whether granting a particular resource req. will leave the system in a safe state.

1. Safe state
be executed
its max.

2. Unsafe st
potential for

Algorithm
Safety Alg
(1) Input-a
(2) Initiali
(3) Find a
(4) Check →

Resource
(1) Input
(2) Check
(3) Allocat
(4) Check

→ run sa
with up
→ if syst
safe st
→ otherwi

Deadlo

Deadloc

(1) Single
if cyc
then

28/12/24

Module - 4

Deadlocks

→ a situation in a multi-process system where a grp. of processes becomes stuck, each waiting for resources held by the others, such that none of them can proceed.
→ This creates a state where no process can make progress, leading to system halting.
→ This situation forms a cycle of dependencies, and no process can continue.

Necessary & sufficient conditions for deadlock :-

(1) Mutual Exclusion → At least one resource must be held in a non-shareable mode, meaning only one process can use the resource at a time.

(2) Hold & wait →
A process holding atleast one resource is waiting for addl. resources that are currently being held by other processes.

(3) No Preemption →
Resources can't be forcibly taken from processes holding them, they can only be released voluntarily by the process holding them.

(4) Circular wait → A set of processes exists such that each process is waiting for a resource held by the next process in the set, forming a cycle.

If all these 4 conditions are met, Deadlock may occur.

28/12/24

1) Reader - writers Problem →

Problem → Involves processes accessing a shared resource (eg. DB or a book).

Reader → can read resources simultaneously (no conflict)

Writers → need exclusive access bcz. writing changes the data.

Challenge is to prevent conflicts, such as a writer modifying data while readers are reading it.

Solution → Give priority to either readers (to avoid waiting) or writers (to ensure updates are done quickly).

→ use synchronization tools to manage access.

2) Dinning Philosophers problem →

Problem →

5 philosophers sit around a circular table with a bowl of spaghetti. Each philos. needs 2 forks to eat.

However, they are only 5 forks for 5 philos.

Challenge is to avoid.

- Deadlock → all philos. pick up one fork & wait indefinitely for the 2nd.
- Starvation → one philosopher never gets a chance to eat.

Solution →

- use algo. to ensure only a limited no. of philos. try to eat at the same time.
- for ex- allow only 4 philos. to try picking up forks, leaving at least one free fork.

(1) Deadlo
→ a situ
becomes
such
→ This
leadin
→ This
can

Necess

(1) Mutu
Share
at a

(2) Hol
A
reso

(3) No
Resou
they
holdi

(4)
proc
proc

If
occ

operations on semaphores:-

- wait() → Decrements the semaphore value.
  → If value > 0, process is blocked until the value becomes non-negative.
- signal() → Increments the semaphore.
  → If value was -ve, it unblocks a waiting process.

Event counters

- synchroniz° mechanism
- use to track the no. of occurrences of an event.
- used to signal when an event has occurred & to coordinate actions b/w processes based on no. of occurrences.

Monitors

→ an abstrac° that encapsulates both data & operation on that data, proving M.E for concurrent processes.

→ Includes → Operations - set of procedures
→ variables - data structure
→ mutex - ensures only 1P can execute an operation at any given time.

→ provides higher-level abstrac° than semaphores for synchroniz°.

Message Passing

- an IPC mechanism
- processes communicate by sending & receiving messages.
- done either synchronously or asynchronously.
- often used in distributed systems where processes are on diff. m/c.
- ex:- a P sends message to another P req. some data, and the receiving process sends back a response.

Classical IPP Problems

- the way diff. processes (or prog.) share data / resources while ensuring proper synchroniz° & avoiding conflicts.

(1) Reader & writer problem
(2) Dining Philosopher problem.

Producer-Consumer Problem

→ shows how 2 processes (threads) can work together while sharing a common resource.

→ 1 P is Producer → generates data/items & puts them into a shared space (eg. queue, buffer, list).

→ other P is Consumer; → takes data/items from that shared space & uses them.

Challenge is to make sure -

1. The [Producer] doesn't overfill the shared space (buffer)
2. [Consumer] doesn't try to take items when buffer is empty.

This req. proper synchronization to avoid conflicts.

example:- Producer generates data → eg. web server producing HTTP req. or a logging system producing log entries.
Consumer processes data → eg. applic? handling HTTP req. or writing log entries to a file.
The shared resources is buffer (eg. memory queue or a file) which has limited space.

Sol?: using synchronization → Semaphores/mutexes.

Semaphores

- synchronization primitive
- used to control access to shared resources.
- signal b/w processes / threads.

ex:- Parking Lot → Imaging a Parking lot with 5 spaces.
When a car enters, it takes up a space. When the lot is full, new cars have to wait until someone leaves.

1. Semaphore value: 5
2. Car Enters (wait):- Reduces value by 1 (4 spaces left)
3. car leaves (signal):- Increases value by 1 (5 spaces left)
4. FULL LOT: if Semaphore value reaches 0, no car can enter until a car leaves

1. Turn Variable →
shared variable called turn. - used to keep track of whose turn it is to access the CS.
For ex:- turn = 0 → Process 1's turn
turn = 1 → Process 2's turn

2. Process Flow →
each process checks the value of turn before entering the CS. If it's their turn, they proceed; if not, they wait until the other P finishes & updates the turn variable.

Adv →1) Simplicity
2) ensures mutual exclusion

disadv →1) unfair waiting
2) Inefficiency (Busy waiting)
↳ P keeps checking turn variable, wasting CPU time.
3) limited to 2 processes.

Peterson's Solution

•) Software-based algo. to solve CS problem, ensuring ME for 2 processes
•) allows 2 processes to share a resource
•) eg. (a file, a printer without interfering with each other, while avoiding deadlocks & race conditions).

It uses 2-shared variable →
(1) flag[] :- an array where each process sets its flag to true, when it wants to enter the CS.
(2) turn :- a variable that indicates whose turn it is to enter the CS.

How it works →
1. Wanting to enter
2. Yielding turn
3. Checking cond?
4. critical section
5. Exiting.

Features →
1. Mutual exclusion
2. Progress
3. Bounded waiting.

Adv
· simple to understand & implem
· solves ME without requiring hw support.

Disadv
· works reliably only for 2 processes
· can't handle multiprocessor systems efficiently.

→ faster than sw-based solns.

Hardware solution

- method of solving a prblm or achieving a task using physical devices (hw) rather than relying on sw alone.

- it is a method of achieving mutual exclusion through atomic instructions provided by the hw, such as Test-and-Set or compare-and-swap. These instructions prevent race conditions by ensuring that certain action occur without interruption.

1. 2. 3. 4. 5. 6. 7. (1) Test-and-set instructions - special inst built into the hw of processor. It checks the value of a variable (like a lock) & sets it to a new value (like "locked") in a single atomic operation. If the variable was already set to locked, it means another process is using the resource, & the current P must wait.

(2) Compare-and-swap (CAS) Inst. -
Compares the value of a variable with a certain value (eg. "unlocked"). If comparison is T, it swaps the value to locked. This operation is atomic (happens in 1 uninterrupted step; preventing multiple P from accessing the resource at the same time)

(3) Disable interrupts -
In some systems, the P can disable interrupts temporarily. While a process is working with critical resource, the OS won't interrupt it to switch to another process.

i) ii) iii) iv) **Strict Alternation →**
- Simple method to achieve ME
- 2 P (or threads) takes turns to access a shared resource.
- It's like setting a rule:- "P1 will go 1st, then P2, then P1 again, so on").
- method enforces strict turn-taking b/w processes
- One P can't proceed until other has completed its turn.
- used to control access to critical sec". (a part of code that accesses a shared resource).

1. Turn variable → shared variable call
For ex:- turn = 0
turn = 1

2. Process How → each process chec if it's their tur other P finishes

Adv. → 1) simplicit
2) ensures m

Peterson's So
- Software-based processes
- allows 2 pr
- eg. (a file, avoiding

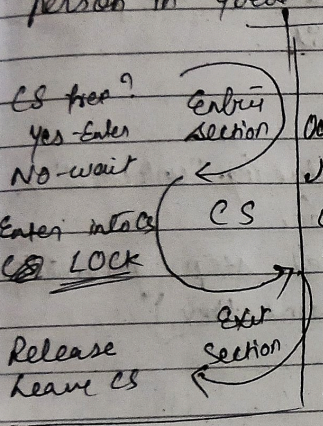
It uses 2-
(1) flag[] :-
(2) turn

How it wo
1. wanting
2. Yieldin
3. Checkin
4. critica
5. Exitin

Mutual Exclusion - any one process can be inside critical sec^{n}. at any time. If any other P req. the CS, they must wait until its free. ex - person waiting in queue.

Progress - if a P is not using CS, then it should not stop any other process from accessing it (any process can enter a CS if it is free). ex - any person can enter if Booth is free, no one should restrict if booth is free.

Bounded waiting - each P must have a limited waiting time. It shouldn't wait endlessly to access the CS.
ex - we can set a Time limit for every person to task, so person in queue no need to wait endlessly.



Race condition

Occurs when multiple processes / threads access shared resources concurrently, & the final outcome depends on non-deterministic order in which they execute.
→ this leads to inconsistent / erroneous results.

ex - 2 threads updating a shared variable at the same time might cause unexpected outcomes bcz. the interleaving of their actions isn't controlled.

Mutual Exclusion → ensure that only 1 Process (thread) can access a shared resources at a time.
→ When multiple processes / threads try to access & modify the same resource at the same time, it can lead to conflicts, errors & unpredictable results.

Ex:- Bank Account → if 2 users try to withdraw money from same bank account at the same time, system could end up giving more money than available. ME ensures that only one transaction happens at a time.

ex:- File access → if 2 programs try to write to same file at same time, one progr. might overwrite the other's data.
ME make sure that only one prog. can access the file at any moment.

23/12/24

Module - 3

(1) Inter-Process Communication

Process executing concurrently in OS may be either independent processes or cooperating processes.

Independent processes can't affect / be affected by other processes executing in the system.

Cooperating processes can affect / be affected by other processes executing in the system. / shares data with other process

for this we need inter-process communi?

Inter process communication (IPC)
→ allows processes to communicate with each other & synchronize their actions.
→ essential for coordinating tasks in multi-process / multi-threaded environment
→ multiple processes / threads need to share data, resources & info.

Critical section → part of code that accesses shared resources (variables, files, DS)
→ only 1 inside CS at a time.
→ shared resources are not corrupted by concurrent process.

In concurrent programming, to avoid erroneous behaviour the shared resource is need to be protected.
The protected section is the critical section / critical region.

It can't be executed by more than 1 process at a time.

Entry section → handles the entry into critical sec?
→ acquires resources needed for execution by process

Exit section → handles exit from CS.
→ releases resources & also informs other processes that CS is free.

Sol? To critical sec? → (to synchronize diff. processes)

Mutual exclusion
Progress
Bounded waiting.

Mutual Exclusion - any time. If any its free. ex -

Progress - if other process (if it is free) should restric

Bounded wai it shouldn't ex - we can person in qu

CS free? yes - Enter NO - wait

Enter into CS LOCK

Release Leave CS

Mutual

a shared

→ When

resourc

unpredic

Ex:- Ba

same

giving

happe

ex:- Fi

time,

- M

an

Feature	RM	EDF
Priority Assign.	Shorter period → higher priority (Fixed)	Earliest deadline → higher Prio. (Dynamic)
Preemption	Preemptive	Preemptive.
Feasibility (Uniprocessor)	optimal	Optimal.
feasibility (multi-pro)	Partitioned / Global (not always optimal)	Partitioned / Global (optimal for feasible systems)
complexity	simpler to implement	more complex due to dynamic priority.
Task assign.	assign to specific processors.	assign to any processor.

ex:- RM
- Task T1 :- Execution time : 2ms, Period = 5ms
  T2 1ms 3ms
  T3 1ms 4ms

Priority Assignment.
T2 (Period 3ms) - highest priority
T3 (4ms) - 2nd "
T1 (5ms). - lowest "

ex:- EDF
- Task T1 : Exec. Time = 2ms, Deadline = 4ms
  T2 1ms 3ms
  T3 1ms 5ms

Priority Assignment.
T2 (deadline 3ms) - highest priority
T1 (" 4ms) - 2nd "
T3 (" 5ms) - lowest "

RR - Round Robin - Each process is assigned a fixed time slice (Quantum) for execution, & if not completed, it's placed at end of queue.

Process no.	AT	BT	CT	TAT	WT	RT
P1	0	~~5~~ ~~3~~ 1 0	12	12	7	0
P2	1	~~4~~ ~~2~~ 0	11	10	6	1
P3	2	~~2~~ 0	6	4	2	2
P4	4	~~1~~ 0	9	5	4	4

Criteria - "Time quantum"
mode - Preemptive.
Given TQ = (2).

Sol:- Ready Queue :- | P1 | P2 | P3 | P1 | P4 | P2 | P1 |
0

Running Queue :- | P1 | P2 | P3 | P1 | P4 | P2 | P1 |
0 2 4 6 8 9 11 12

Multiprocessor Scheduling

In real-time system, tasks must complete within predefined time limits (deadlines)
Multiprocessor systems include multiple CPUs (cores), add complexity to real-time scheduling, requiring careful alloc of tasks to processors to ensure deadlines are met.

2 common Real-time scheduling algo.

RM (Rate Monotonic Scheduling)

EDF (Earliest Deadline First)

Feature	
Priority Assign.	
Preemption	
Feasibility (uniprocessor)	
Feasibility (multi pro.)	
complexity	
Task assign.	

ex:- RM

ex:- EDF

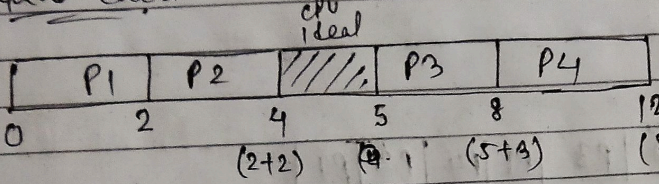
FCFS (First come 1st Serve) → Process are executed in order they arrive.

Or

Process no.	Arrival time	Burst Time (Execution)	Compl. time	TAT	WT	RT
P1	0	2	2	2	0	0
P2	1	2	4	3	1	(1-1)=0
P3	5	3	8	3	0	(5-5)=0
P4	6	4	12	6	2	(8-6)=2

Criteria "Arrival Time"
Mode " Non-preemptive.

Soln:- Gand chart



TAT = [CT - AT]
WT = [TAT - BT]
RT = [Time at which a process got the CPU for 1st time - AT]
Avg. TAT = $\frac{\text{total (TAT)}}{\text{total no. of process}}$
Similarly, Avg. WT

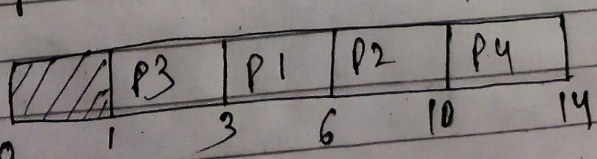
SJF (Shortest Job First) → process with shortest burst time is scheduled first.

Or

Process no.	AT	BT	CT	TAT	WT	RT
P1	1	3	6	5	2	(3-1)=2
P2	2	4	10	8	4	(6-2)=4
P3	1	2	3	2	0	(1-1)=0
P4	4	4	14	10	6	(10-4)=6

Criteria:- Burst time.
mode:- non-preemptive.

Soln:- Gand chart (running queue)



③ Process Scheduling

→ method by which OS decides which process to execute next on CPU.

→ Objectives → maximize CPU utilizⁿ.
→ minimize waiting time & turnaround time.
→ ensure fairness & avoid starvation

Type of Schedulers

Long-term	Short-term	medium-term
decides which process should be admitted in ready queue.	decides which process in ready queue should be assigned the CPU next	handles swapping processes in & out of main memory.

Scheduling criteria

CPU utiliz?: → maximizing CPU usage to improve system efficiency.
Throughput → no. of processes completed per unit time.
Turnaround Time → total time taken to execute a process (from arrival to completion)
Waiting Time → " " a process spends waiting in ready queue.
Response Time → Time b/w process's submission & its 1st response

Scheduling Algorithm

- ① Preemptive (Round Robin)
- ② Non-preemptive (FCFS, SJF)

Scheduling algo. FCFS, SJF, RR

Multiprocessor scheduling

Real-time scheduling

Preemptive scheduling → scheduler can forcibly remove a running process from CPU.

Non-Preemptive " → A process retains control of CPU until it completes / blocks.

loads the

efficiently
How signals
ution,
a running
process.

nning
ocess)

aiting for I/O)

2
dy)

load state

Save state

up)

② Thread :-

- Definition → lightweight process
  → shares same memory space within a process
  → but have its own prog. counter, registers & stack.
  → used for performing multiple tasks simul. within a proc.

- Various states of a Thread -
  - New - Thread is created but not yet started.
  - Ready - Thread is ready to be scheduled for execution
  - Running - Thread is currently executing
  - Blocked/waiting - " waiting for some event / resource
  - Terminated - " has completed its execution.

- Benefits of Thread -
  - Efficiency → it shares same memory space → ↓ memory overhead → making comm". b/w threads faster.
  - Responsiveness → it improve resp. of applic". by allowing multiple task to run concurrently.
  - Better CPU Utilization → it make effi. use of CPU by running multiple threads in parallel.

- Types of Thread -
  - user-level
    - managed by user-level lib.
    - OS is unaware of their existence.
  - Kernel-level
    - managed by kernel
    - better (providing) perfor. & handling blocking issues efficiently.

Multithreading → ← (OS like Windows, Linux, MacOS supports) to enable efficient use of CPU resources

→ allows multiple thread to exist within a single process & execute independently without sharing the same resources (memory, files, data)
→ perform multiple task simultaneously
→ improving efficiency & responsiveness of applications.
→ widely used in - Web servers
- GUI Applic".
- Gaming
- Real-Time systems.

Context Switching

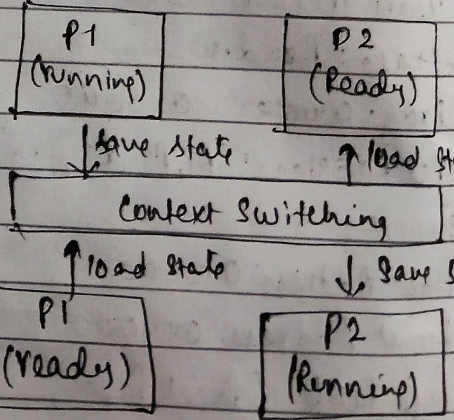
→ OS saves the state of a currently running process & loads the state of another process.
→ allows CPU to switch from one process to another
→ enables multitasking in time-sharing systems.

Necessary for → multitasking → multiple processes share CPU efficiently
(advantage) → Interrupt Handling → respond to interrupts (HW signals) and resume normal execution,
→ Preemptive scheduling → allows OS to interrupt a running process to execute a higher priority process.
→ I/O Operations → Switches CPU to another process when current pro. is waiting for I/O.

disadv:
1) overheads to CPU execution time.
2) freq. switching can degrade syst. perfor. (called Thrashing)

Steps → ① Save the context of current Process.
CPU state (prog. counter) stored in PCB of currently running proces.
② Update the current process state.
State is updated to ready (if interrupted) or blocked (if waiting for I/O)
③ Load the context of Next process.
CPU loads the state to be executed.
④ Update the CPU Scheduler.
Scheduler decides which process to execute next based on scheduling algo. (FCFS, Round Robin).
⑤ Resume Execution
CPU starts executing instr. of new process.

ex:- Assume 2 processes, P1 and P2
1. P1 is running on CPU
2. A timer interrupt occurs, preempting P1.
3. OS saves P1's state in PCB
4. Scheduler selects P2 for execution
5. OS loads P2's state from its PCB
6. CPU starts executing P2.



② Thread :
- Definition →
- Various s
- New - Th
- Ready - Th
- Running -
- Blocked
- Terminate
- Benefits
- Efficiency
- Responsiv
- Better
- Types
- user-
- managed
- OS is

Multith
→ OS like
CPU re
→ allows
indepen
→ perfor
→ impro
→ widely

Process control Blocks (PCB)

→ data str. used by OS to store info. about a process.
→ acts as repository for all attributes & metadata related to process, enabling OS to manage & track processes efficiently.

It includes following info.

1) Process Identific. info.
- Process ID (PID) → unique identifier assigned to each process
- Parent PID (PPID) → ID of process that created current process
- user ID (UID) → Identifies user who owns the process
- Group ID (GID) → group associated with the process.

2) Process state :- New, Ready, Running, Blocked/waiting, Terminated.

3) CPU Registers (accumulator, prog. counter, stack pointer) for context switching

4) CPU scheduling information :- Scheduling queue (ready queue, waiting queue)

5) Memory management info.
- Base & limit registers → define address space allocated to process.
- Page Tables or Segment Tables → for virtual memory management
- Code segment, Data seg., Stack seg. → Logical div. of memory.

6) Accounting info. → CPU time, memory usage, I/O operations performed

Process ID (PID): 101
Parent Process ID (PPID): 100
user ID (UID): 500
State: Ready
Prog. counter: 0x0045
CPU Registers: [R1, R2, R3, ...]
Priority: 5
Base Address: 0x2000
Limit Address: 0x4000
Page Table: [Page1, Page2, ...]
Open Files: [File1, File2, ...]
Resource usage:
- CPU Time: 120ms
- memory: 1MB
- I/O Devices: [Device1, 02, ...]

Role of PCB in Process management:

1. Context Switching
2. Scheduling
3. Resource Allocation
4. Tracking & Debugging

12/24

Module-2

① Processes :-

Definition → It is a program in execution
→ consisting of address space (code, data, stack) and execution context (registers, prog. counter).
→ fundamental unit of execution in an O.S.

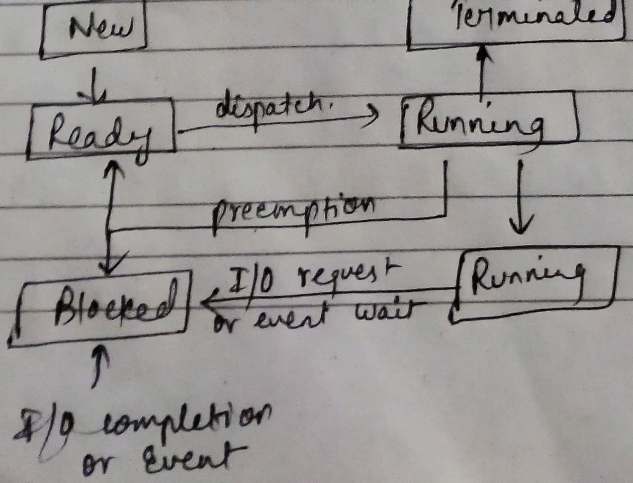
Process Relationship →
- Parent & child process → process can create another process (child) using system calls like fork in UNIX.
  → parent process controls the child process.
- Sibling processes → Two process that shares the same parent
- Process hierarchy → tree str. that links parent & child processes.

Diff. States of process →

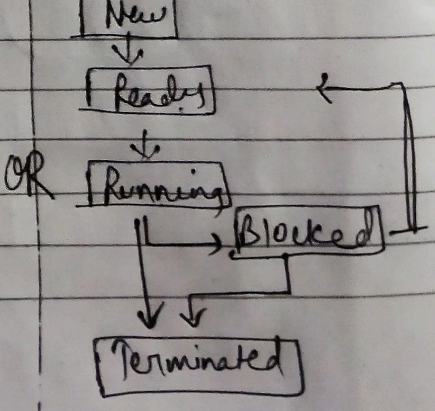
New	→ process is being created
Ready	→ process is ready to run but waiting for CPU time.
Running	→ process is currently executing on CPU
Blocked / Waiting	→ process is waiting for some event like I/O complet.
Terminated.	→ process has finished execution.

Process State transitions →

New → Ready	→ process created & moves to ready state
Ready → Running	→ scheduler selects the process for execution
Running → Blocked	→ process req. I/O or other resources, causes it to pause.
Blocked → Ready	→ occurs when event/resource/process was waiting for available. become
Running → Terminated	→ process finishes execution, terminated by OS.
Blocked → Terminated	→ In some OS, if blocked process is killed by OS or user, it transitions directly to terminated state.



OR



Process control Blocks (PCB)
→ data str. used by OS
→ acts as repository enabling OS to manage
It includes following info.
1) Process Identific. Info.
- Process ID (PID) →
- Parent PID (PPID) →
- user ID (UID).
- Group ID (GID)

2) Process state :-

3) CPU Registers (a

4) CPU scheduling

5) Memory manag
- Base & limit
- Page Tables or
- Code segment

6) Accounting in

Process ID (PID)
Parent Process I
user ID (UID)
State: Ready
Prog. counter
CPU Registers
Priority: 5
Base Addres
Limit Addr
Page Table:
Open files:
Resource

Case Study on UNIX and WINDOWS OS

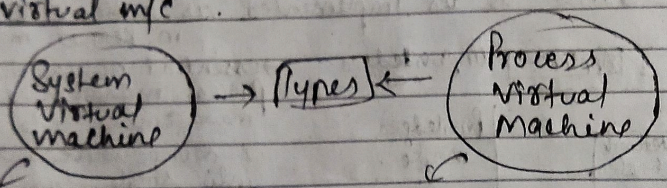
Parameters	UNIX	Windows
Basic	Command-based OS.	menu based OS.
Licensing	Open-source system used under General Public License.	proprietary sw owned by Microsoft.
User Interface	text-base interface (harder to grasp for newcomers)	graphical user interface (simpler to use)
Processing	multiprocessing	multithreading
Security	more secure (changes req. explicit user permission)	less secured as compared to UNIX.
Case sensitive	fully case-sensitive, files can be considered separate files.	case sensitivity as an option.

Concept of virtual machine

→ It is an OS that runs on s/w that replicates special h/w.
→ Abstracts the h/w of our personal comp. like CPU, disk drivers into many diff. execution env. as per our req.
→ When we run diff. processes on an OS, it creates an illusion that each process is running on a diff. processor having its own virtual memory with the help of → CPU scheduling
→ virtual-memory techniques.

Virtualization → method of creating a software-based, "virtual" version of computer with good amounts of CPU, memory, & storage (from host comp. to remote server)

Types of virtual m/c



→ full virtualiz^n. VMs
→ facilitates replacement for an actual m/c.

→ created for executing several programs of comp. within the platform-independent environment.

Adv.
→ easy maintenance, availability
→ energy & cost savings
→ easy backup & clone.
→ flexibility & customiz^n.

disadv.
- One VM can be affected by other running VM depending on workload (when multiple VM simulta. running)
- not efficient as real one when accessing the h/w.

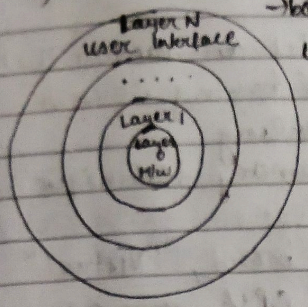
~~Case Study~~ .

Case Study on UNIX

Parameters	
Basic	Comma
Licensing	Open-so under
User Interface	Text (hard
Processing	
Security	
	Case sensitive.

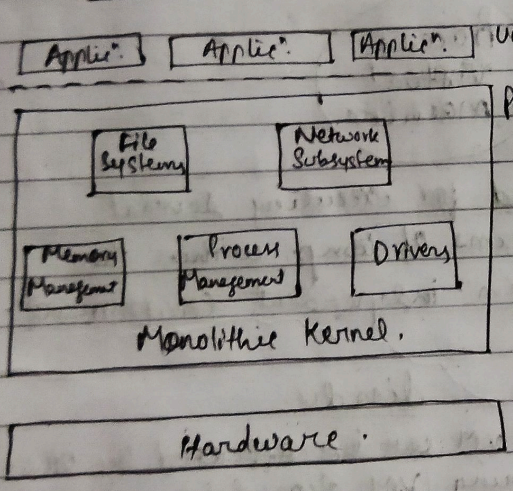
Structure of OS → Layered Str.
→ Monolithic Systems
→ Microkernel.

Layered Str. → OS is broken into no. of layers (levels)
→ bottom layer (layer 0) is H/w. Topmost layer (layer N) is user interface.



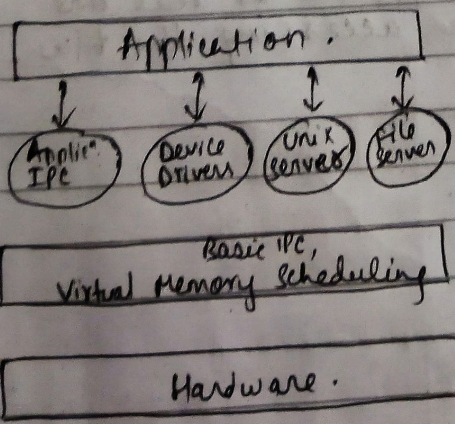
→ These layers are so designed that each layer uses the fun". of lower-level layers.
→ This simplifies debugging process.
→ ex:- UNIX.
→ adv. → easy to perform debugging & system verifie".
→ disadv. → req. careful planning for designing the layers, as higher layers use the functionalities of only lower layer.

Monolithic System → entire OS is implemented as a single large process in kernel mode.



→ essential OS services like:- process management, memory management, file systems & device drivers are combined into single code block.
→ adv. → performance- fast (∵ everything runs in a single block) ∵ communic" is quick.
→ disadv. → hard to maintain as a small error can affect entire system.

Micro-Kernel



→ removing all non-essential components from kernel & implementing them as system & user programs.
↳ This results in smaller kernel called micro kernel.
↳ ex:- Mac OS
↳ adv. → makes OS portable to various platforms.
↳ disadv. → increased level of inter module communic". degrades system performance.

6) Mobile OS → Smartphones, tablets, wearables.
→ power efficiency, touch-based interfaces & connectivity.
→ ex:- Android (open-source)
iOS (Apple, used in iPhone, iPads & iPods
→ Some wearable devices - watchOS by Apple run on smartwatches, offering features like notific°, fitness tracking & voice assistant.)

OS Services

1> Process management → manage processes, scheduling & synchronization.
2> Memory " → allocates & deallocates memory, handles paging, segment°. & virtual memory.
3> File management → manages files, directories, permissions.
4> I/O " → manages I/O devices, device drivers, spooling
5> Security & Protection → through authentic°, encryption, access control.
6> User Interface → provides command-line & Graphical interfaces for user interac°.
7> Error Handling → Detects, reports & recovers from errors in system operations.

System calls

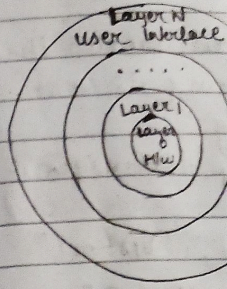
Programming interface b/w user applic°. & OS.
Provide mechanism for requesting services from OS (file manipulation, process control & memory management)

Key services:-
i> File operations → Open( ), read( ), write( ), close( ) - manage file operat°.
ii> Process management → fork( ), exec( ), exit( ) - for process creation, execution & termination.
iii> Memory Allocation → malloc( ), free( ) - manage dynamic memory alloc°.
iv> Device I/O :- System calls for reading from or writing to devices

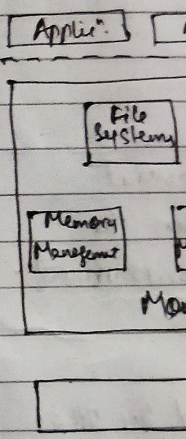
ex- System calls in UNIX, Linux, Windows API calls (eg. CreateFile( ) in Windows).

Structure of OS

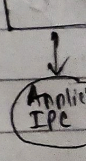
Layered Str. →



Monolithic S



Micro-Ker



5. 5th Generation → 2000s - Present

eg. Windows 10/11, Linux, macOS, Android etc.

→ Multiprocessors, Adv. networking, cloud computing.
→ Mobile OS → smartphones - lightweight & energy efficient
→ Security & cloud integration.
→ AI & ML

Types of OS

① Batch OS → processes jobs in batches without interaction with user during execution.
→ ex- IBM OS/360 (earliest OS)
UNIX (earlier version)
→ Banking System → In banks, it is still used to process end-of-day transac. and generate statements in bulk.

② Time-Sharing OS → allows multiple users to interact with computer simultaneously.
→ CPU time divided into small time slices.
→ ex - UNIX (multitasking OS, widely used in academic, research) Linux, windows, etc.
→ Online game servers - multiple users can interact with same server in real time, each getting a slice of processing time - essential for multiplayer online games.

③ Distributed OS → coordinates multiple comp. in a network to appear as a single system to users.
→ allows resource sharing
→ Google's MapReduce - large-scale data processing & distributed storage across many m/c in cloud computing env.
→ Large-scale websites - like Google, fb, Amazon - manage & process data across multiple users worldwide, ensuring quick response times & high availability.

④ Network OS → Commun. & resource sharing among computers
→ Microsoft window server
→ Corporate Network → companies use to manage comm. b/w comp., manage files on shared drives.

⑤ Real Time OS → manage tasks that req. immediate & predictable responses.

- Soft RTOS: Android, Windows 10
- Hard RTOS: missing a deadline can lead to system failure or dangerous consequences.

→ ex:- Medical equipment - hard RTOS like VxWorks (used in devices like pacemakers)

21-12-24.

Module - ①

Operating System

① Introduction.

Concept of OS:- A software layer that manages H/w resources & provides services for comp. prog. It acts as an intermediary b/w the user & the computer H/w, ensuring efficient use of computer's resources.

Generations of OS - Evolution of OS over time (influenced by technological advancements, hardware dev., and needs of users & industries)

1. 1st Generation → 1940s - 1950s

ex- ENIAC, UNIVAC I.

→ Vacuum tubes
→ no OS.
→ m/c operated manually, each prog. loaded individually.
→ prog. written in m/c lang., users directly interacted with h/w
→ each job processed sequentially, no multitasking

2. 2nd Generation → 1950s - 1960s

ex- IBM 7090, IBM 1401

→ Transistors
→ Batch Processing System - multiple jobs collected into batches & executed one after another without user interaction.
→ Reduced the need of manual intervention.
→ early form of monitoring.
→ prog. langs → Fortan, COBOL used to ease S/w dev.

3. 3rd Generation → 1960s - 1970s

ex - IBM OS/360, Multics, CTSS

Compatible Time Sharing System.

→ Integrated circuits (ICs).
→ Multiprogramming - allowed multiple prog. to be loaded into memory at same time, improving CPU utilizⁿ & throughput.
→ more sophisticated services → Time Sharing → Interactive systems. (users to interact with computer in real-time.)
→ manage resources more efficiently.
→ Virtual memory - enabling prog. to execute even if they req. more memory than physically available.

4. 4th Generation → 1980s - 1990s

ex. MS-DOS, UNIX, Windows 95, macOS

→ Microprocessors
→ Personal comp. (PCs)
→ GUI (Graphical user interface) (eg. windows, Macintosh).
→ Networking & Destributed System. (to connect multiple m/c - share resources)
→ Multitasking - running multiple appⁿ at same time.

5. 5th Generation →

eg. Windows 10/11, Linux, macOS, Android etc.

Types of OS

① Batch OS → processes
→ ex. IBM UN
→ Banking and ge

② Time-Sharing OS

③ Distributed OS

④ Network OS

⑤ Real Time OS

Soft RTO
Android
Windows 10