Q's impt

* Goal of OS → Primary → Convinience / User friendly
  Secondary → Max CPU util, Less Starvation, High priority process execution

- Evolution of OS

① Mainframe computer
* Only CPU + Mem
* Not interactive
* I/P & O/P → Patch cards
* ↓ Mem, slow, No Mem hierarchy

MS DOS

→

② Batch OS
* Jobs with similar needs are grouped into batches

Disadvt → Jobs have direct access to CPU, Inefficient CPU util

ATLAS

→

③ Spooling ↗ Simultanious Perepheral operations Online
* Mem hierarchy – Primary & Secondary
* CPU interacts with only MM
* I/P o/p devices → Very slow

Advt → Better CPU util
→ Jobs dont have access to I/P

Disadvt → No Multiprogramming

~~THE, Dijkstra~~

↓

④ Multiprogramming OS
* Supports Multiprogramming
* Context switch when a process is busy

Disadvt → Scheduling algo is difficult, MM mngt is required

~~CTSS, MIT~~ THE, Dijkstra

⑤ Multitasking OS
* Time sharing
* 1 CPU but context switching is too fast

CTSS, MIT

⑥ Multiprocessing OS
* 2 or more CPU + 1 OS

Eg → Windows

⑦ Distributed OS
* >=1 CPU, >=1 MM, >=1 GPU
* LOOSLY CONNECTED AUTONOMOUS interconnected System

LOCUS

⑧ RTOS

ATCS

- Ram and ROM

Ram
- DRAM – Used in MM, Capacitor
- SRAM – Used in Cache, Transistor
- Synch DRAM
- DDR

ROM
- PROM
- EPROM – UV rays
- EEPROM – Voltage

- Virtualization → H/w level virtualization → 1 OS + 1 CPU → 2 OS + 1 CPU
  → Hypervisor / VMM
  → VMware etc

  2 independent tasks can run simultaniously on 1 OS each

- Containerization → S/w level virtualization
  → Application + Dependencies
  → Eg → DOCKER

- BIOS (Basic I/O System) → Low level S/w that resides in a chip on the mother board. Responsible for waking up computer h/w component.

  POST → MBR → boot loader → Start

  UEFI — Limitations → < 2.TB disk
  → Only on 16 Bit OS

  ↳ No limitation on disk space and can run on 64, 32 bit systems as well

- MBR → (Master Boot Record) → 4 Primary Partitions, <=2TB disk
  GPT → associated with UEFI → 128 Partitions; GUID, Multiple copies of data
  Both are memory Partitioning tables.

- Multiprogramming → Context switch when a process is waiting for resource
  Multitasking → context switch for each quantum time → Time Sharing
  Multiprocessing → >/2 CPU → Multitasking with different CPU's
  Multithreading → Context switch b/w threads.

- Monolithic Architecture → File, Mem, device, process mngt etc everything is managed by kernel [UNIX, LINUX]

  Microkernel Architecture → Only IPC, Mem Mngt CPU sch, is kept in kernel. Other functionalities like File, Application etc is kept in user space
  → It states that OS components and drivers run in their own private address space in kernel

  [MINIX] → UNIX based OS

  Windows → OS component (IPC, OS, drivers, CPU sch) live in common address space and no protection b/w them → Not Microkernel
  But still has some Microkernel char → Kernel mode components dont reach into one another's DS.
  ∴ Windows → Microkernel + Monolithic

- Turn ON computer

  1) BIOS loads up → waking up h/w components → System Startup (POST)
  2) Boot loader → MBR → Present at first sector of drive.
     → Contains boot strap loader & Partition info

     called GRUB or LILO in linux

BY SHIVKARAMAN

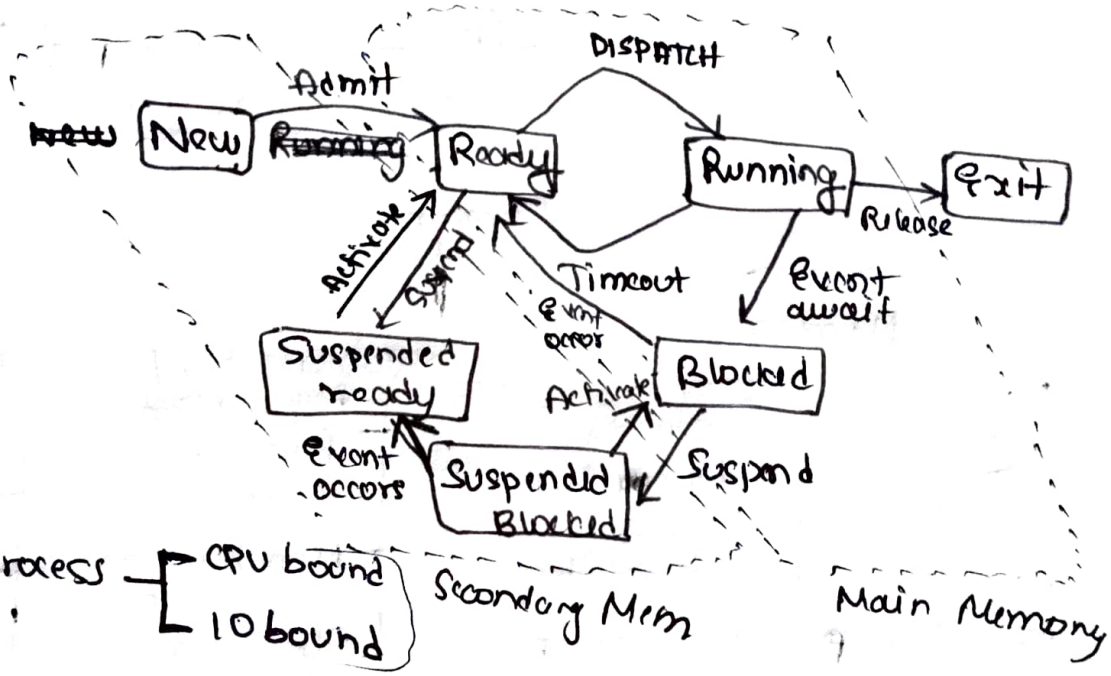
3) **Init** → looks for files / etc / init tab if there is entry for init default. → Determines initial run level of system. (There are 7 levels 0-6)

Design of init → Sys-V (System 5)

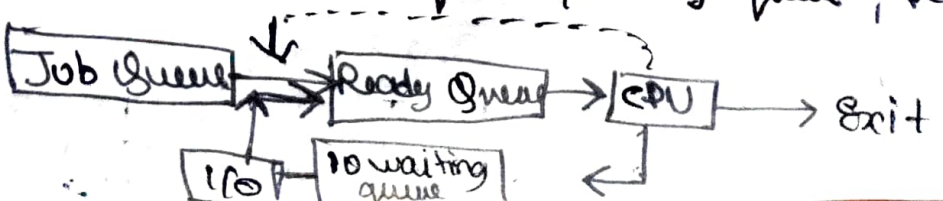
→ Starts various DEAMONS → Background processes like networking → X server Demon is most ~~used~~ impt. manages keyboard, mouse, display

→ When X server ~~stream~~ demon is started, you see GUI and login screen

- **States**



- Types of Process — CPU bound / IO bound
- Schedulers — Long term / Job Scheduler - Sec mem - MM - Controls deg of Multiprog
  - Short term / CPU schedulers - MM - CPU
  - Medium term - MM → SM
- PCB - Used in Multiprogramming as to identify a process
  - Pointer, PID, Process state, PC, Register, Mem limit, Open files etc
  - Process table → PID ↔ Pointer to PCB
- Process — Premtive, Non-Preemptive
- Degree of Multiprog - No of process in MM / Ready queue
- Scheduling Queue → Job queue, Ready queue, Device queue



- Context Switching triggers → Interrupt, Multitasking, User & Kernel mode switching
  → CPU idle while context switching

- IPC — Shared Memory — Preferred when large amount of info has to be shared quickly
  Problem → Producer consumer Problem
  - Bounded Buffer
  - Unbounded Buffer

- Message Passing — used in distributed systems
  - Msg — Msg header + Body
  - Comm links has to be established before sending msg.

Communication

- Naming
  - Direct
    - Symmetric addressing: send(P, msg), recv(Q, msg)
    - Asymmetric addressing: send(P, msg), recv(id, msg)
  - Indirect
    - Mailbox/Port acts as intermediatory: sent(MB, msg), recv(MB, msg)
- Synchronization/ Asynchronized
  - Blocking Send (Synch)
  - Non-blocking Send (asynch)
  - Blocking recieve (synch)
  - Non-blocking recv (asynch)
- Automatic/ Explicit buffering
  - Zero capacity
  - Bounded Buffer capacity
  - Unbounded buffer capa

- Zombie Process → Process completed but still entry in process table
  Orphan Process → Parent process dead/completed but child still running
- Thread → Basic unit of CPU utilization, segment of a process
  → context switching b/w threads does not reset cache

Max no of zombie process depends on size of process table

- User Thread
- Kernel thread

Multithreading Models

- Many to one
  - Green thread of Solaris
- One to one
  - LINUX, WINDOWS 95-XP
- Many to many (Best)
  - UNIX (HP-UX)

BY SHIVKARAMAN

- Benefits of Multithreading → Responsiveness, Resource Sharing
  - Responsiveness → Allows a program to run even if a part of it is blocked
  - Resource Sharing → Threads share common code, data and files
  - Economy → Allocating memory and resources for a process creation is costly job. It is more economical to create and context switch threads
  - Scalability → Threads run parallelly in Multitasking and multiprogramming increasing scalability.

- Examples of Multithreaded apps → Web browser, Web server, Computer games, Text editors, IDE
  - Computer games → Different obj (car, bike, etc assigned a diff thread)
  - Text editors → Spelling, syntax checking, formatting etc
  - Web server → New thread is assigned to each new request

- Optimal no of threads → 1 thread per core if threads dont do I/O synchronization etc → Ideal condition and not possible in real life
  - → ↑ Thread ⇒ ↑ performance only upto a limit
  - → Best Method → Trial and Error

- Dispatcher → Module that gives control of CPU to process selected by short-term scheduler
  - → Time taken for dispatcher to stop 1 process and start another process → DISPATCH LATENCY

- Scheduling Criteria → ↑ CPU utilization, ↑ Throughput, ↓ Waiting Time, ↓ Response time, ↓ Turn around Time

- Terminologies → Arrival Time (AT), Completion Time (CT), Turn Around Time TAT = CT − AT, Burst Time (BT), Waiting Time WT = TAT − BT

- Algos – FCFS
  - Non preemptive
  - highest avg WT
  - Convoy Effect – Several small process may need to wait if large process comes first

BY SHIVKARAMAN

- **SJF**
  - ↓ BT
  - BT of two process is equal ⇒ First Come process is executed
  - Starvation → high priority process keeps coming and low priority process get blocked for indifinite time

  Solution → Ageing → Increase priority as waiting increases

- **Priority Scheduling** – ↑ Priority ⇒ Selection
- **Priority Preemptive** – Executed in 1 unit time → Check Priority
- **Round Robin** → Quantum time.

  At time quantum too large → RR behaves like FCFS

  too small → RR behaves as Process Sharing

  – time sharing technique

  – No starvation but lot of context switching

- ~~Preemptive SJF / Shortest Remaining Time first~~

  – 1 unit → Check RT

- **Multilevel Queue Sch (MLQ)**

Partitioned Ready Queue

High Priority
- System Processes → Queue 1
- Interactive Process → Queue 2
- Batch Process → Queue 3 → Background process

Low Priority

Each queue has different sch algo. A process is permanently assigned to queue

– Starvation

- **Multilevel Feedback Queue Sch (MLFQ)**

  MLQ + Process moves from low priority queue to high priority queue as it ages

  – Most complex algo

- **Process Synchronization** → Co-ordinating execution of processes in such a way that no two processes can have access to ~~each other resources~~ shared data & resources at the same time

BY SHIVKARAMAN

- **Race Condition:** More than one process manipulating a shared data/resource due to which outcome differs from actual expected o/p.

- **Critical Section** → when two or more process try to access the same segment/resource → this segment is called CS.

- **CS sol should satisfy** → Mutual Exclusion, Progress, Bounded wait (optional)

  Mutual Exclusion → only 1 process can access CS at a time

  Progress → only process that aren't running in Remainder section can decide which process should run in CS

  Bounded wait → Maximum of time a process can wait after req for CS

- **CS solution**

  2 process Sol
  - S/w sol → S/w sol 1 (turn) [NEV, Prog X], S/w sol 2 (flag) [NEV, Prog X], Peterson, Humble, Dekkers (turn & flag) → NEV, BW, P
  - H/w sol → Test & set (lock), Swap (key & lock) → NEV, P, BW - Not ensured

  Called H/w sol coz they perform atomic operation on shared mem (lock) and CPU directly supports without any complex CPU synch methods.

- **Semaphore** → Integer variable shared b/w threads
  - Binary (Mutex lock)
  - Counting

  wait(S) → while(S<=0); S--; lock

  signal(S) → S++ unlock

  Initial value S=1 (unlocked)

  S=0 → locked

  Disadvt → BW X

  S → tells how many processes can simultaniously access the CS problem

  Application → CS problem, Order of exec of process, Resource Mngt

  Solution to BW → Maintain a queue.
  - → In wait → if CS is locked, block the process and add to queue
  - → In signal → Remove blocked process from queue

- Producer consumer Problem → empty (n), full (0), mutex (1) prod, cons → process, buffer → CS

- Reader Writer Problem → If process is writing, no other process is allowed to read/write. If any reader is

BY SHIVKARAMAN

reading, no other process can write.

- wrt, mutex (1) (1), Reader, writer → Process
  reader_count → CS

- Dining philosophers problem
  - No of C-S = No of chopsticks = 5
    ⇒ S = 5 ie Semophore is nothing but chopsticks coz it cannot be shared b/w any philosopher

Possible Solutions

- Assumption – All philosophers picks right chopstick first
  - Sol – One philosopher picks one chopstick at a time. If he has both chopsticks he eats. He drops both chopsticks after eating
  - Semophore Chopstick[5] All initialised to 1

- Allow philosopher to pick chopstick only if both chopsticks are available

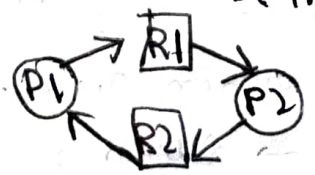
- Odd numbered philosophers picks Left and then right
- Even numbered R & then L

- Busy wait → Repeatedly checks condition in loop
- Spin lock → Specific type of lock used for mutual exclusion

Diff → Spin lock specifically refers to synch mechanism and busy wait is in general

- Deadlock → If process is unable to change its waiting state indefinitely because the resource is held by another waiting process then the process is said to be in deadlock

- Necessary conditions
  - Mutual exclusion
  - Hold & wait
  - No premption
  - Circular wait



Deadlock handling

- Prevention
  - Violation of any one property
  - ME violation not possible
- Avoidance
  - Bankers algo
  - Resource allocation graph
  - wait for graph
- Detection & Recovery
  - Detection
    - Active
    - Passive
  - Recovery
    - Process termin
    - Resource
- Ignorance

BY SHIVKARAMAN

1) Prevention

- ME → Not possible
- Hold & Wait → P cannot request for R if it is holding any resource
- Non-preemptive → Making it preemptive
- Circular wait → P holding R_i can request for R_j if $j > i$

2) Avoidance

* Req additional info abt how, what, when Res will be requested by Process

Res allocation graph

P → R → Request edge

P - - - → R → Claim edge

P ← R → assignment edge

wait-for graph

contains only P

3) Detection & avoidance

Detection

- Active → Actively look for deadlock
- Passive → when CPU util drops significantly

Recovery

- Process Termination
  - Abort all deadlocked process
- Resource Preemption
  - Prempt all deadlocked resources

4) Ignorance → Ostrich algorithm

- Logical Address → Virtual address → Generated by CPU while prog is running and doesnt physically exist.
- Logical Address space → Has all address generated by CPU
- Physical address → Actual address in MM
- User can never access Phy add. He can access only logical address.
- Logical Add → Phy add → Done by Mem Management Unit (MMU)

Mem allocation

- Contigious
  - Fixed partitioning
    - Fixed size
    - Variable size
  - Dynamic Partitioning → suffers External fragmentation
    - Solution
      - Compaction
      - Non contiguous mem allocation
    - Best fit
    - First fit
    - Worst fit
    - Overlays
- Non Contigious
  - Paging
  - Segmentation

Internal, External Fragmentation and deg of multiprogramming is limited by no of partitions

BY SHIVKARAMAN

- **Paging**

  * SM divided into fixed size → PAGES
  * MM divided into fixed size → FRAMES

  Size of page = frame

  Order of page doesnot matter ⇒ Ext frag solved

  but if page size is large ⇒ Internal Frag → Reduce Page size

- **Address Translation & binding**

  → limit reg, relocation reg

  → Logical add —(MMU)→ Phy add → Address Translation

  * Programs, data etc mapped from 2° M → 1° M → Address Binding

  Symbolic address ← address → Absolute add

  COMPILE TIME BINDING

  when compiler knows where in MM prog will be loaded

  symbolic add ↓ Relocatable add ↓ abs add

  Load Time Binding

  During runtime, if process is removed from MM and loaded back with diff address

  Runtime Binding

- **Static linking**

  prog → Compiler → Objcode → Linker → Loader → MM

  Even unnecessary modules / function even if not used

- **Dynamic linking**

  * uses Shared library → Contains freq used functions
  * use stub in mm which refers to func inside library

- **Add translation in paging**

  eg → Goto 10 in pag2 → Symbolic add

  To find abs add

  1) Find logical base add of page 2
  2) Find offset of 10 in page 2
  3) logical add = base add + offset

Page table : page no ↔ Frame no.

log add → page no | offset Phy add → frame no | offset

BY SHIVKARAMAN

- PT ~~implementation~~ Access & Storage
  1) Using Register → Increases context switching time
  2) PT in MM → and Page Table Base Reg (PTBR) contains base address of PT in MM
     → Prob → 2 Mem access req for each add translation
  3) Use TLB (Translation look aside Buffer) → hardware cache which contains PNO ↔ FNO
     → TLB hit → 1 Mem access
     TLB miss → 2 Mem access
- PT implimentation
  - Hash Table
  - Inverted PT
- Segmentation → Divide progr into diff segments according to user view
  → Uses Segment Table | segno | offset |
- Paging increases context switch time because for every context switch, the page table must be updated every time a process moves in and out of memory
- Virtual memory → Provides an illusion to programmer that a program greater than size of MM can be executed. Here 2° Mem is used as if it is part of MM.
  → Here in PT, we have an additional col which contains VALID/INVALID bit → Tells if page is loaded in MM or Not
  → If page not present → Page ~~Replacement~~ Fault
  → Page fault and MM full ⇒ page in MM should be taken out and desired page should be bought → PAGE REPLACEMENT ALGO
- Demand Paging → Some pages of process in MM
  → Rest → Bought on demand | Paging + Swapping |
- Pure demand paging → No pages in MM
  → Every page is bought on demand
- Swap space → Section of SM used for ~~RAM~~ virtual mem.

BY SHIVKARAMAN

- Page Replacement algos

1) FIFO (Belady's anamoly) - oldest page selected
States that increasing no of frames increases no of page faults for given ref string

BEST ← 2) Optimal Page Rep Algo - Replace page which will not be used in longest dimension of time in FUTURE

3) Least Recently Used Algo → Replace LRU in PAST

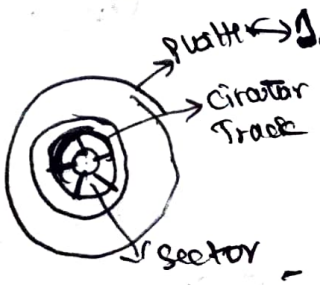
$$\boxed{\text{OPRA} > \text{LRU} > \text{FIFO}}$$

- Trashing → CPU is spending major portion of time handling page faults. It is directly proportional to deg of multiprogramming

→ Explain using Demand paging

Solution → Increase MM size, Allocate exact no of frames that are actually required, Use efficient page replacement algo, Reduce Multiprogramming

- Disk architecture



platter → Info is stored magnetically
Multiple platter (CD like plates) → Each platter has two magnetic surface. Platter → Logically divided into CIRCULAR TRACKS → Divided into SECTORS
- Basic unit of Storage → SECTOR

- Seek Time → Time req to move arm to req track
- Rotational Delay / Latency → Time req for disk to rotate so that req sector comes under read/write head (= 1/2 rotation time) Approx value works most of time
- Positioning timing / Random access time
  - Seek time + Rotational delay
- Transfer rate → No of heads × Capacity of 1 track × RPM

BY SHIVKARAMAN

No of heads

2 × No of platters × No of tracks in each platter × No of sectors × Data size in each sector

Tot no of tracks

Tot no of sectors

Size of disk.

- **Disk Scheduling algorithm**

  Goal → Minimize Seek Time

  1) FCFS
  2) SSTF (Shortest Seek Time first) → Goto nearest request and move in same direction and change direction when no more req in that direction
  3) SCAN → Given direction, go along direction until largest req is processed and also move till last value (even if not in req queue) and then change direction
  4) LOOK → SCAN algo but dont go to end value
  5) C-SCAN → direction → End → opp direct begin → direction
  6) C-LOOK → C-SCAN - without extra