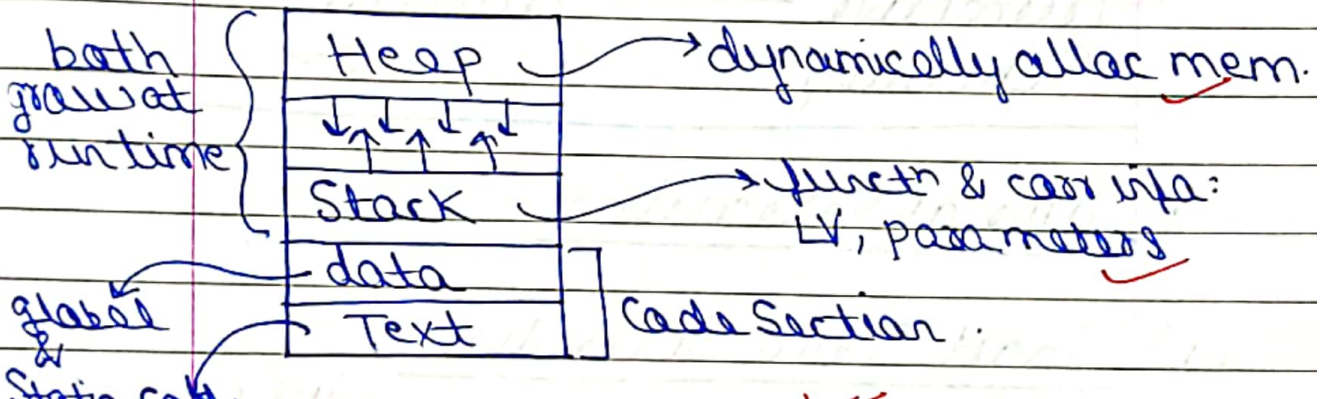
OPERATING SYSTEMS

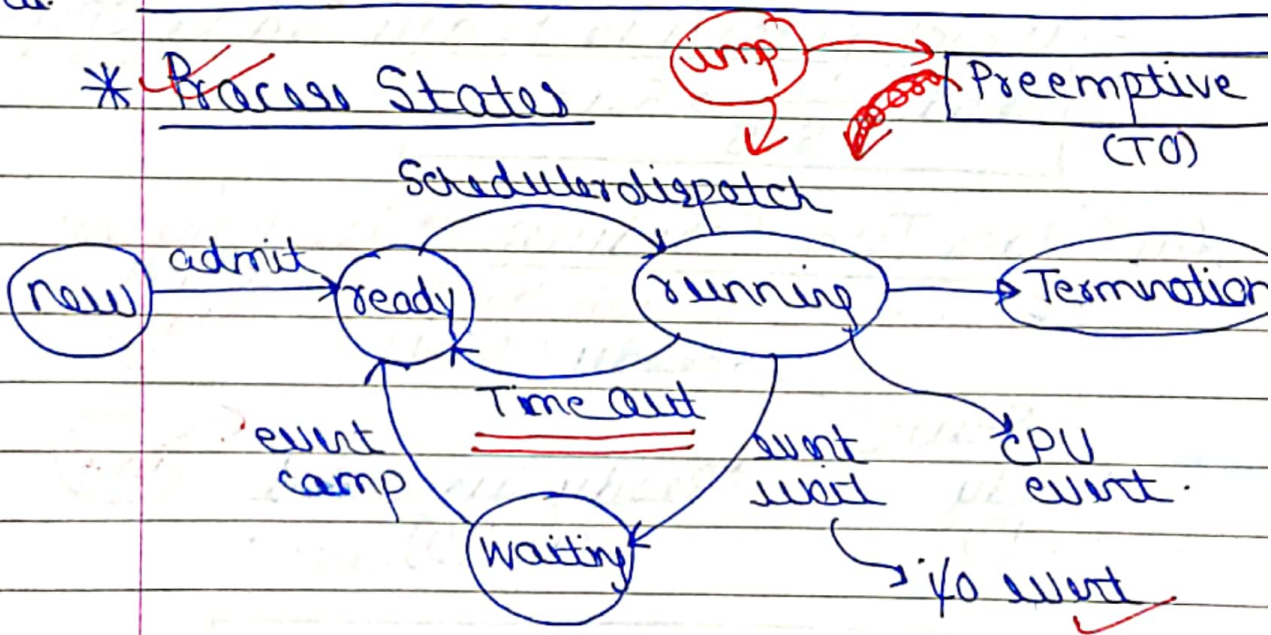
[Avg: 7M - Imp]

1. PROCESS

- OS acts as interface b/w h/w & s/w.
- Process: Program under execution. (loaded in MM from SM). [Hard Disk]



* Process States



- Ready: loaded in MM (in Partition).
- Running: CPU executes from MM.
- When P hits i/o event ⇒ WS.

* Timeout: max amt of time, it can spend with CPU.

⇒ Ready & Waiting, both are implemented as Queue, with LinkedLists (PCBs)

* Non-Preemptive Scheduling → P1 terminates
→ P1 waits for I/O wait.

* Preemptive Scheduling → Timeout

* Types of Schedulers

(i) Short Term Schedulers (imp)

- Which process to be assigned to CPU.
- Ready —STS→ Running

(iii) long Term Scheduler → doesn't use any CPU Scheduling Algo.
[controls deg of multi prog]

New → Ready (MM)
(already in HD) → Ready Suspended (in HD) (imp)

(iii) Medium term Scheduler

- involves in Swapping from SM to MM & vice-versa, when Process hits I/O wait. (imp)
- which Process must be sent as brought from SM, at the time of Swapping.

eg → Pg 11

* Process Control Block (PCB)

Process Id
Process States
CPU registers
PC
CPU scheduling info → WT, AT, TAT, BT
Memory mang info → PT
Accounting info → Amt of real time.
Device info & OpenFiles → List of resources.

* Process Switching

P1 (stores PCB1)

P2 (loads PCB2)

- Saving current contexts (PCB) & loading the contexts of new Process: Context S.

Dispatcher
mass control of CPU to that Process
[This interrupts]

Scheduler (Gate)
takes decision that which Process to be exec next
[This doesn't interrupt]

* Dispatcher latency: time taken to switch from 1 Process to another

* Operations on Process

1) Process Creation (imp)

- forK(): returns -1 : failure
  returns 0 : child Process.
  returns Pid child : Parent.

eg → Pg 19 (order of exec^n) (VA)

eg → Pg 21 [i/n forK() → no. of P = 2^n]

eg → Pg 22 (10 times)

ii) forK(), executed n times:
No. of Child Process = $2^n - 1$

2) Process Termination

- Normal / exit() / simult. termination of Parent & child / sequential

* Scheduling Criteria (imp)

1. CPU utilisn : % of time CPU used.
2. throughput : no. of Process exec per unit time

3. turnAround time : entire time Gap b/w Process Arrival & completion. (imp)
4. Waiting time : amt of time in Ready Q
5. Response time : time taken to produce first response from CPU

② CPU Scheduling Algorithms

1) FCFS (First Come First Serve)

- Non-preemptive algo.
- suffers from Convoy effect : longer jobs executed b4 Shorter Jobs, leads to delay in Shorter Jobs execution.

eg → Pg 38 (WT = 4.5, TAT = 9)

eg → Pg 40 [% CPU utilisn]

eg → Pg 43 [x = 28.6]

2) SJF [Shortest Job First]

- non-preemptive
- No Convoy effect Here

3) SRTF [Shortest rem time first]

- preemptive.

- late coming Shorter jobs will take control from longer time, current job. (imp)

eg → Pg 47 [SJF & SRTF]

- Process can suffer from Starvation. (drawback)
  [longer Jobs Starve] (imp)

* Predicting next CPU Burst

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$ (imp) [CBT]

- exponential avg.
  τ_{n+1} : pred. value of next CPU B.
  t_n : actual CPU Burst at n^{th} itn
  τ_n : predicted CPU B at n^{th} itn.

4] Priority Scheduling

if Priority on BT : SJF/SRTF
if Priority on AT : FCFS

eg → Pg 53 (Premp & Non-Premp. Priority)

- low Prior. Process can suffer from Starvn
  Solution : Aging (improve priority of Process, as waiting time increases) (imp)

eg → Pg 60 {3, 2, 1}

eg → Pg 62 & 63 [SRTF & Priority]

5] Round Robin Scheduling Algorithm

- CPU time equally divided among Proces
- used in time Sharing.
- Preemptive FCFS : RR

TQ.

come back after : $t \leq (n-1)q$ (imp)

- RRA produces excellent response time.
- TQ & Avg TAT varies irregularly.

eg → Pg 68 (ResponTym b/w FCFS & RR)

- when TQ ↑ it behaves like FCFS.
- if CS ↑, RT ↓ [better resp] (imp) (back)

eg → Pg 17 (Rev copy) [RR]

- Throughput is poor in RR.
- RT best in RR.
- SRTF is optimal (throughput, Avg WT) (Gate)

6] LRTF [Longest remain time first] (Preemptive)

- longest jobs given max priority.

eg → Pg 18 [Rev copy]

7] Multilevel Queue (Preemptive)

H
- System Process
- Interactive Process
- Inter. editing Process
- Batch Process
- User Process

(imp) foreground (served using RR); background (served using FCFS)

L
[Priority]

Multilevel feedback Queue

H
Q1 TQ=8 ms (imp)
Q2 TQ=16 ms
Q3 FCFS

Each Queue: diff. Scheduling Algo, diff. priority.

L
- Process inside Q3 will be served only when Q1, Q2 empty.
- excellent response time. [like RR]

8] HRRN (Highest response Ratio next)

- Non-Preemptive
- give priority to Shorter jobs, but also takes care of Longer Jobs in Queue.

$R = \frac{W+S}{S}$ (Gate)

eg → Pg 78 | P1 | P4 | P3 | P2

eg → Pg 20 (LRTF - tick)

eg → Pg 84 (RR) GATE

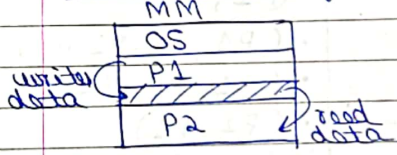
eg → Pg 87 (CPUB & i/o B) (imp)

8] INTER PROCESS COMMUNICATION

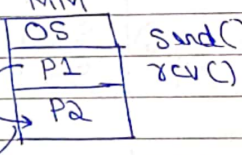
- Process → independent
- Process → cooperative

⇒ 2 techniques (imp)

(i) Shared Memory



(ii) Message Passing



Direct: send(msg, dest)

Indirect [MailBox]: send(msg, mailBo

* Blocking Send/rcv : sender/receiver gets blocked (can't perform any comp) until send/rcv opn completed.

Overall exec time ⇒ Communicn time + computation time

* Non-Blocking Send/rcv :

Overall exec time ⇒ max (communicn time, computation time)

④ * Concurrency

(i) Serial exec : coffee m/c used one by one
(ii) Parallel exec : 10 coffee m/c, parallely, many served.
(iii) Concurrent exec : 1 coffee m/c, 2 users, 1M, 1F will be served concurrently.

P1	P2
S1	S2
S3	S4

imp

- (P1, P2)
- (P1, P2, P1, P2)
- (P1, P2, P1)
- (P2, P1)
- (P2, P1, P2, P1)
- (P2, P1, P2)

eg → Pg 95 [Concurrent exec]

* Due to improper synchronisation : it leads to incorrect result : race condith. → inconsistency.
→ lack of mutual exclusion.

eg → Pg 98 [register Values]

⑤ Process Synchronisation

- CS : Place where a commonly Shared V. are Kept. (Gate)

* Solutions to CS Problem

do { Entry Section : → Lock = 1;
// CS
Exit Section
RS → non code not dependent on CS
Change Lock = 0.
} while (true);

* Soln to CS Problem must Satisfy : (imp)
(i) Mutual exclusion
(ii) Progress
(iii) Bounded wait

- ME : only 1 Process can be in CS, at a given time.
- Progress : CS shouldn't be empty.

Progress ✓, Strict Alternation X

- Bounded wait : If P2 req. to enter CS, P1 can't enter again. [Not ∞ waiting]

eg → Pg 103 [Strict Alternation] (imp)

Complete Solution

① Peterson Soln to CS Problem (S/W)

Code → Pg 29 (Rev Copy) imp

- ME. (P1 in CS, P2 cant enter)
- Progress: P1 can enter again & again, & if P1 leaves, P2 can also enter
- Bounded wait: If P2 in SpinLock, P1 leaves CS, can't enter again, P2 enters.

② H/W Solution to CS Problem (H/W)

(i) ME using test&Set()

init Value of lock = false;

Code → Pg 31 (Rev Copy) imp

- for 1 Process, testandSet() returns F, for rest (outside CS), returns True.

(ii) ME using Swap()

Whoever gets inside CS, makes Lock → true & Key → false,
while leaving, make lock → false
[available]

Code → Pg 32 (Rev Copy) imp

(iii) Complete H/W Solution imp

Code → Pg 33 & 34 [Read Carefully]

eg → Q4 (Pg 118)
eg → Q5 (Pg 120) } GATE Q.

* SEMAPHORE

- doesn't always guarantee mutual exclusion/deadlock
- used for synchronization purpose.
- variable upon which we perform wait & signal.

imp

```
wait(Semaphore S)
{
  while(S<=0);
  S--;
}
```

```
signal(Semaph S)
{
  S++;
}
```

- Semaphores → Counting S (resource alloc)
  → Binary S (0, 1)

eg → Pg 126 (min no of Semaphores)

* Drawback of BS: [traditional] when 1 process in CS, other can in SL, simply wasting CPU cycles. imp

* Modern implementation of BS (imp)

```
wait(Semaphore S)
{
  S--;
  if(S<0)
  {
    Place Process in Q;
    Block();
  }
}
```

```
Signal(Semap. S)
{
  S++;
  if(S<=0)
  {
    wakeup();
  }
}
```

* Deadlock

- When none of the process can enter CS. (imp)

If any 1 sequence shows DL : DL
If no sequence shows DL : DL free

eg → Pg 128 [Deadlock Seq]

eg → Pg 131 [DL] (P3 → P2) (imp)

* POPULAR PROBLEMS (Semaphore)

① Producer-Consumer Problem

Code → Pg 40 [Rev Copy] (Gate)

[mutex, full, empty]

② Readers-Writers Problem

- Only 1 writer is allowed to perform write oper^n at a time. (No reader/wrtr)
- Any no. of readers are allowed [At that time, writer not allowed]

Code → Pg 41 [Rev Copy] (imp)

③ Dining Philosopher Problem

- 5 Philosophers, 5 Chopsticks

Soln.: [Deadlock Problem]

1. allow only 4 P to enter.
2. Perform taking of CS, when 2 is avail.
3. even takes right CS first & then left, odd takes left CS first & then right CS. (imp)2

eg → Pg 139 (Q1)

eg → Pg 143

eg → Pg 145

eg → Pg 149 (imp)

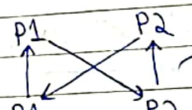
Gate Qs

⑥ DEADLOCKS

resources → consumable
→ reusable

sharable (files) — non-sharable (printer)

DL



→ R2 held by P2, R1 requested by P1.
[Circular Wait] : DL.

Test

* Conditions for occurrence of Deadlock:

(i) ME : (Non-Sharable resource)
(ii) Hold & wait :
(iii) No-Preemption (cant Snatch)
(iv) Circular wait : (Pn holding Rn, req R1)

Necessary

Sufficient

* Resource Allocation Graph (RAG)

G = (V, E) → req/assign/claim.
↳ Process & resources

multi instance

If cycle : Possibility of DL
If no cycle : No DL

→ in RAG

imp

* Deadlock Handling Methods

(i) DL Prevention
(ii) DL Avoidance
(iii) DL (Recovery) Detection
(iv) DL Recovery.

(PAD-R)

(1) DL Prevention
- avoid any one condition.

(i) ME : use Sharable resource then.

(ii) Hold & wait :
- allot all resources to P1 at start.
- P1 can request another, only when it is holding none.

imp

(iii) No Preemption :
- if P1 req R2, R2 not allocated ; P1 needs to release all its resources.

(iv) Circular wait :
- never allow, allow no resource to be requested. [enumerate in inc order]

P1 → R2
P2 → R3
Pn → Rn+1 not, <n (not).

(2) DL Avoidance → Knowledge of resources & req. a priority

Test

(1) Safe State Method
(2) RAG
(3) Banker's algo.

(ii) Safe State Method → Single resource type

MAX - Allocation = need (imp)

Avail = Total - Alloc

eg → Pg 48 (Rev Copy)

eg → Pg 161 [x=5]

(iii) Banker's Algorithm → Multi resource type

- multiple resource types

MAX[i][j] : Pi req max of K no. of resources of type Rj.

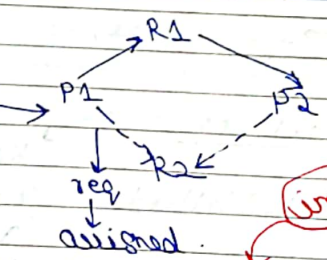
MAX[i][j]
ALLOC[i][j]
need[i][j] = MAX - ALLOC → Matrix.

Available[j]
Total[j] → array (imp)

work = Avail ; work = work + Alloc.

eg → Pg 51 (Rev Copy)

(iii) RAG



imp

- b4 allocating R2 to P1, checks possibility of DL in future.

5) DL Detection

(i) Single instance : Construct WFG (imp)

(ii) Multi instance : Use Banker's Algo.

4) DL Recovery

(i) Process termination → all Process in DL

→ Process by Po, based on Priority

Test → [Wait-die, wait-wound]

(ii) Resource Preemption

- Select the Victim Process & snatch all its resources. (Roll Back)

DL Prevention > (more restrictive) DL Avoidance (Safe)

eg → Pg 171 (Q 2)

eg → Pg 54 (Rev Copy) [imp] (imp)

eg → Pg 175

→ No Preemption possible

Gate Q

Scanned with CamScanner

7) THREADS

- light weight Process
- Thread contains:
  - thread id
  - PC
  - Register Set
  - Stack

(imp)

→ Thread shares code, data, files but has their individ. register/ Stack. (Gate)

- Open an Process is heavier & slower than thread.

Threads → Kernel level threads
Threads → User level threads

diff → Pg 178

Main adv of thread: Multi Process Archit.

(imp)

ULT → implemented by user & cant be split to diff CPUS.
KLT → Yes

8) MEMORY MANAGEMENT

1) Partitioning technique → Static
                          → dynamic.

(i) Static

- Partitions already created in MM (init).
- Process brought here for execution.

(imp)

P1 100KB → 128 | 128 | 128 | 128

internal fragmentation

(iii) Dynamic

- init: partitions not present, created due to placement of process.
- Holes created here (External fragment)
- Compaction technique used: big hole made

eg → Pg 58 (Rev Copy)

(imp)

2) Placement Algorithm

P1 = 100KB

MM	
64KB	PAR1
512KB	PAR2
250KB	PAR3
128KB	PAR4

- Best Fit: PAR4
- Worst Fit: PAR2
- First Fit: PAR2

3) PAGING

(imp)

- MM divided into frames & process into pages.

Page Size = Frame Size

- pages must be loaded into MM for execution.

F#	MM
0	A·0
1	A·1
2	A·2
3	B·0
4	B·1
5	B·2
6	B·3

Page Table of A

Pg#	F#
0	0
1	1
2	2

Page Table of B

Pg#	F#
0	3
1	4
2	5
3	6

- Page Table: page corresp frame info.
  - Contents: F# & V/i bits
  - Pg# acts as index

imp

* Address translation → after MM with

- logical Address to Physical Add.
- MMU does it.
  - CPU generated add.

eg → Pg 190 (Add. Trans)

↳ offset = 13, Pg Size = Fr. Size = 2^{13}
No. of entries in PT ⇒ $2^{Pg\#\ bits}$

imp

Size of PT ⇒ size of each entry * total entries.

⇒ V/i bit: If V, Page is avail in MM.

eg → Pg 62 [Rev Copy - 32 bit LA]

* TRANSLATION LOOKASIDE BUFFER.

- H/w lookup cache [PT defined for every Process]
  - PT → all Pages of that Process
  - TLB → Page corr. frame info of recently accessed Pages from var. Process

imp

disc → Pg 193 [TLB].

hierarchical.

- Effective Mem Access time:
  (LA → PA) + (Mem Access using PA)

imp

$t + (1-h)m + m$

- t: TLB Access time
- TLB HR
- m: page Table[in MM] access.

$EMA = t + m(2-h)$

$h \propto 1/EMA$

* Shared Pages.

eg → Pg 196 (ed1, ed2)

(4) PAGE TABLE STRUCTURES

1. Multilevel PT / Hierarchical PT.

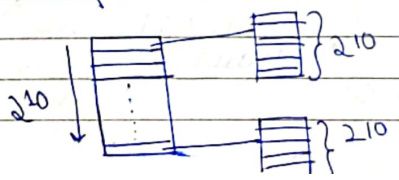
- In Simple PT, allocating large amt of space contiguously in MM: difficult.
  ∴ Multilevel PT.

imp

P1	P2	offset (d)
10	10	12

Pg Size = 2^{12} B.

- No of entries in outer PT = 2^{10}.
- No of entries in inner PT = 2^{10}.



∴ Total = 2^{20}.

- Contiguous memory location not req.
∴ memory efficient.

2. Hash Structured PT

- Page no. hashed into index using HF.
- Chaining technique is used to avoid collision.

disp → Pg 199

3. Inverted PT

Frm#

No. of entries in IPT	=	No. of Frames in which Mem is divided

disp → Pg 200

imp

Contents of IPT: Pid & Pg#
Index of IPT: F#

→ Pgs of diff Process can be placed inside IPT unlike PT.

eg → Pg 201 [Q2]
eg → Pg 203 [Q4]
} Gate Q [Inverted PT]

Size of MM: 512MB.
Address: $29 * 2^{20}$ bits = 29 bits.
→ Byte Addressable
2^{29} bytes.

imp

eg → Pg 207

5. Segmentation

imp

- Memory divided into variable size parts: Segments which can be allocated to Process.

Process: Seg 0, Seg 1, Seg 2

MM:

Seg 0	1000 – 1500
Seg 1	1800 – 2100
Seg 2	2300 – 2700

Diff

eg → Pg 209 [Segmentation work?]

6. Virtual Memory Management

[Max size]

Size of MM < Size of VM < Size of SM.

- VM: enhanced form of memory.
- Allocating VM: increases degree of multiprogramming.

imp

7. Demand Paging

imp

- Degree of MultiProgramming ↑.
- (i) 100 F in MM, 10 Process, each Proc. (= 100 pgs)
→ load 10 pgs of each Process.
(instead of 100 pgs of 1 process).

* Strict demand Paging (imp)

- Never load a page into MM, unless it's required/needed.

* Handling Page faults (imp)

When pge corresponding frame info, has "invalid bit" in Page Table, that means the pge is not in MM = Page Fault

eg ⇒ Pg 212 (Handling PF)

(If no Free Frame = Page Replacement Algo)
[Swap page out of disk]

⑧ * Page Replacement Algorithm

(i) Optimal Algorithm. [work on future requests]

(ii) LRU (least recently used).

eg → Pg 70 (Ram copy)

(iii) FIFO (First in first out).

- Belady's Anomaly: happens in FIFO, (Gate)
[no. of PF ∝ no. of PF-Size]
[Page fault] [Page Frame].

Generally, inverse occurs

(iv) LIFO.

Eg → Pg 71 (FIFO & LIFO)

(Not so imp/come in options)

(v) MRU } Tricky.
(vi) MFU }

⑨ THRASHING

- Due to demand Paging, degree of MP usually leads to thrashing.
- CPU spends more time in Swapping. (∵ prob. of not finding the page incr.).

Thrashing leads to excessive Page I/O. (Gate)

- depending on funct°, diff. pgs are referred.

* Locality of reference

Spatial
[Recent num each referenced again & again]
- array
[Look Ahead]

Temporal. (imp)
[same mem. each, referred again & again].
- int i in for loop.
[Look Aside]

(10) Frame Allocation

Gate

- done by instruction set architecture.

(i) equal allocation

ii) 100F, 5 Process:
Each Process → 20 frames.

(ii) Proportional allocation [100F]
P1 = 160 pgs P2 = 40 pgs
⇒ 80F ⇒ 20F

(iii) Priority allocation — imp

Process with higher P, more frames.

(11) Placement

does by working set strategy.

- Local Placement: Process cant access frames allocated to other Process.
- Global Placement: If any frame of Process is free, it can be used by Process with excess Pgs.

eg → Pg 4 [Rev Copy 2]

eg → Pg 225 [Demand Paging] — imp

eg → Pg 227 (Q7)

(12) FILE SYSTEM

(i) Allocation methods. — internal frag. can be avoided in any

- allocate the blocks in disk to file Blocks.

(i) contiguous Allocation

file	start	len
F1	5	3

Adv: read performance is excellent
disadv: external fragmentation. — imp

(ii) linked list allocation

eg → Pg 2 (Notes – FS) — imp

- Every file contains start Physical Block.

Adv: - external fragmentation avoided.
Disad: - Random access is very low
- Pointer takes up few bytes

(iii) File Allocation Table [for every file → LFAT]

- for every disk blocks in HD, make entry.

Adv: random access is easier. — imp
disadv: FAT size can be too huge, to be stored in MM.

eg → Pg 3 (FAT)

(iii) Indexed allocation (imp)

→ for every file, maintain 1 special block called index Block / I-node

File	Inode
File A	10

[go inside 10, 10 having all Block no. 8 : 3, 4, 5, 8].

Adv: need not bring FAT in MM (memory efficient). [just one node] (Index Node)

- [if very less no of blocks] → indexed block size ↓.
- [if too many blocks, entries in index block ↑] → many index Blocks → multi level index block.

eg → Pg 5 [Notes]

(iv) Extended indexed allocation (Gate)

diag → call

→ if too many blocks. (imp)

eg → Pg 6 (2004 & 2012) [Vimp]

2) DISK SCHEDULING ALGORITHMS

→ OS main Purpose is that HD serves every i/o request in an optimum way.

Seek time: moving r/w Head to the appr. cylinder / track. (imp)

When Process gets suspended / blocked i/o, it is swapped onto HD, queue of process get piled up one by one, they get served [i/o event]

(i) FCFS Scheduling

– No Starvation.

eg → Pg 8

(ii) SSTF Scheduling (imp)

→ least movement of the disk arm

→ reduces total Seek time.

– Starvation is possible → like SRTF.

eg → Pg 9 [Best Throughput]

(iii) SCAN [Elevator]

outside → inside (0)
inside (0) → destination

eg → Pg 10 (dirn will be mentioned)

(iv) C-SCAN

outside → inside (0).
inside (0) → outside (199)
outside (199) → destn.

diag → call

(v) LOOK
- Like Scan, but dont move till innermost cylinder.

eg → Pg 11

(vi) C-LOOK
- like C-SCAN, go till lowest & then rushes)
[Not innermost/outermost].

eg → Pg 12 [Gate 2016 Q] (imp)

3) FREE SPACE MANAGEMENT.

- how to show which blocks in disk are free. [Super block keeps count] (imp)
- bit notation.
- linked notation (list of free blocks).
- counting technique.
- indexed: [i block having list of fB].

4) DIRECTORY STRUCTURE

eg → Pg 9 [Rev Copy-2]

* POINTS from Prev Years.

- Timer interrupt will invoke the scheduler & scheduler just selects the next process to be scheduled.
- It doesn't cause interrupt.
- Dispatcher causes interrupt.

* Transparent: means functioning without being aware. [like VIT]

* Synchronous i/o: Process in blocked S, ISR after i/o, place process back to ready
* Asynchronous i/o: process not blocked process continues executing in that time. ISR after i/o notifies the process, that i/o is done. (imp)

T → Time Quantum

* Scheduler manipulates the process priorities every T time units: [Round robin].

* Semaphore operations implemented within OS Kernel. [Gate-1990]

* Disk has Swap Space, since all the swapping of Processes (carried by MTS) b/w MM & Disk occurs. (imp)
[suspended i/o].

* Hoare's Monitor uses wait/signal operations.

* Locality is commonly used to determ the no. of assigned Pages. The no. of pages that meet the requirement of locality is called working Set. (imp)

* Amt of VM available is limited by the availability of secondary Storage. LRU can cause thrashing obviously.

* Buddy System: Buddy memory alloc technique divides memory into partitions to try to satisfy a memory request. [Runtime environment → CD] (imp)
Garbage collector deprences dangling pointer.

→ LookAhead Buffer → Spatial locality
Look Aside Buffer → Temporal locality

* Locality of reference
↳ page reference being made by the process is likely to be one of the pages used in the last few page references. (imp)

Object Module → Object code
→ relocation bits.
→ Names & locations of all external Symbols. (imp)

→ In a resident OS computer, loader must reside in MM (in all situations)

→ A Page replacement Algorithm suffers from Belady's anomaly, when it is not Stack algorithm.
FIFO → Not Stack
LRU → Stack.
Random → Not Stack. ∴ Belady's

→ Dirty bit: write-back; [if page is modified, & not written back to mem, DB set to 1. (imp)
→ R/W bit: true if page is read only/ r/w. used as Page Protection.
→ Reference bit: used in page replacement algo; to avoid replacement of heavily used Page.
→ Valid: for page initialization. (in mem/ not)

Working Set Strategy

→ OS Selects a process to suspend, if the sum of the sizes of the working Sets exceeds the total no. of available frames. & also: aim of working Set Strategy: initiate other Process, if extra frames available. (Vimp)
[global allocation]

Page fault Service

* Instruction takes i μsec.
Page fault takes additional j μsec.
Page fault occurs every K instn.

4.24

∴ Effective inst time = $i + \frac{1(j)}{K}$

∴ EMA = % of Page miss * PF Service time + % of Page hit * Memory Access time

imp

$\Rightarrow 1 * \frac{1}{K}*(i+j) + \left(1-\frac{1}{K}\right)* i$

Page Fault Service time

$\Rightarrow \frac{i}{K} + \frac{j}{K} + i - \frac{i}{K}$

∴ $\frac{i+j}{K}$.

TLB & Cache

Q 4.35 TLB HR = 96%, Cache HR = 90%.
m = 10 ns, C_t = 1 ns, TLB_t = 1 ns

Vimp

∴ Process → [LA → PA] + [Access PA]

imp

$(TLB_t) + (1-H_t)(2 \times m)$
$\Rightarrow 1 + 0.04 \times 20$
$\Rightarrow$ 1.8 ns

$(C_t) + (1-H_c) \times m$
$\Rightarrow 1 + 0.1 \times 10$
$\Rightarrow$ 2 ns

∴ 3.8 ns.

Page No.: 37.
Date:

→ Q 4.47.

Dirty Page + Page Fault Service

imp

$3 = (1-p)1 + p[p \cdot 300 + (1-p)100]$.

Q 4.39

TLB & Page Fault

EAIT = CPU + 2 * EMAT.

EMAT = (VA → PA) + Access byte from PA.

[Extra Notes] [& next pg of extra N].

* MM access time = 30 ns.
Page fault rate = 15%.
Page fault Service time = 10 μs.
EMA?

Page fault Service

imp

Soln:

EMA = $p(10) + (1-p)(30)$
$\Rightarrow 0.15 \times 10 + 0.85 \times 30$

* If head changes its direction after servicing every request; Pattern comes out to be: observe the pattern [from Q.]

$$\text{new min dist} = (\text{old dist} \times 2) + 1$$

* MMU (Special h/w Support), needed only for VM. If VM is removed, this h/w is also removed. TESTS-1 (imp)

* Scheduling algorithms (like FCFS, SJF..) are used by STS, LTS uses none.

* ULT: T_1, T_2, T_3 → KLT: T_1 [many to one, uses 1 KLT, otherwise can't use multi Processor facility] (imp)

→ If ULT is blocked, whole Process gets blocked; Kernel can't do CS to run another ULT (unblocked).

→ MultiProgramming: increases CPU utilization

Time Sharing: increases CPU responsive

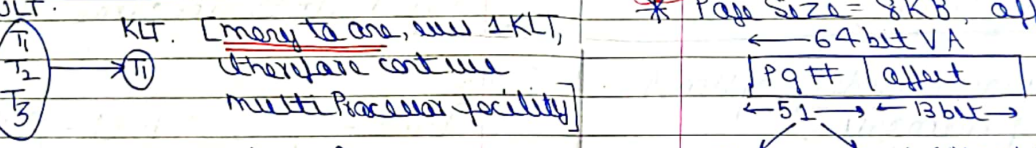
* If all Jobs have identical Run lengths, RR Scheduler → horrible TAT.
[All jobs complete at same time]

* In bit Map Approach: (imp)
[No. Every disk block mapped to one bit].
if no. of disk blocks = 40M.
40M bit = $40M \cdot \frac{1}{8} B = 5MB$. (Size of Bit Map)

* LTS controls deg of multiProgramming.
[No. of Process present in MM at any time]
MTS can only decrease deg of MP, by suspending some Process.

* Semaphore can't always avoid DL. [Thread can hold many locks at a time]
eg → Q7. [GATE]

* Page Size = 8KB, offset = 13 bit. (imp)



$p = n/k$ = no. of tags / k-way

eg → Min Size of TLB Tag [GATE]

* Indexed allocation can support both sequential + direct access. [Linked → Seq.]

* If PA = 30 bit, PA space = 2^{30}.
$\frac{\text{PA Space}}{\text{Page Size}}$ = No. of frames.

eg → Inode (Present in Memory)
Q15 [No. of disk access required to access a byte Position]

* If, Virtual address Space = 4 GB
Virtual Address = 32 bit
If, MM has 32K physical Pages,
∴ No of Frames = 2^{15} ⇒
frm # = 15 bit
∴ content of PT = 16 bit = 2 B (1 WB)

* Demand Paging taken care by OS, nothing to do with Programmer.
Shared Pages can be Swapped out to disk
→ OS automatically loads page from disk, when necessary (imp)

* Purpose of dynamic loading: optimal utilization of memory

* Routine not loaded until called: dynamic
All routines are part of code, & are loaded at the same time, regardless of called or not: Static

* CPU utilization increases as the deg. of multiProgramming increase up to a threshold after that utilization starts decreasing.
[due to thrashing]

Page Table Size of 1st PT = VA of 2nd PT

NOTE: If there is deadlock, progress not Satisfied
if progress satisfied, no deadlock

[Attempt]

eg → Q27 [Levels of PT required, Page Table Sizes are calculated at each level]

eg → Q31 [Page faults, like Prev Year Gate Q]
[not just circular Wait] (imp)

* For System to be in deadlock, all 4 conditions must be satisfied.
* if each resource type has single instance then cycle in RAG implies deadlock.

eg → Q33 [Tough Q on Scheduling & Lock → Concept Clearing (Imp)]

TEST-2 (2020)

* 30 instances of resource 'R'. (mistake)
Each Process → 4 instances
Max possible no. of Processes for System to be in safe State?
9 ✓ (3×8) = 24; 1 will get 6.
10 ✗ (3×10) = 30; None will get 4

* external fragmentation: When there is enough total memory space to satisfy req., but avail. space is not contiguous.
[dynamic Partitioning]

(imp)

* System is well utilized when: CPU utilization ↑, disk utiliz? ↓

* A Thread running in User Mode in critical Section may get context switched (imp)
∴ Process can get prempted, while exec. Critical Section code.

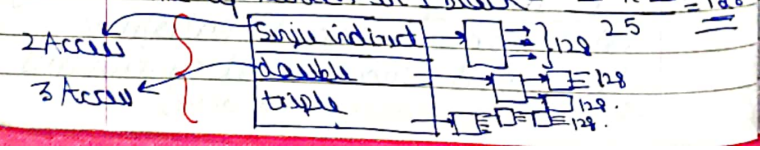
* Process Id is a part of PCB.
Process creation is done in Kernel Mode. (imp)
There is a bit in PSW (Prog Status Word) that indicates mode: User / Kernel.

* Approximations in Page Table Size

- No. of PT entries = 2^{23}.
- Frm# = 16 bit.

Given (2 valid bit, 2 MB, 22 bit).
∴ 16 + 2 + 2 + 2 = 22 bit.
∴ 22 bit ≈ 3B. ← (imp) (Approx)
∴ PT size = $2^{23} * 3$.
= $2^{20} * 2^3 * 3$ = 24MB

* Disk block Size = 512B (imp)
Pointer = 32 bits.
∴ No. of pointers in 1 block = $\frac{2^9 \times 2^3}{2^5} = 128$



∴ 1 double indirect pointer = 128 × 128 blocks
1 triple indirect = (128 × 128 × 128) blocks
1 single indirect = 128 blocks

* Dynamic linking → on demand linking
- dynamic linking optimizes (decrease) memory space utilization. (imp)
- Increase prog. execution time.
- (Dynamic linking, dynamic loading, dynamic add. binding), practically used in majority of OS.

* SRTF can be implemented in a time Sharing OS. (Its round Robin). (imp)

* TLB needs to be saved sometime on a context Switch between process.
PC must also be saved during CS.

* System calls invoked using S/W interrupt

(imp) Context Switch → PC, Stack counter, GP register, TLB } needs to be saved

eg → Q28 [Similar Q in Prev Year]

* Page Size = 1K Words [1024 words]
4 Frames, 8 pages

Given: List of Virtual Addresses.

Pg	Address
0	0 - 1023
1	1024 - 2047
2	2048 - 3071

(imp) Approach

* CO Q → Magnetic Disk.

12380 bytes per track
Rotation time = 20 msec
Seek time = 45 msec
Time to read a 1024 B block ?

⇒ Time = ST + Rot.L + TransferTime.
⇒ $45 + \frac{20}{2} + TT = 56.65$

1 rotation covers 1 track → imp

12380 —— 20
1024 —— x
$\therefore x = \frac{1024 \times 20}{12380} = 1.65$

* Page Size = 1KB = 1024 B
Info to be stored = 4000 B
∴ 4 pages are required
Max. 4 TLB miss will occur.