Operating System

Avg marks: 8 marks

I Introduction & Background

II Process management
- process concept
- CPU scheduling
- synchronization
- Concurrent programming
- Deadlocks
- Threads

40%

III Memory Management
→ RAM chip Implementation
→ Loading, Linking & Address Binding
→ Techniques
  - paging
  - multi-level paging
  - inverted paging
  - segmentation
  - segmented paging
→ Virtual memory

40%

IV File system.

mail:- Balakrishna.veerala@gamail.com

1,2,3 ← part A
4,5 ← part B
6,7,9 ← part C

↑ OB chapter

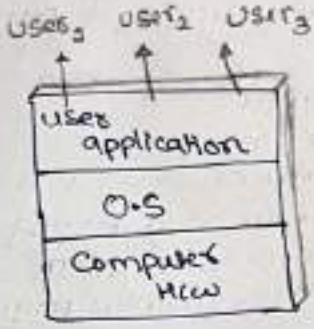
1 milli sec = 10^3 micro sec → 10^{-3} sec
→ 10^6 nano sec

1 micro sec → 10^{-3} milli sec → 10^{-6} sec
→ 10^3 nano sec

1 nano sec → 10^{-6} milli sec → 10^{-3} micro sec → 10^{-9} sec

- Introduction & Background

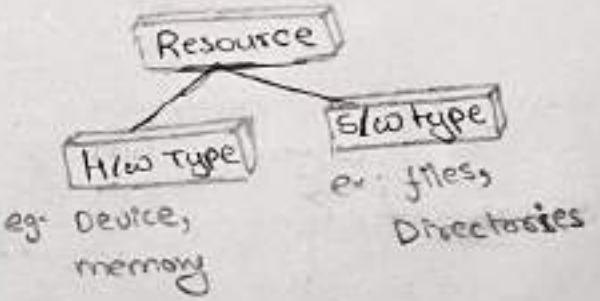
what is an OS:- It is an Interface b/w user & computer hardware




```
void main()
{
  int x;
  printf("Hello");
}
```

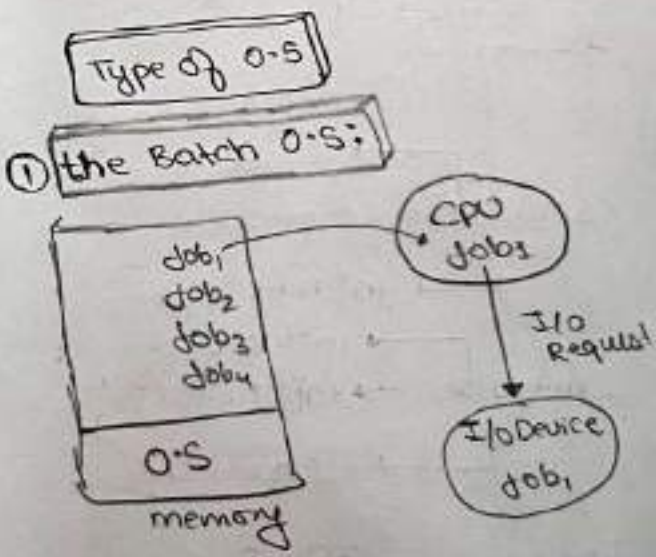
Internally call write() system call invokes to communicate with the monitor

- System call
- System call is request made by the user program to the O·S in order to get any kind of services
- O·S is also called as resource allocator b/c it is responsible of allocating resource of the Computer

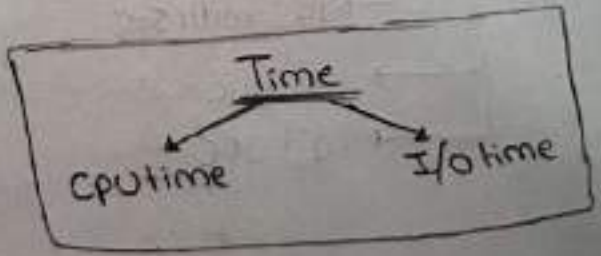


- Goal of O·S

1. the primary goal is convenience (easy to use)
2. the secondary goal is efficiency (Stability)



1. If the job is completed completely then only another job will be schedule
2. Increased CPU idleness
3. Decreased throughput of the system.

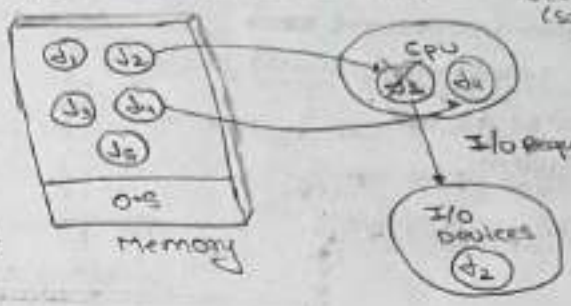


ex: IBM, OS/2

throughput

no. of job completed per unit time is know as throughput of the system.

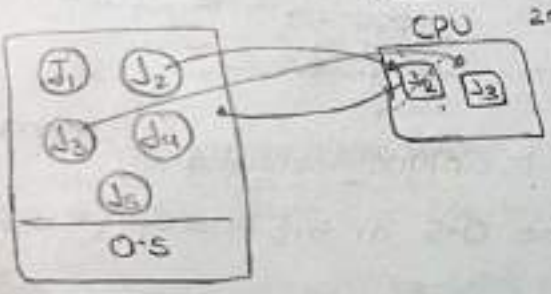
2. **Multi-programming O.S** → need more security (so it need Dual mode operation)



1. If the job is leaving the CPU to perform I/O operation then another job which is ready to execution will be Schedule on to the CPU.
2. Increased CPU utilization
3. Increased throughput

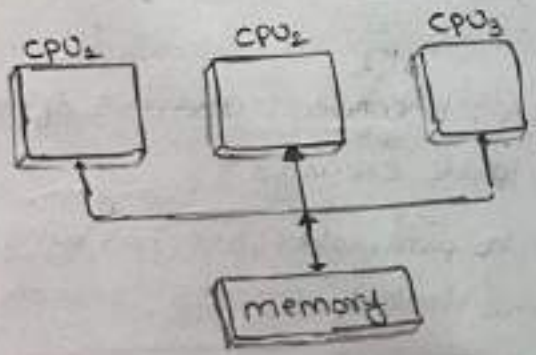
ex: windows, UNIX

3. **Multi-tasking O.S**



1. multi-tasking is an extention of multi-programming O.S
2. Job while be executed by using time sharing mode.

4. **Multi-processor System** → (parallel system)



1. Increased throughput of the system.
2. Reliability (fault tolerant system)
3. Economical

ex: UNIX

5. Real time System

ex: S_x works, V_x works, RTOS

- the system which are strict deadly time bound are called as real time system
  - these should not be any delay
  - **Hard Real time** → no miner delay is allowed
    ex: satellite systems, missile system
  - **Soft Real time** → minor delay is allowed
    ex: Banking sector

Process Management

- O·S has to keep track of everything about the process (context)

- **Process concept.**

program under execution is called as process

- It should reside in the main memory
- It is accepted the cpu to execute the instruction
- Active & dynamic (taking I/p & producing o/p)
  - active: b/c it is executing some instruction on CPU

- **Attributes.**

1) process id
2) process state
3) program counter
4) priority
5) general purpose Register
6) List of open file
7) List of open device
8) protection information

[1] process id: it is unique identification no. assigned by the O·S at the time of process creation

[2] process state: it contain current state information of a process where it is residing

[3] program counter: it contains address of the next instruction to be executed.

[4] priority: it is a parameter assigned by the O·S at the time of process creation

---

- All the Attribute of the process is called as Context of the process
- the context of the process is called as PCB [Process Control Block]

P·id	P·S
P·C	Priority
L·O·F	L·O·D
G.P.R	P·I

P·C·B

- Every process will have it's own PCB (Consist of 4 or 5 PCB keep track of context)
- PCB will be stored in main memory & will be implemented by double linked list
- Process will have various state

States
1. New
2. Ready
3. Run
4. Wait or Block
5. Termination or Completion
6. Suspend Ready
7. Suspend wait

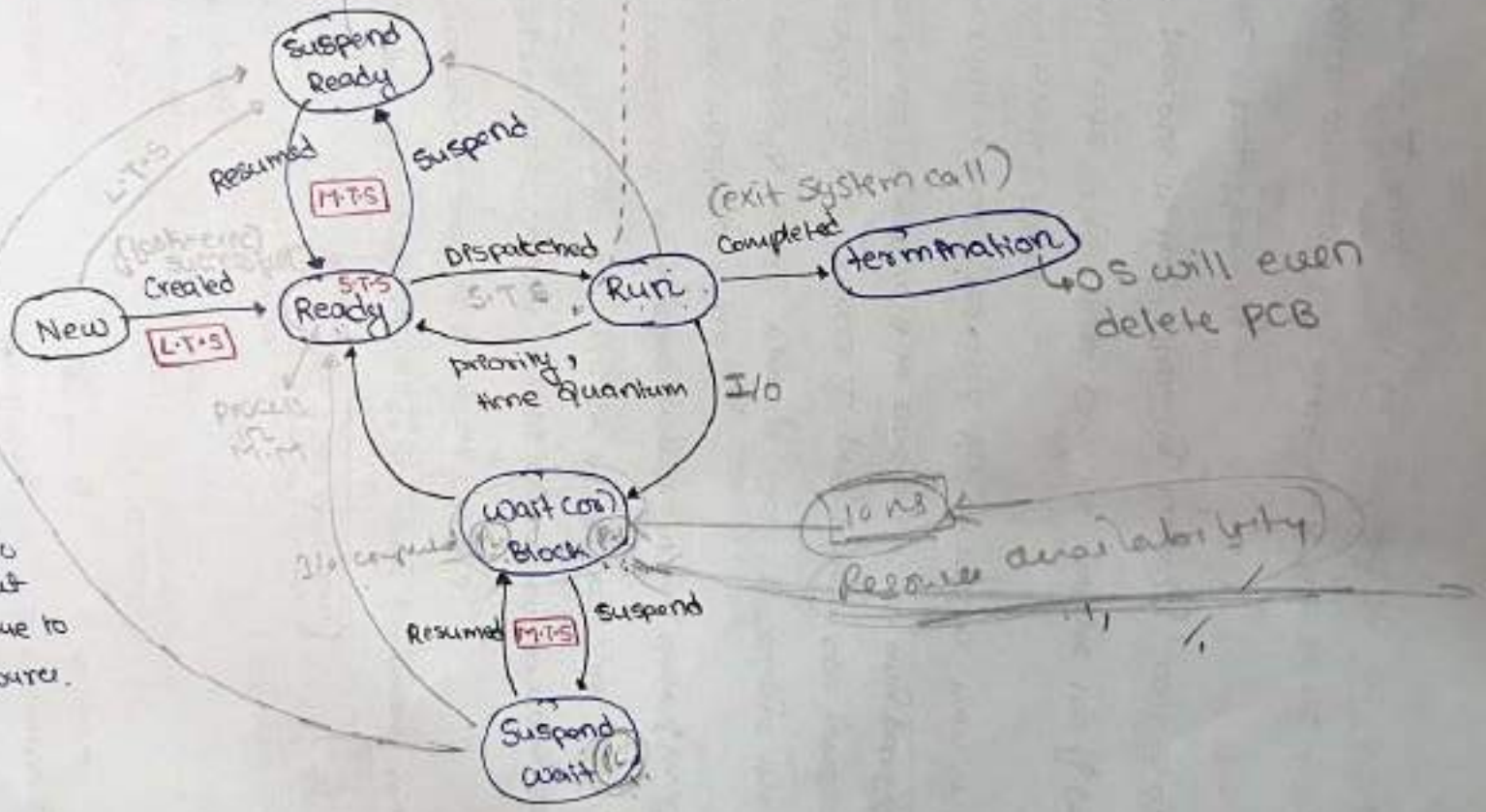
- Various operation performed on the process
  1. Create
  2. Scheduling
  3. Dispatching
  4. Execute
  5. Termerating or killing
  6. Suspend
  7. Resume

Process State diagram

1. Initially process will be in the **new state**, it means that process is under creation or process is being created
2. once the process is created it will be moved to ready state & in the **ready state** there can be multiple no. of processes.
3. one of the process will be selected from the ready state & that will be dispatch to the running state
4. when the process is in the running state it occupys the cpu & execute the instruction of the process & performing cpu time
5. In the **run state** there will be one process at any point of time
6. If the running process required I/o operation then it will be moved on to **wait/Block** state
7. In the wait state also there can be multiple no of processes.
8. when the process is the Reading, Running, wait state it is Residing in the Main Memory
9. If the resource are not sufficient then the some of the processes will be suspended & they will be moved on to **suspend ready** State

• any transition blw the State is using a System call

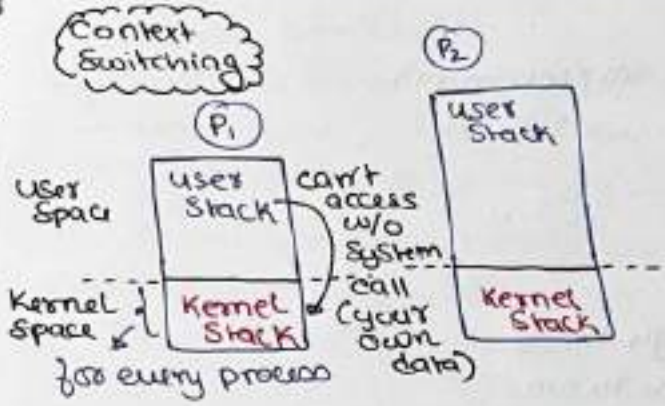
• if priority, time Quantum is not in the Diagram then its an non-preemptive process state Diagram
else its preemptive process state diagram.



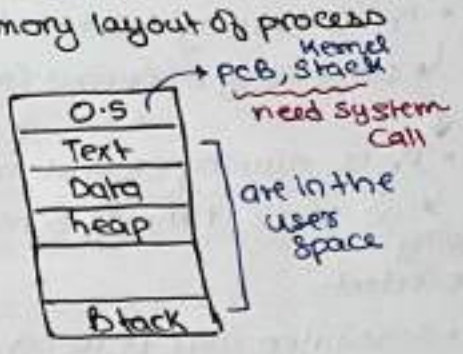
process is Completed I/O operation but suspended due to lack of resource.

i·e preemptive process state Diagram

Context Switching

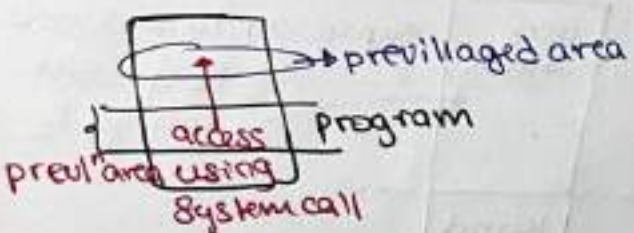


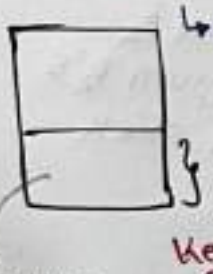
memory layout of process



- every process has some information to store
  - → PCB
  - → Registers in use [meta data]

where do we store this information
↳ in memory (previllaged area of memory)





↳ for every process, I want to store something in the previllaged memory

Kernel Stack { ↳ in that memory i am having some stack

one of the use case is that we can push the register value here at that time of context switching

- In user mode
  - ↳ P_1 is running.
    - P_1 time is over
    - P_1 want to leave cpu by choice
    - P_1 want to go in block State b/c of I/O.

B/C of one of this reason (System call/Interrupt) P_1 will be in Kernel mode.

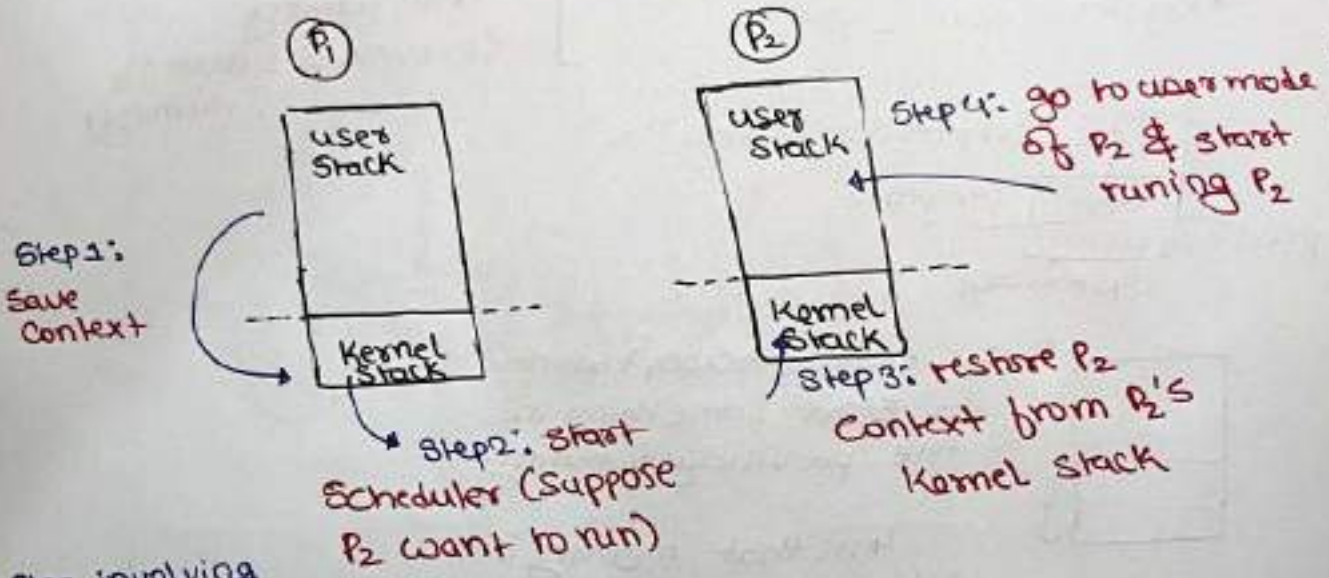
- P_1 is running (user mode)
  - Something happened (system call / Interrupt)
    - [Sched. Init.]
- P_1 is running (Kernel mode)
  - we start running Scheduler.

Saving the context

- Scheduler will be going through list of processes & deciding which to run.

Restore the context

now the next process start running



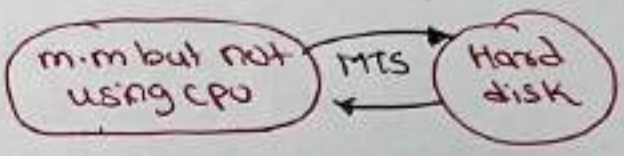
- Step involving
  - P_1 user mode
  - P_1 Kernel mode (I want to save P_1 context in previlleged area)
  - Scheduler
  - P_2 Kernel mode (I want to restore P_2 context)
  - P_2 user mode

- Long term scheduler
- if process is submitted to O.S then
  → where to put in ready queue
  → to put or not to put in ready queue

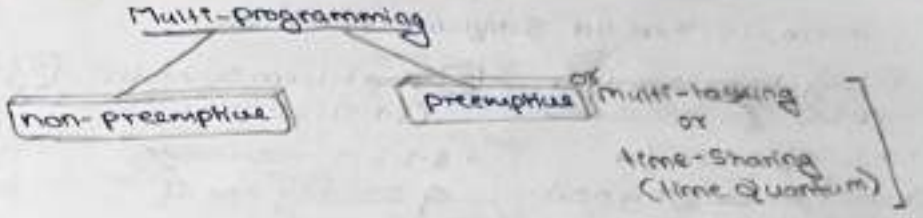
  } these kind of decision is taken by L.T.S
- L.T.S are invoked with very less frequency (sec. / min. / hours) etc.

we call it "long time" bic we invoke it after a long time
(since we invoke it occassionaly hence it's OK for us even take more time)

- Mid-term scheduler
- Scoop out the process from memory to harddisk



- Short term scheduler
- Select one of the process in the ready queue to be executed



10. When the process is in the suspend ready state then it is residing in the backing store [secondary memory].

- the process which required more amount of CPU time are known as CPU bound process
- these process will spend more time in running state

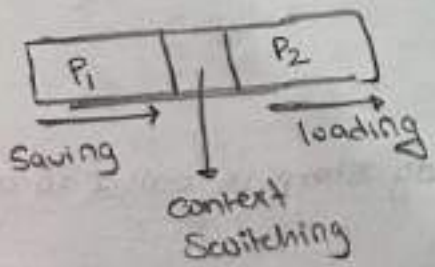
- the processes which required more amount of I/O time are called as I/O Bound processes
- these process will spend more time in wait/Block state

Degree of Multi-programming

- no. of process present in the Main memory at any point of time is known as degree of multi-programming

- each and every time when the processes is moving from one state to another state then the context of the process will change.

Context Switching



- Saving the context of one process & loading the context of another process is called as context switching
- if the context of the process is more than context switching time will also increase & which is undesirable.
- context switching time is consider as overhead of the system

note: In some special cases if there is only one process still it is called as context switching ex: Round Robin Schedule with 1 process

In the O.S there are 3 different schedulers:-

1) Long term scheduler (L.T.S) (Job Scheduler)
- It is responsible for creating or bring new process into the system

2) Short term scheduler (S.T.S) (cpu scheduler)
- S.T.S is responsible of selecting one of the process in the ready state for scheduling the cpu.

3) mid/medium term scheduler (M.T.S)
- M.T.S is responsible for suspend and resuming the processes.
- job done by M.T.S is called as swapping

Dispatcher
- It is responsible for loading the selective job onto CPU
- It is also responsible of context-switching performing

- Long term schedulers should select good combination of both CPU bound & I/O bound processes in order to get good throughput for the system.
- Long term schedulers control degree of multi-programming.

Q) Consider a system which as 'n' CPU processors then what is the min. & max. no of processors that may present in the Ready, Running & wait states.

	min	max
Ready	0	"Any"
Run	0	n
wait	0	"Any"

Any depend on max no. of multi-programming in the system (Depending on Main memory)

- process is having various different time

1. Arrival time (A.T) (submission time)
   the time when the process is arrived to the ready state is called as a arrival time of a process.
2. Completion time (C.T)
   the time when the process completed its total execution of the process is called completion time

[3] Burst time
- the time required by the process for its execution is called as Burst time as process

[4] Turn around time (T·A·T)
- the time difference btw C·T & A·T is called T·A·T of the process

T·A·T = C·T − A·T

[5] waiting time (w·T)
- the time difference b/w T·A·T & B·T is known as w·T

w·T = T·A·T − B·T

[6] Response time (R·T)
the time difference b/w 1st Response & A·T is called Response time

CPU scheduling

where: In Ready state
who: Short term scheduler
when:
1. Run → termination
   Run → wait
   Run → Ready
2. wait → Ready
3. new → Ready

Goal
1. to maximize the CPU utilization & throughput of the system
2. to minimize the avg. TAT, avg w·T & avg. Response time of the process.

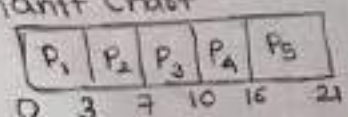
[1] First Come First Serve (F·C·F·S)

Criteria: Arrival time
mode: non preemptive

Q)

P·no	A·T	B·T	C·T	T·A·T	w·T
1	0	3	3	3	0
2	1	4	7	6	2
3	2	3	10	8	5
4	5	6	16	11	5
5	6	5	21	15	10

Gantt Chart



Avg. T·A·T = $\frac{28+15}{5} = 8.60$

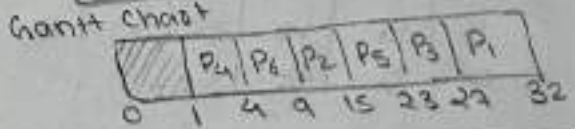
Avg. WT = $\frac{22}{5} = 4.40$

TAT = CT − AT
WT = TAT − BT

Q)

P·no	A·T	B·T	C·T	T·A·T	w·T
1	6	5	32	26	21
2	3	6	15	12	6
3	5	4	27	22	18
4	1	3	4	3	0
5	4	8	23	19	11
6	2	5	9	7	2

Gantt Chart



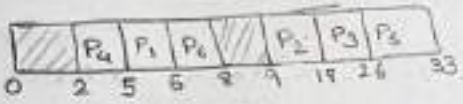
note: if the A·T of the process matching the schedule the process which have the lowest process ID

Avg TAT = $\frac{89}{6}$ → 14.83

Avg WT = $\frac{58}{6}$ → 9.66

Qus)

Process	A·T	B·T	C·T	T·A·T	W·T
P_1	3	1	6	3	2
P_2	9	9	18	9	0
P_3	4	8	26	15	7
P_4	2	3	5	3	0
P_5	13	9	33	20	11
P_6	3	2	8	5	3



Avg T·A·T = 55/6 = 9.16

Avg WT = 25/6 = 4.16

Convoy effect

In FCFS the BT process is having CPU bound process & followed by many I/O bound processes then it will have major effect on Avg. W·T of the processes. these effect is called as convoy effect.

note:- If B·T of the processes are matching then Scheduling the process which has lowest A·T

Qus)

Process	A·T	B·T	C·T	T·A·T	W·T
P_1	0	20	20	20	0
P_2	1	1	21	20	19
P_3	2	1	22	20	19

Avg W·T = 38/3 = 12.66

Qus)

Process	A·T	B·T	C·T	T·A·T	W·T
P_1	0	1	1	1	0
P_2	1	1	2	1	0
P_3	2	20	22	20	0

Avg W·T = 0

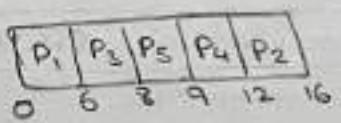
Shortest job first (S·J·F)

Criteria :- Burst time

mode :- non-preemptive

Qus)

P.no	A·T	B·T	C·T	T·A·T	W·T
P_1	0	6	6	6	0
P_2	2	4	16	14	10
P_3	4	2	8	4	2
P_4	6	3	12	6	3
P_5	7	1	9	2	1

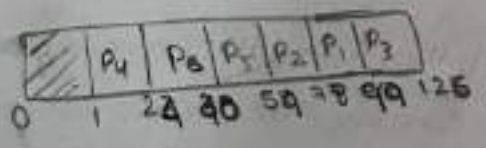


Avg T·A·T = 32/5 = 6.40

Avg W·T = 16/5 = 3.20

7 Qus)

P·NO	A·T	B·T	C·T	T·A·T	W·T
P_1	21	21	99	78	57
P_2	17	19	78	61	42
P_3	11	27	126	115	88
P_4	1	23	24	23	0
P_5	14	19	59	45	26
P_6	5	16	40	35	19



Avg TAT = 357/6 = 59.50

Avg WT = 212/6 = 35.66

Shortest Remaining Time First (S.R.T.F)

Criteria:- Burst time

mode:- preemptive

1 - X 16
2 - 18 15
3 - X 10
4 - X 6
5 - X 3
6 - X

Qus1

P.no	A.T	B.T	C.T	T.A.T	W.T
P_1	0	17	82	82	65
P_2	2	15	61	59	44
P_3	3	11	28	25	14
P_4	5	9	18	13	4
P_5	7	4	12	5	1
P_6	8	2	9	1	0

P_1	P_2	P_3	P_4	P_5	P_6	P_5	P_4	P_3	P_2	P_1
0	1	3	5	7	8	9	13	18	28	61 82

Avg T.A.T = $\frac{142}{5}$ = 28.66

Avg W.T = $\frac{85}{6}$ = 14.16

Qus1

P.no	A.T	B.T	C.T	T.A.T	W.T
P_1	15	23	95	80	
P_2	9	24	72		
P_3	12	17	29		
P_4	3	3	6		
P_5	10	22	49		
P_6	7	27	120		

	P_4		P_6	P_2	P_5	P_3	P_5	P_2	P_1	P_6
0	3	6	7	9	10	12	29	49	72	95 120

Gate 2016

Qus1

P.no	A.T	B.T	C.T	T.A.T
P_1	0	10	20	20
P_2	3	6	10	7
P_3	7	8	18	11
P_4	8	3	13	5

P_1	P_2	P_3	P_2	P_4	P_1
0	3	7	8	10	13 20

33/4 = 8.25

Qus1

P.no	A.T	B.T	C.T	T.A.T	W.T
P_1	18	27	107	89	62
P_2	9	20	55	46	26
P_3	15	11	26	11	0
P_4	0	7	7	7	0
P_5	13	15	39	26	11
P_6	4	24	80	76	49

P_4	P_6	P_2	P_5	P_3	P_5	P_2	P_6	P_1
0	7	9	13	15	26	39	55	80 107

Avg T.A.T = 255/6 = 42.5

Avg W.T = 148/6 = 24.66

Gate 2007

Qus1

P.no	A.T	B.T		C.T	T.A.T	W.T
P_1	0	20	~~0~~			
P_2	15	25	5	55	40	(15) W.T of P_2
P_3	30	10				
P_4	45	15				

P_1	P_1	P_2	P_3	P_2	P_4
0	15	20	30	40	55 60

Qus] consider 3 processes A,B,C to be scheduled as per SRTF algo. the process 'A' is known to be scheduled 1st. & when 'A' been running for '8' unit of time, the process 'c' has arrived. the process 'c' has run for '2' unit of time, then the process 'B' has arrived & completed running in '3' unit of time then what could be the min. B.T of processes 'A' & 'C'?

a) 15 & 6
b) 14 & 5
c) 10 & 5
d) 14 & 6
e) none of the above

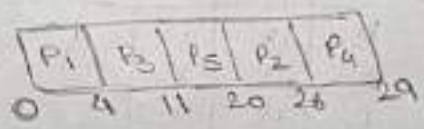
A → 8 + 7 = 15
B → 2 + 4 = 6
C → 3

Longest Job First (L.J.F)

criteria: Burst time
mode: non-preemptive

note
- If the B.T of the processes is matching the schedule the process which have lowest A.T

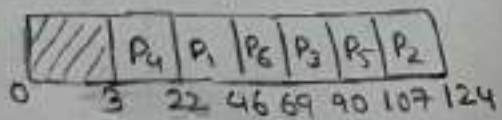
P.NO	A.T	B.T	CT	T.AT	WT
1	0	4	4	4	0
2	1	6	26	25	19
3	4	7	11	7	0
4	6	3	29	23	20
5	8	9	20	12	3



Avg. T.A.T = 71/5 = 14.20
Avg W.T = 92/5 = 8.20

Qus]

P.no	A.T	B.T	C.T	T.AT	WT
P_1	17	24	46	29	5
P_2	13	17	124	111	94
P_3	15	21	90	75	54
P_4	3	19	22	19	0
P_5	11	17	107	96	79
P_6	9	23	69	62	39



Avg T.AT = 389/6 = 65.33

Avg WT = 231/6 = 45 16

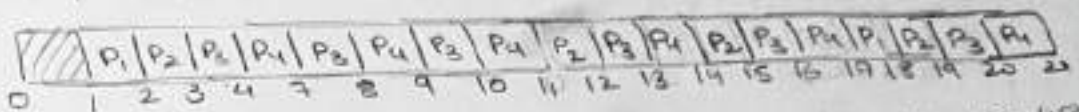
85
9
271

Longest Remaining Time first (L-R-T-F)

Criteria: Burst time
mode: preemptive.

Qus)

Pno	A.T	B.T	CT	TAT	WT
P_1	1	2 1	18	17	15
P_2	2	4 3	19	17	13
P_3	3	6 5	20	17	11
P_4	4	8	21	17	9

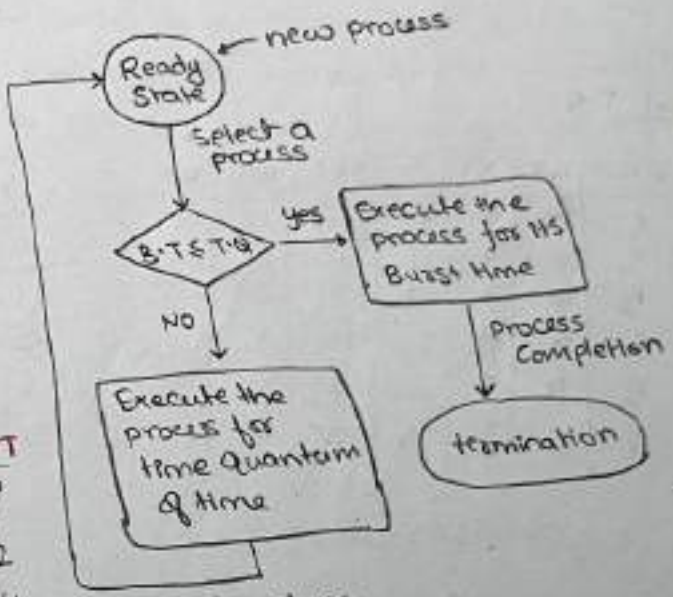


Avg TAT = 68/4 = 17 Avg WT = 48/4 =

Rath

Round-Robin Scheduling

Criteria: Time Quantum (or) time slice, Arrival time
mode: preemptive- preemptive



Qno) T.Q = 2

P.no	A.T	B.T	C.T	TAT	WT	R.T
P_1	0	4	8	8	4	0
P_2	1	5	19	18	13	1
P_3	2	6	21	19	13	2
P_4	4	3	18	14	11	4
P_5	5	2	14	9	7	7
P_6	6	1	15	9	8	8

$R.T = 1^{st}$ Response $- A.T$

Avg = 22/6 = 3.66

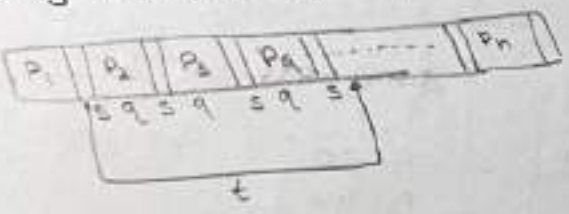
Ready queue:
$P_1, P_2, P_3, P_1, P_4, P_2, P_5, P_6, P_3, P_4, P_2, P_5$

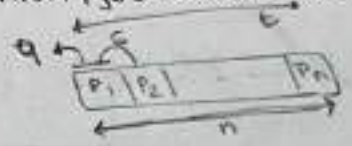
Avg TAT = 77/6 ⇒ 12.83 Avg WT = 56/6 = 9.33

Ques] Consider 4 process P_1, P_2, P_3, P_4 all arrive in the ready queue in the same order at time '0' the B.T of these process are 4,1,3,2 respectively what is the completion time of P_1 assuming RR of TQ: 1

P_1 4 3
P_2 1
P_3 3 2
P_4 2 1

Ques] Consider 'n' processes sharing the cpu in Round Robin algo. the context switching time is 's' unit then what must be the time quantum 'q' such that, the no. of context switches are reduced but at the same time each process is guaranteed to get it's turn/job at the cpu, after every 't' second of time?

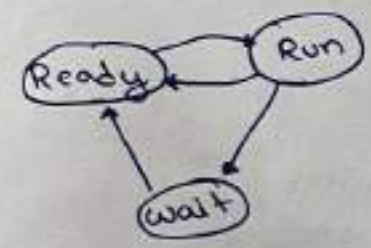




$t = n*s + (n-1)q$

$t - ns = (n-1)q$

$q = \frac{t-ns}{n-1}$



Life cycle
CPU Time
I/O Time
CPU time
.
.
.
Complete

Scenario's
1) CPU
2) I/O
3) CPU / I/O
4) CPU, I/O, I/O
5) I/O, I/O, I/O
6) CPU, CPU, I/O

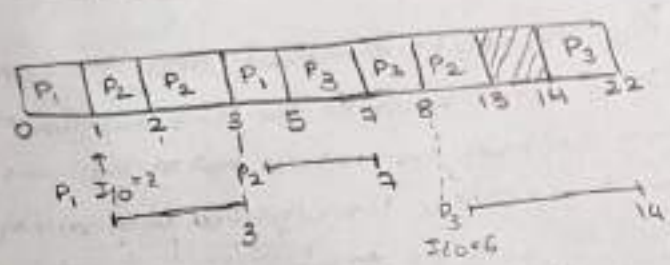
- CPU time, I/O time concept:

note: 1) the process first spend cpu time followed by I/O time & and followed by cpu time again.

2) I/O of the processes can be overlapped as much as possible

- CPU time, I/O time concept:

P.no	AT	Execution time: CPU time	Execution time: I/O time	Execution time: CPU time
P_1	0	20	2	2
P_2	1	2	4	5
P_3	2	3	6	8



Qus/ what is the completion time of the processes P_1, P_2, P_3 using SRTF algorithm?

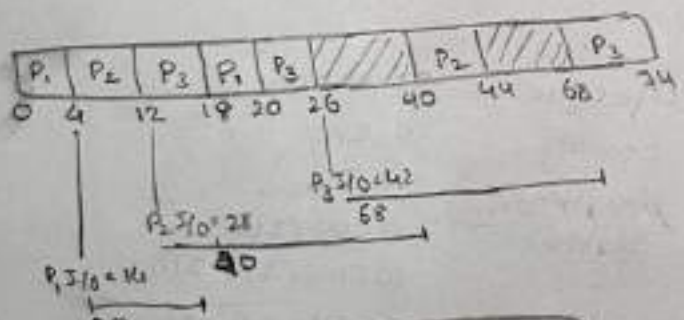
P_1 : 5
P_2 : 13
P_3 : 22

Using SRTF

P_1 : 20
P_2 : 44
P_3 : 74

Qus/

P.no	A.T	Execution: CPU time	Execution: I/O time	Execution: CPU time
P_1	0	4	14	2
P_2	0	8	28	4
P_3	0	12	42	6

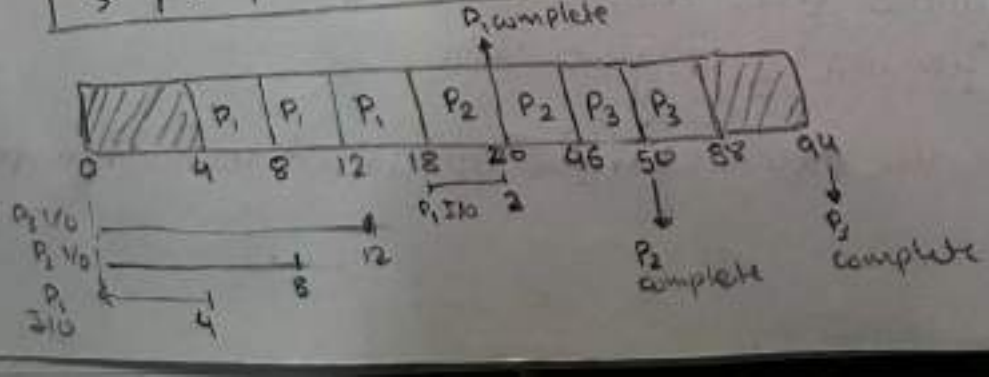


Using SRTF

P_1 : 20
P_2 : 50
P_3 : 94

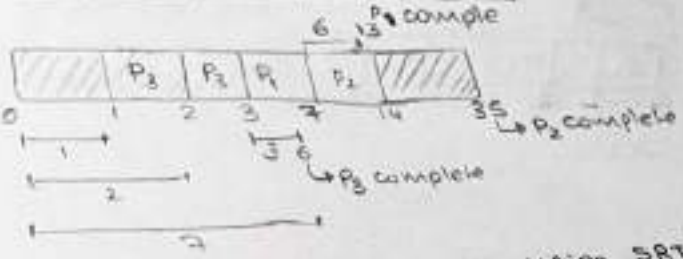
Qus/

P.no	AT	Execution time: I/O time	Execution time: CPU time	Execution time: I/O time
P_1	0	4	14	2
P_2	0	8	28	4
P_3	0	12	42	6



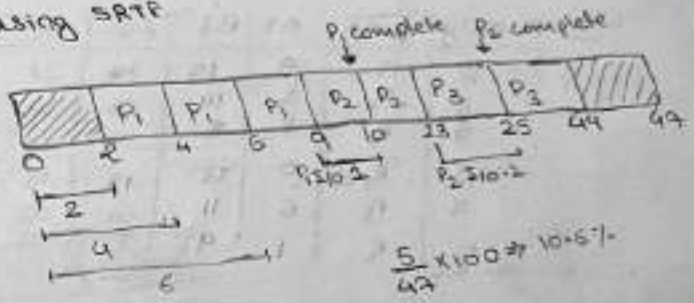
Using SRTF

P.no	AT	Execution time I/O	CPU	I/O
P_1	0	2	4	6
P_2	0	7	14	21
P_3	0	1	7	3



Using SRTF

P.no	AT	Execution I/O	CPU	I/O
P_1	0	2	4	1
P_2	0	4	14	2
P_3	0	6	21	3

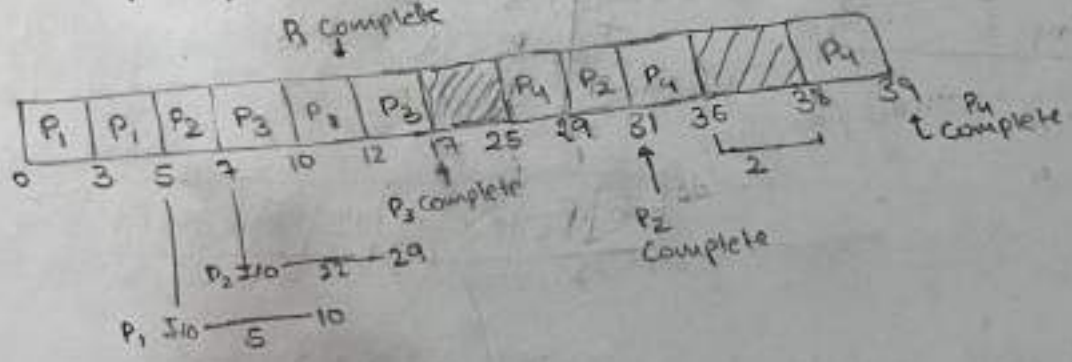


$\frac{5}{47} \times 100 \approx 10.6\%$

P.no	A.T	Execution CPU time	I/O	CPU
P_1	0	5	5	2
P_2	3	2	22	2
P_3	7	8	0	0
P_4	25	9	2	1

Using SRTF

P_1 : 12
P_2 : 31
P_3 : 17
P_4 : 39



Priority Based Scheduling

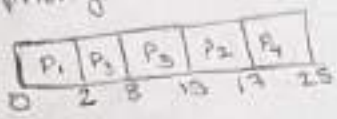
Criteria: priority
mode: non preemptive

note: If the priority of the process matching then schedule the process which have lowest A.T (Arrival Time)

Q.no	Priority	P.no	AT	BT	CT	TAT	WT
	3	P_1	0	2	2	2	0
	6	P_2	1	4	17	16	12
	7	P_3	2	6	8	6	0
	9	P_4	3	8	25	22	14
highest no is higher priority ←	9	P_5	4	5	13	9	4

Avg TAT = 55/5 = 11

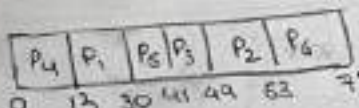
Avg WT = 30/5 = 6



Q.no	Priority	P.no	AT	BT	CT	TAT	WT
highest ←	10	P_1	9	17	30	21	4
	6	P_2	3	14	63	60	46
	7	P_3	8	8	49	41	33
	3	P_4	0	13	13	13	0
	7	P_5	6	11	41	35	24
	5	P_6	1	9	72	71	62

Avg TAT = $\frac{241}{6} = 40.16$

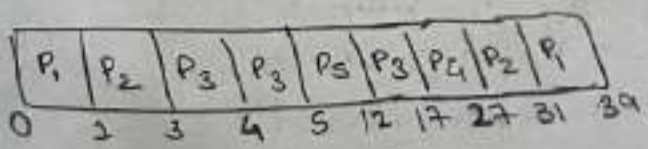
Avg WT = $\frac{169}{6} = 28.16$



Criteria: priority
mode: premmptive

using priority Based Scheduling

Q.no	Priority	P.no	A.T	B.T	C.T	TAT	WT	R.T
	3	P_1	0	9	39	39	30	0
	9	P_2	1	6	31	30	24	0
	5	P_3	3	7	17	14	7	0
	5	P_4	4	10	27	23	13	13
highest →	7	P_5	5	7	12	7	0	0



Avg TAT = 113/5 = 22.60

Avg WT = 74/5 = 14.80

Qus]

Priority	P.no	AT	BT	CT	TAT	WT
highest → 9	P_1	8	17	30	17	0
6	P_2	4	18	55	51	33
3	P_3	9	15	87	78	63
4	P_4	10	9	72	72	63
2	P_5	12	16	40	83	67
5	P_6	3	12	66	63	51

Avg TAT = $\frac{314}{6} = 52.33$

Avg WT = $\frac{222}{6} = 37.33$

P_4	P_6	P_2	P_3	P_5	P_1	P_5	P_2	P_5	P_4	P_5
0	3	4	7	9	13	30	40	55	66	72	87

29)

P.no	AT	CPU	I/O	CPU	I/O
P_1	0	10	90	10	90
P_2	0	10	90	10	90

a) FCFS →
b) SRTF →
c) priority →

P_1	P_2		P_1	P_2		P_1
0	10	20	80	110	120	200

d) RR with TQ = 5ms

P_1	P_2	P_1	P_2		P_1
0	5	10	15	20	105

BS

Highest Response Ratio next (H·R·R·N):-

criteria : Response Ratio
mode : non-preemptive

$R \cdot R = \frac{w+s}{s}$ or $\frac{w+BT}{BT}$

w = waiting time
s = service time or Burst time

apply/discuss this formula 1 row internal that is working & what all factors are affected by these parameters.

Qus]

P.no	AT	BT	CT	TAT	WT
P_1	0	3			
P_2	2	6			
P_3	4	4			
P_4	6	5			
P_5	8	2			

$R \cdot R_3 = \frac{w+s}{s} = \frac{(9-4)+4}{4} = \frac{9}{4} = 2.25$

$R \cdot R_4 = \frac{w+s}{s} = \frac{3+5}{5} = \frac{8}{5} = 1.6$

$R \cdot R_4 = \frac{1+2}{2} = \frac{3}{2} = 1.5$

P_1	P_2	P_3	P_5	P_4	
0	3	9	13	15	20

$R \cdot R_5 = \frac{9+2}{2} = 5.5$

$R \cdot R_4 = \frac{7+5}{5} = \frac{12}{5} = 2.4$

Note

these Algo. favors the shorter job & limits the waiting time of longer job

Qu)

P no	A T	B T	C T	TAT	W T
P_1	1	2	3	2	0
P_2	4	6	10	6	0
P_3	5	3	14	9	6
P_4	6	5	19	13	8
P_5	8	1	11	3	2

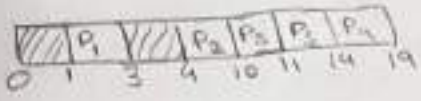
$RR_3 = \frac{5+3}{5} = \frac{8}{5}$

$RR_4 = \frac{6+3}{3} = \frac{9}{3}$

$RR_5 = \frac{4+5}{5} = \frac{9}{5}$

$RR_4 = \frac{5+5}{5} = \frac{10}{5}$

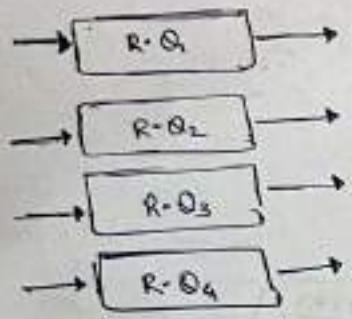
$RR_5 = \frac{2+1}{1} = 3$



Avg TAT = 53/5 = 6.6

Avg WT = 16/5 = 3.2

multi-level Queue Scheduling



- depending on the priority of the process in which ready Queue the process has to be placed will be decided
- the high priority process will be placed in the ready Queue & low priority process will be placed in the Bottom of the queue.
- only after completion of all the process from the top level ready queue the further level ready queue process will be scheduled
- If this is the strategy followed then the process which are placed in the bottom level ready queue will suffer from starvation

Starvation

- the indefinite waiting of a process is called as starvation

Note

- to avoid the problem of starvation the concept of ageing is used

Ageing

1. If the waiting time of process increase then priority of the process will increased
2. If the age of process increases by increasing the priority the process will definitly get a chance to execute

Multi level feedback Queue scheduling

- In above Algo every process will definitely get a chance to execute for some amount of time but still there is a possibility of Starvation

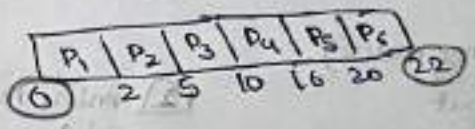
$$\text{throughput} = \frac{\#\text{ of processes}}{\max(CT) - \min(AT)}$$

$$\text{Schedule length} = \max(CT) - \min(AT)$$

Qus)

P.no	A.T	B.T
P_1	0	2
P_2	1	3
P_3	2	5
P_4	3	6
P_5	5	4
P_6	7	2

what is the throughput of system using FCFS Algorithm?



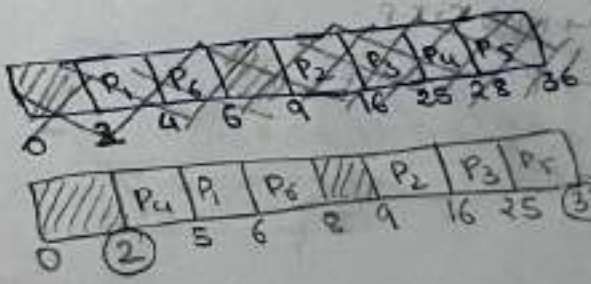
$\text{Throughput} = \frac{6}{22-0}$

$\Rightarrow 0.2727$

$\% = 27.27$

Qus)

P.no	A.T	B.T
P_1	3	1
P_2	9	7
P_3	11	9
P_4	2	3
P_5	13	8
P_6	3	2



$T_n = \frac{6}{33} = \frac{6}{33-2}$

$\Rightarrow \frac{6}{31} = .1935$

$\% = 19.35$

Algorithm	Starvation
1. FCFS	NO
2. NP → SJF	YES
3. SRTF	YES
4. R.R	NO
5. NP → L.J.F	YES
6. LRTF	NO
7. NP → priority	YES
8. P → priority	YES
9. HRRN	NO
10. ML.Q	YES
11. M.L.F.B.Q	YES

Note
majority of the O.S practical implement Round Robin scheduling (Multi level feedback Queue)
ex: window

- In general:
(without any value)

1. Best throughput
   - NP : SJF
   - P : SRTF

2. min Avg. TAT
   - NP : S.J.F
   - P : S.R.T.F

NP: non preemptive
P: preemptive

3. min Avg W.T
   - NP : S.J.F
   - P : S.R.T.F

4. min.avg Response time
   - Round Robin

- Dispatch latency — the time taken to stop a process & start another process
- Gang Scheduling – Thread scheduling
- Rate monotonic scheduling – Real time scheduling
- Fair share scheduling – guaranteed scheduling

Note

- In the SJF Algo. the B.T are predicate by using the following formula

$$\tau_{n+1} = \alpha T_n + (1-\alpha)\cdot \tau_n$$

τ_{n+1}: next expected B.T; T_n: actual previous B.T; τ_n: previous expected B.T

α: is a parameter controls recent & past history

$0 \le \alpha \le 1$

W-B page:170 32 & 33

(32)

P.no	BT
P_1	5
P_2	8
P_3	3
P_4	3

$\tau_1 = 10$

$\alpha = 0.5$

(1) $\tau_5 = ?$

$\tau_2 = \alpha \cdot T_1 + (1-\alpha)\tau_1$

$= 0.5 * 5 + (1-0.5) * 10$

$\tau_2 = 7.5$

$\tau_3 = \alpha \cdot T_2 + (1-\alpha)\tau_2$

$= 0.5 * 8 + (1-0.5) * 7.5$

$\tau_3 = 7.75$

$\tau_4 = \alpha \cdot T_3 + (1-\alpha)\tau_3$

$= 0.5 * 3 + (1-0.5) * 7.75$

$\tau_4 = 5.375$

$\tau_5 = \alpha \cdot T_4 + (1-\alpha)\tau_4$

$= 0.5 * 5 + (1-0.5) * 5.375$

$\tau_5 = 5.1875$

(33) if $\alpha = 1$

$\tau_5 = \alpha \cdot T_4 + (1-\alpha)\cdot \tau_4$

$= 1 * 5 + (\quad)$

$= 5$ //

- System call - Interrupt
- Command interpreter - user mode
- Wrapper - medium term schedule
- Dispatcher - context switching

- Volatile memory : when you switch off the power the content will erase or vanish

- Throughput will not always increase with increase in degree of multiprogramming b/c it will increase upto same point & then decrease [thrashing]

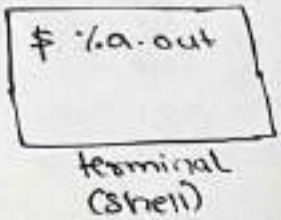
- System call are used in all area

- When an interrupt occur then process switching may be there and context saving must be there
- Special s/w to create a job queue is called Spooler
- on receiving an interrupt from an I/O device the CPU branches off to the ISR after completion of the current instruction
- Interrupt can't be handle in b/w the execution of an instruction. It will only be able to handle after completion of current instruction

- why does 'printf' need OS support?
→ b/c CPU doesn't own the monitor
→ monitor is external (called I/O device)
→ CPU has to take permission from monitor

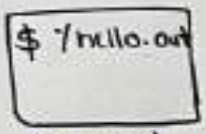
- Create new Process

ex. How to create a new process?
(How do OS load a process in M·M)



- it is not necessary that we need to run O·S code all the time
- O·S is always in M·M (loaded by bootloader at start of compute) but it doesn't mean that O·S is running all the time

ex `Printf(" ");`
for example this line can't be executed w/o interference of O·S
(using the system call we will do it)



1. create space for hello.out
2. load hello.out in main memory

Objective: understand how does the 2 steps get perform

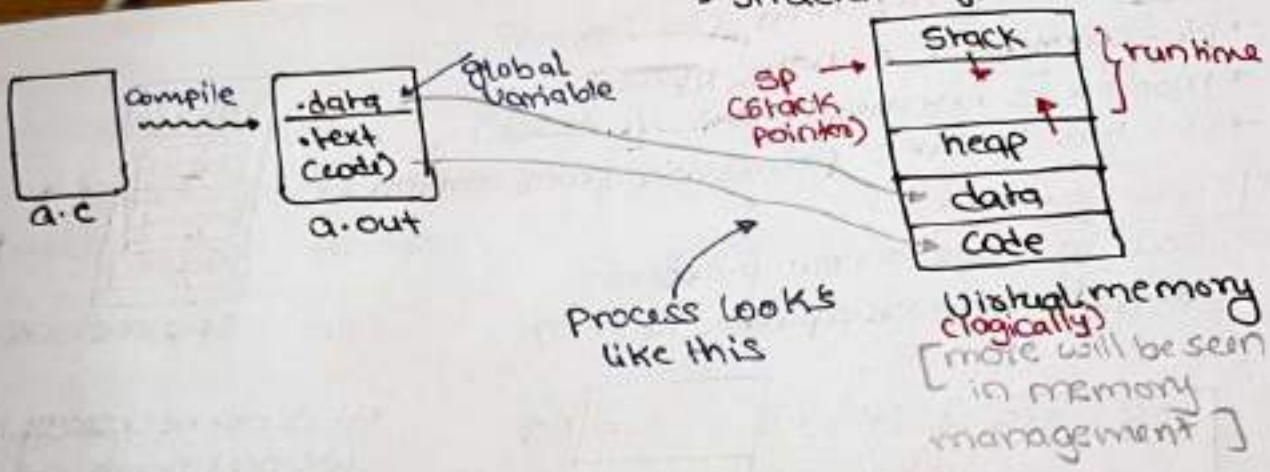
1. create space for hello·out
2. load hello·out in that space

} fork-exec can do these step

- A process running atleast one then it become program

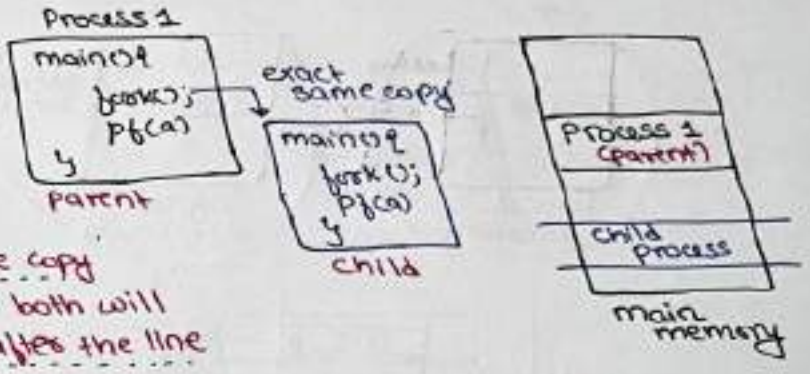
- A process..
  - ↳ program in execution
  - ↳ A process contests for resources
  - ↳ context of a process
  - ↳ Multiple process at same time

• structure of process



Fork() System Call

Create a exact replica of the calling process

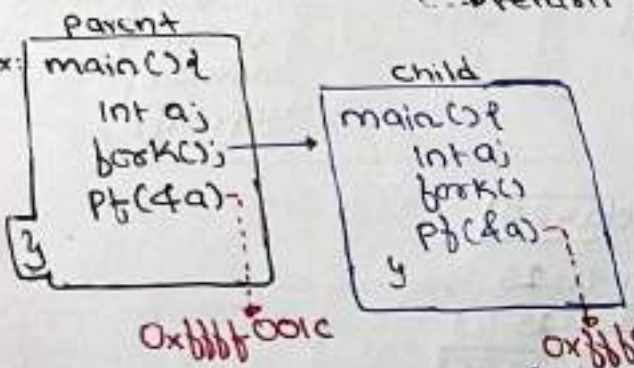


fork():-

1. create EXACT same copy
2. parent and child both will start executing after the line which had fork()
3. fork() will return some integer to both process
   - → return '0' to child
   - → return non-zero to parent
     - child process id

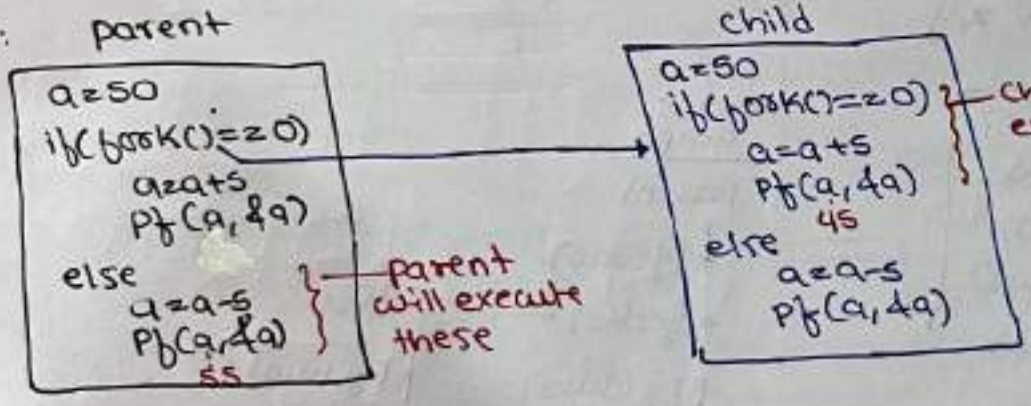
printf always print logical address

ex:



- Fork() System call is used to create a separate duplicate process. but the difference comes when or it have different process id
- returns '-ve' value if child process is unsuccessful

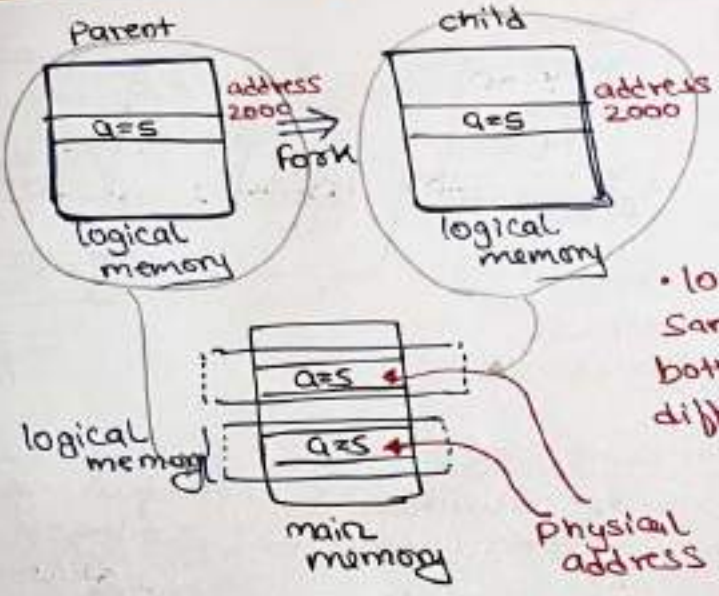
ex:



- it doesn't matter if variable is local (or) glob. al (or) static

- it doesn't matter the execution sequence if child → parent (or) parent → child

} what you are doing at parent doesn't matter to child & what child do doesn't matter to parent



- Both child & parent are execute in different, different 'a'
- logically both are same but physically both are present differently
- both have different logical memory but same logical address (printed by printf)

gate 2005

```
if(fork()==20)
    a=a+5
    pf(a,&a)
else
    a=a-5
    pf(a,&a)
```

child will run these

assume a=20

then u=15 & x=25

So, u+10=x

& v=y

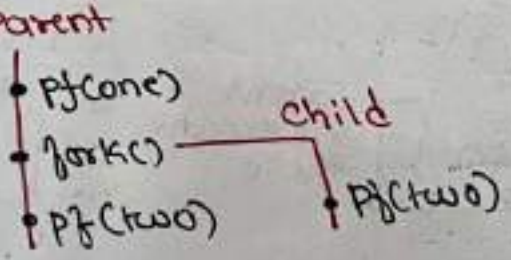
parent print u,v

& child print x,y

ex:

```
int main(){
  pf("one")
  fork();
  pf("two")
}
```

Process graph

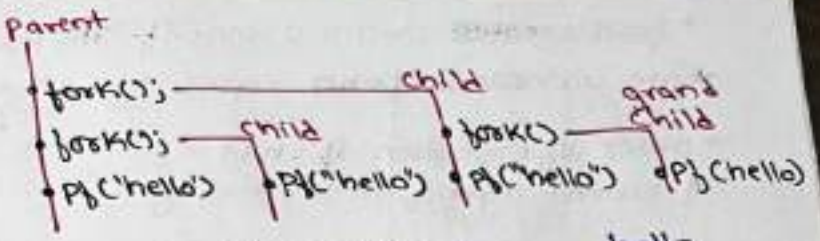


no. of processes = 2

no. of line

ex:

```
int main(){
    fork();
    fork();
    pf("hello");
}
```



- which hello is printed 1st can't be said depend on order of execution.

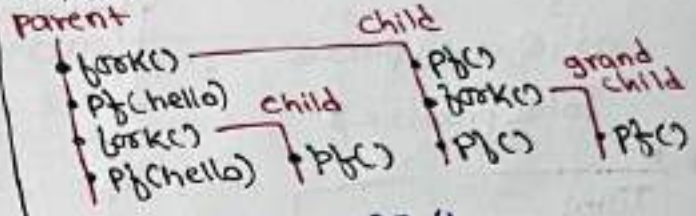
no. of process = 4

hello
hello
hello
hello

ex:

```
int main(){
    int i;
    for(i=0;i<2;i++){
        fork();
        pf("Hello");
    }
    return 0;
}
```

unroll there



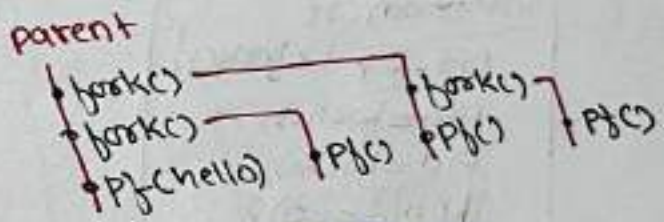
no. of process = 4

hello printed (6) time

ex

```
int main(){
    int i;
    for(i=0;i<2;i++){
        fork();
    }
    pf("hello");
}
```

unroll the loop

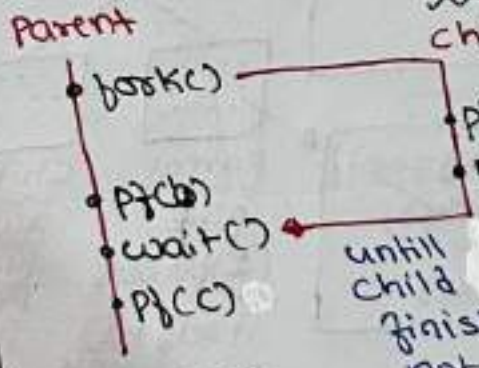


(hello) printed (4) time

ex

```
int main(){
    if(fork()==0)
        pf("a");
    else
        pf("b");
        wait();
    pf("c");
    exit(0);
}
```

whatever code is needed for child.



whatever code is needed for parent

until child unless child doesn't finish we will not processed

o/p
b a c c
a b c c
a c b c

3 pattern are possible

(b) can come any where

a c

- wait() → means parent will be waiting for the child

→ fork creates identical copy of address space (logical address)

→ we understood fork() copies by printing address & changing values

Same

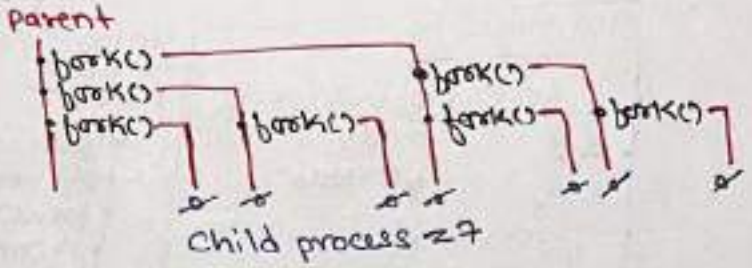
Change to one thing doesn't reflect in another process

→ order of execution of child & parent depend on scheduling

ex: for(i=1; i<=3; i++){
fork();
}

no. of child process.

total process = 8

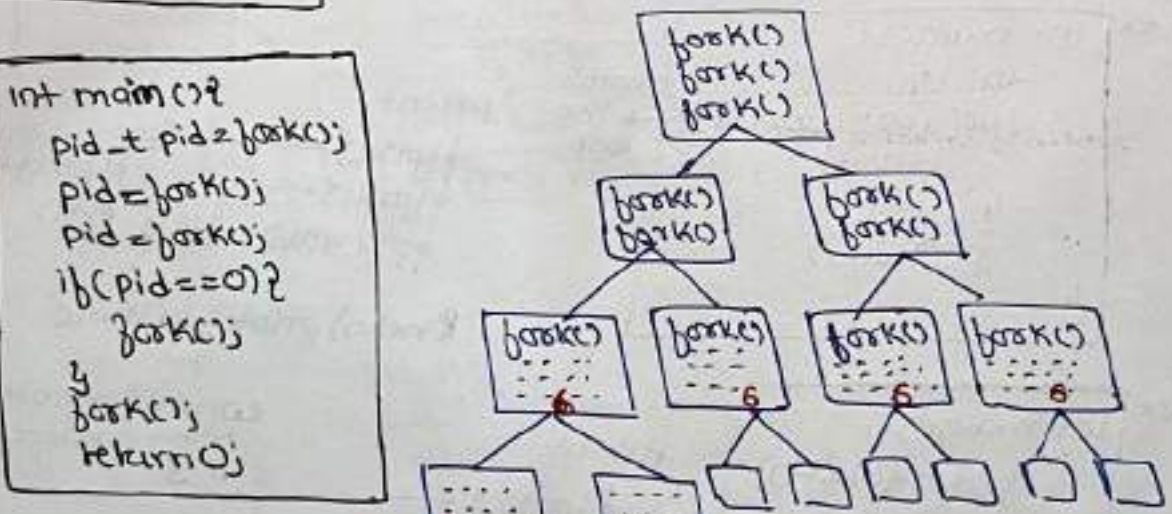


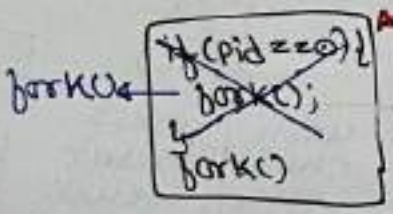
* *

Total no. of child process $= 2^n - 1$

ex:

```
int main(){
  pid_t pid=fork();
  pid=fork();
  pid=fork();
  if(pid==0){
    fork();
  }
  fork();
  return 0;
}
```

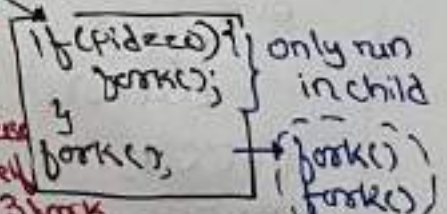




fork()

Parent give (2) process itself + (1) fork

Child give (4) Process itself + 3 fork

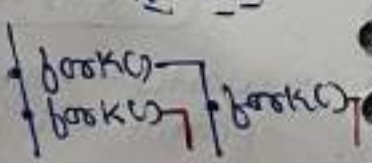


only run in child

So, total (6) process are there.

$ 6×4=24 process

from (1) such forkbox



ex

```
int main(){
    pf(A)
    int child = fork();
    pf(B)
    if(child) wait(NULL);
    pf(C);
}
```

Parent

```
int child = fork();
pf(B)
if(child) wait();
pf(C)
```

Child

```
pf(B)
pf(C)
```

True for parent as True (non-zero value)

O/P A, B, C, C

B can come anywhere

A B B C C
A B C B C

2 pattern

ex:

```
int a=0
int rc=fork();
a++;
if(rc==0){
  rc=fork();
  a++;
}else{
  a++;
}
pf(Hello)
pf(a)
```

rc==non-zero → Parent

```
a++
if(rc==0)
  rc=fork();
  a++;
}else{
  a++;
}
pf(hello)
pf(a)
```

rc==0 → Child

```
a++
if(rc==0){
  rc=fork();
  a++;
}else{
  a++;
}
pf(hello)
pf(a)
```

largest value of a?

So, max among all is ②

```
a++
else{
  a++;
}
pf(hello)
pf(a)
```

→ 2

```
a++
if(rc==0)
  rc=fork();
  a++;
}
pf(hello)
pf(a)
```

```
a++
a++
pf(hello)
pf(a)
```

→ 2

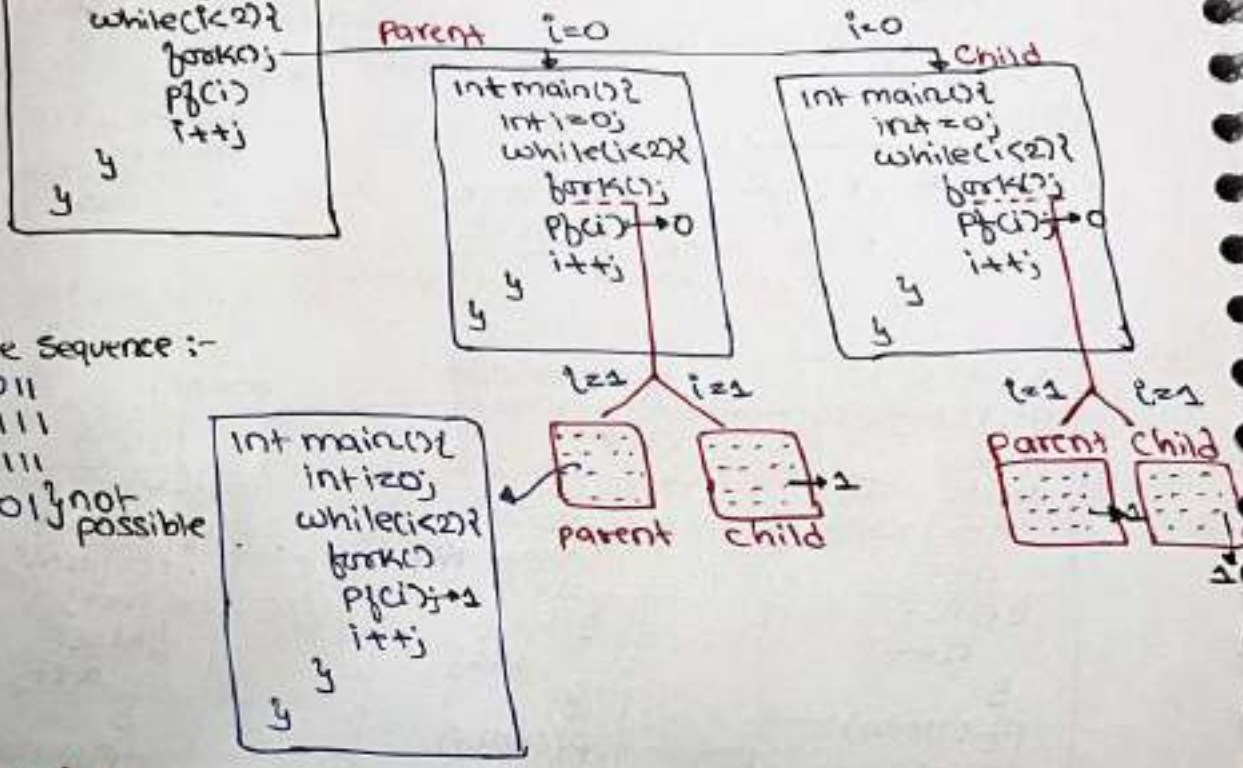
```
a++
a++
pf(hello)
pf(a)
```

→ 2

possible pattern it can print

```
int main(){
    int i=0;
    while(i<2){
        fork();
        pf(i)
        i++;
    }
}
```

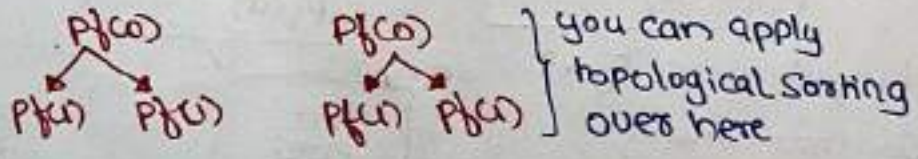
method 1:



possible sequence :-

011011
001111
010111
X011101 } not possible

method 2:

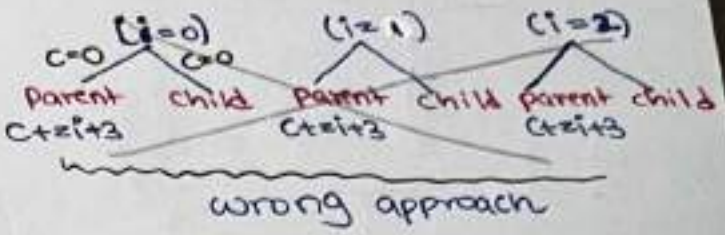


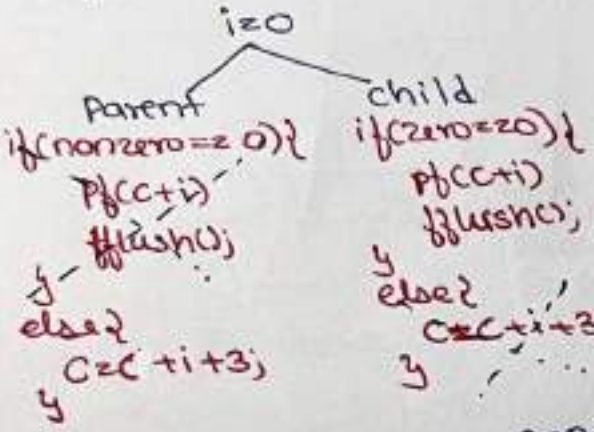
very tough question

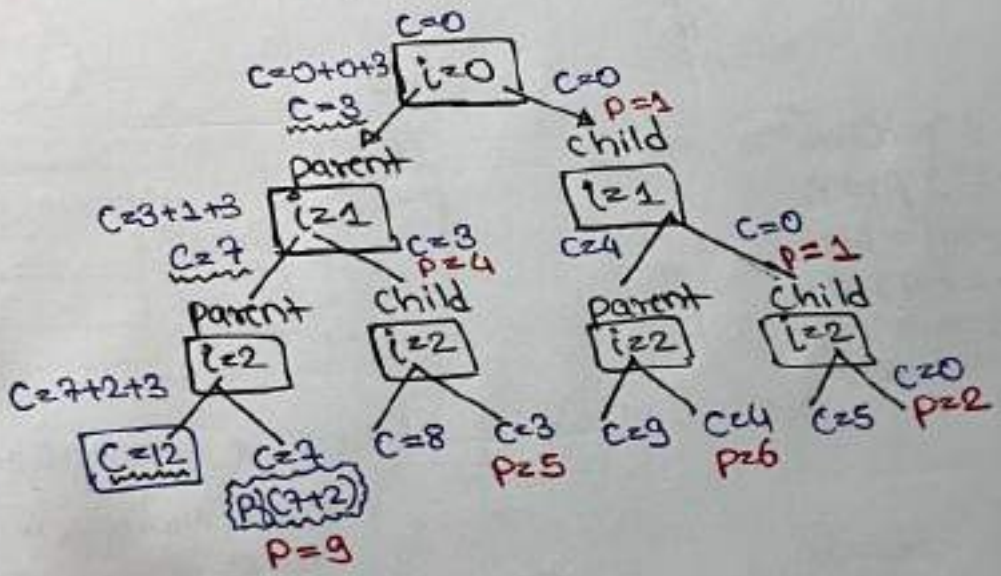
```
ex: int main(){
      int c=0;
      for(int i=0;i<3;i++){
        if(fork()==0)
          Pf(c+i)
          fflush(stdout);
        else{
          c=c+i+3;
        }
      }
    }
```

child part

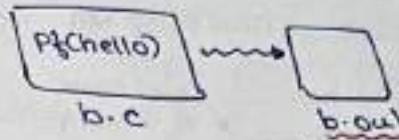
parent part



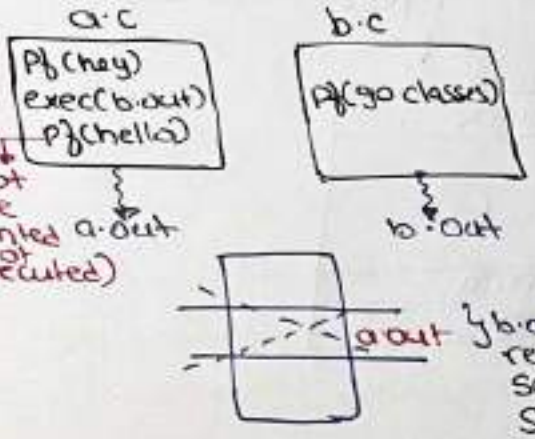


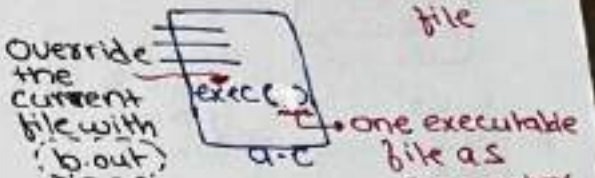


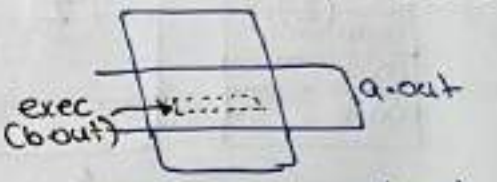
exec System call → will override/destroy the current file progress & run's the given file i.e execc....)



ex: what is the o/p we get if we run a.out file?







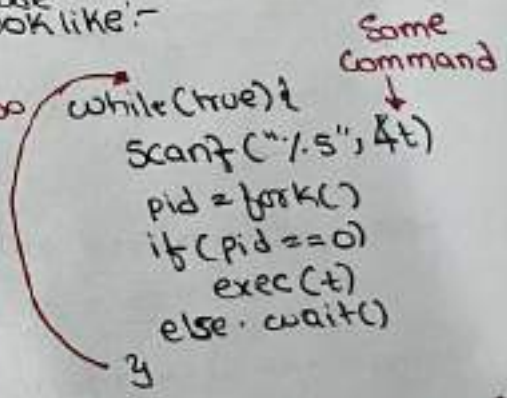
O/p: hey Goclasses.

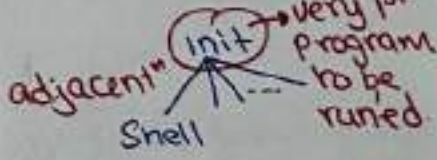
Process Creation

⇒ Create space for a.out

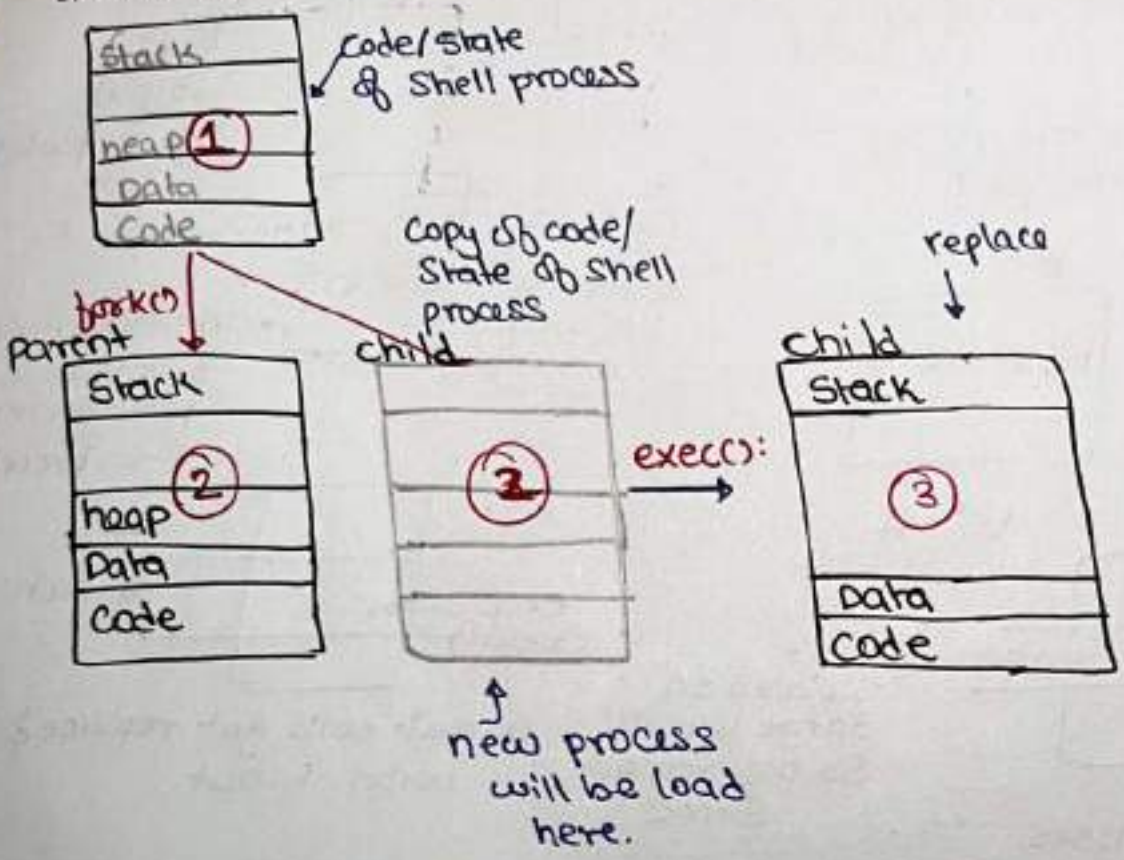
⇒ load the a.out in that space

Shell Code look like:-



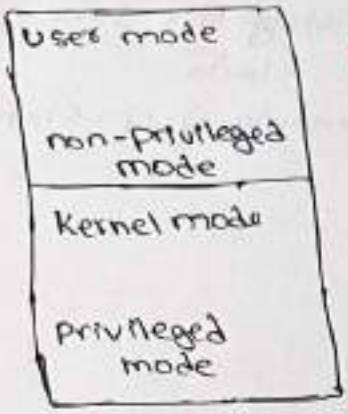


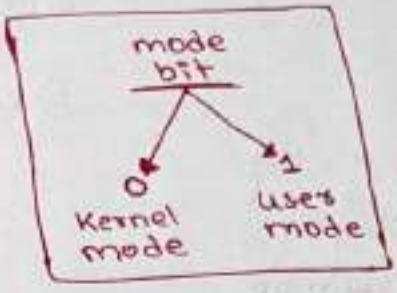
execute a new program



Dual mode operation

errant user → Hacker / unauthorized person





- In the H/w level the instructn are executed by using Dual mode operatn
  1) user mode/ non-privileged mode
  2) kernel mode/ privileged mode/ system mode / monitor mode / supervisory mode
- the Dual mode operation is used in order to provide protection & Security to the user program & also to the O.S from errant user
- It is purely decision of O.S in which particular mode the instruction has to be executed
- the privileged instruction are executed in kernel mode & non-privileged instruction are executed in user mode
- mode bit is used to identify in which particular mode the current instruction is executing
- In the booting time the system always start in the kernel mode
- O.S always run in the kernel mode

Important (need more security)
↑ privileged instruction

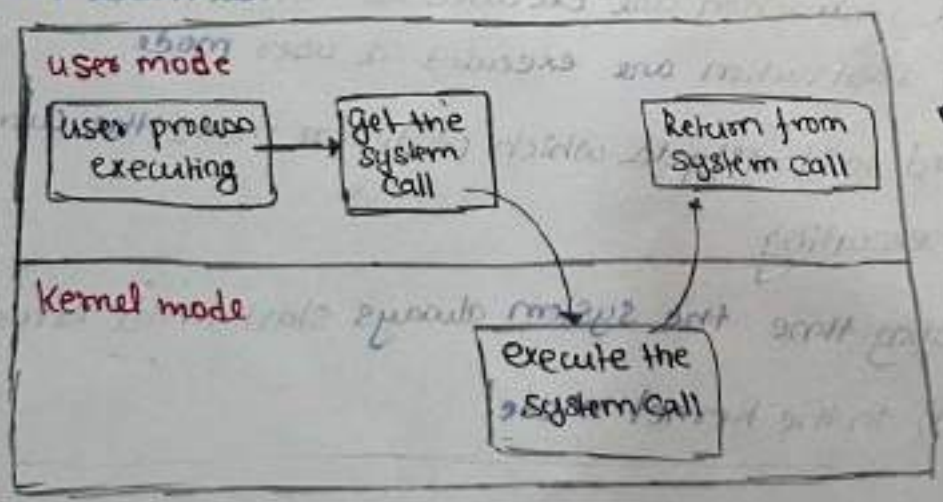
1) I/O operation
2) Context switching
3) Disable interrupts
4) Set the time of clock
5) changing the memory map
6) clearing the memory map

not important
↑ non-privileged instruction

1) Reading the time of clock
2) Reading of CPU status

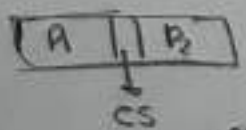
• A multi user, multi processing OS can't be implemented on h/w that doesn't support Atleast two modes of CPU execution (privileged & non-privileged)

System call • The Request made by the user program to the O.S inorder to get any kind of service



mode bit = 0 / 1

Note
• mode switching take very less time campare to process switching

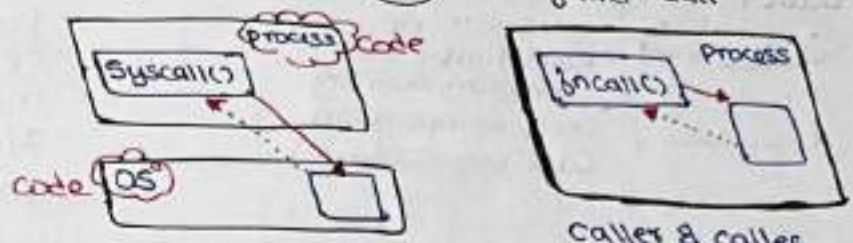


So C.S take more time b/c changing context from one process to another but in mode switching we just change 1bit only

11:50:00

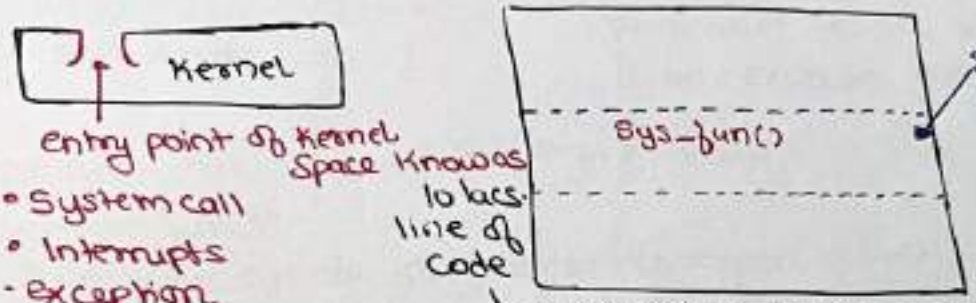
System call

System call (VS) funcn call



↳ OS is trusted, user is not
↳ OS has super-privileges user doesn't

Caller & callee are in same process
↳ Same user
↳ Same domain



entry point of kernel space

- System call
- Interrupts
- exception

Knows to kers. line of code

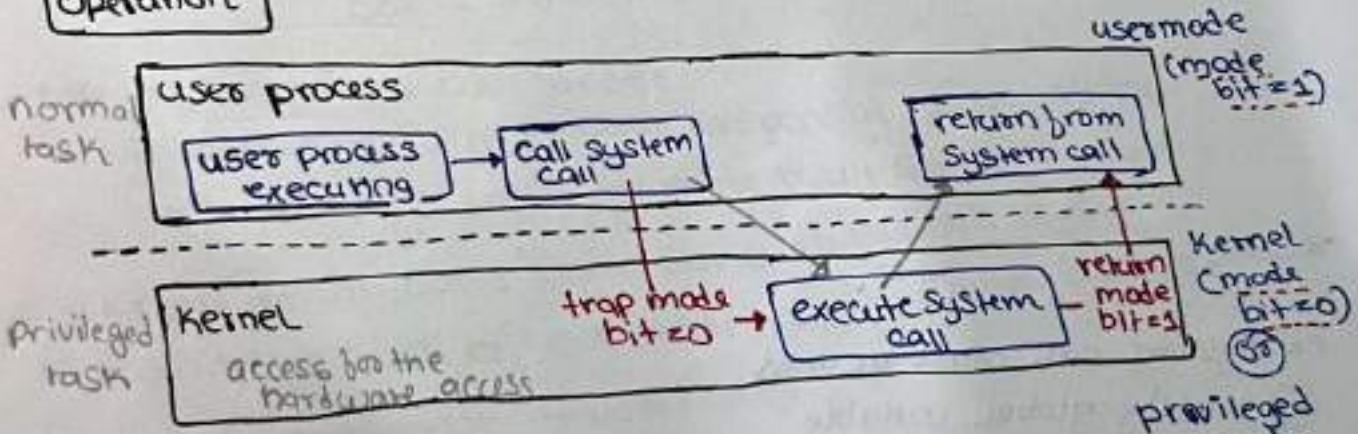
Some imp. piece of code is called Kernel

Some critical funct^n that O.S provide

Operating system code

- you can't enny Kernel from random space

System Call Operation



Dual mode of operation

(mode bit = 1)
- user mode
- Read & write into program memory
- Compute operation (addition, subtract^n)

(mode bit = 0)
- kernel mode
- CPU & memory management
- I/O handling
- critical instruction

ex: System call must be used
(a) modify global variable
(b) call a user written funct^n
(c) write to a file
(d) All of the above

System call Execution

what is system call

Soln: request to Kernel to perform a service

↳ open a file
↳ Execute a new program
↳ Create a new process
↳ Send a msg to another process

(To) entry point providing Controlled mechanism to execute kernel code

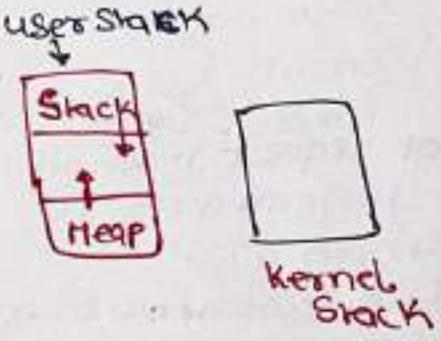
Syscall one of few mechanisms by which program can ask to execute kernel code

- Each system call has a unique numeric identifier
- OS has a system call table that maps numbers to functionality requested.
- When invoking a system call, user places system call no. & associated parameter in an specific register then executes the sys-call inst.

- program calls a wrapper funct^n in C library
- wrapper funct^n
  - packages sys call arguments into register or Stack
  - put (unique) syscall no. into a register
- wrapper flips CPU to Kernel mode (user mode → Kernel mode)
  - Execute special machine inst. (eg. sysenal)
  - Main effect: CPU can now touch memory marked as accessible only in Kernel mode
- Kernel execute Syscall handler
  - Invokes service routine corresponding to syscall no.
    - ↳ Do the real work, generate result status
  - place return value from service routine in a register
  - Switches back to user mode, passing control back to wrapper
    - ↳ (Kernel mode → user mode)

inside the SYSCALL inst^

- Save the old(user) stack pointer value
- Switch the SP to the Kernel Stack
- Save the old (user) PC value (= return addr)
- Save the old privilege mode
- Set the new privilege mode to '0'
- Set the new PC to the Kernel syscall handler



- System call are slower than funct^ call

Synchronization

Processes → Co-operative / Independant

P_1 affecting P_2 ; P_2 affected By P_1

- the execution of one process affect or affected by another process then those processes are said to be cooperative processes otherwise they are said to be independent processes.

- understanding synchronization:-

1. the problem arises not having synchronization blw the process
2. The condition to be followed to achieve synchronization
3. The solution [Correct / Incorrect]

Step 1 The problem arises not having synchronization blw the process

Producer — n=8 — count=3 — Consumer

IN 3 — out 0

0	x
1	y
2	z
3	
4	
5	
6	
7	

Buffer [0---N-1]

- In is the variable used by the producer to identify the next empty slot in the buffer
- out is a variable used by the Consumer to identify from where it has to consume the items.

```
int count=0; // initial buffer will be empty & produce should only execute first
Void producer(void){
int itemp;
While(TRUE){
while(count==N);
Buffer[in]=itemp
in=(in+1) mod N;
count=count+1;
}
}
```

I. load Rp, m[count]
II. INCR Rp
III. store m[count], Rp

```
void consumer(void){
int itemc;
while(true){
while(count==0);
itemc=Buffer[out];
out=(out+1) mod N;
count=count-1;
}
}
```

I. Load Rc, m[count]
II. DERC Rc
III. Store m[count], Rc

- count is a variable used by both producer & consumer to identify no. of item present in the buffer at any point of time.

Shared Resource.

1. Count variable
2. Buffer

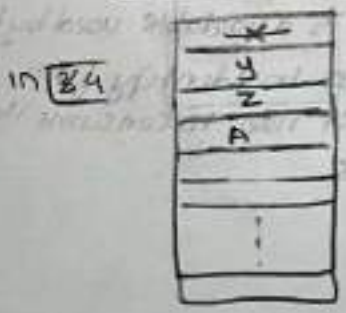
2 condition to be followed

1. If the Buffer is full the producer is not allowed to produce item into Buffer.
2. If the Buffer is empty consumer is not allowed to consume item from Buffer.

- Universal assumption.
  - the running process can get preempted at any point of time after completion of the the current instruction

1. Inconsistency.

Count = 874



in 84

out 01

Analysis

Item p = 'A'

Item c = 'x'

P → I
P → II

C → I
C → II
C → III

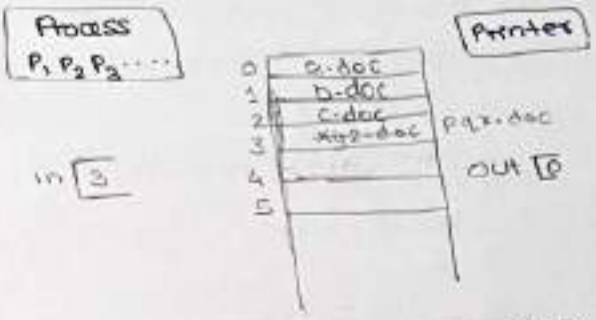
P → III

R_p 84

R_c 82

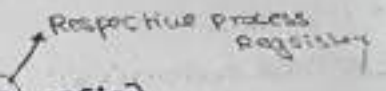
- the producer & consumer are not properly synchronize while sharing the common variable count. hence it is leading to the problem of Inconsistency

- Printer spooler daemon:
  - ↳ Background process



Spooler Directory (SD)

Enter file



I. load R_1, m[in]

II. Store SD[R1], "F-N"

III. INCR R1

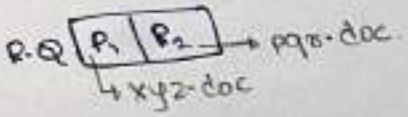
IV. Store m[in], R1

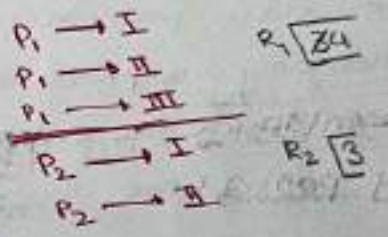
- "In" is a variable used by all the process to identify the next empty slot in the Spooler Directory
- out is variable used by the printer to identify from where it has to print the document

Shared Resource

1. Spooler Directory
2. "IN" variable

[2] Loss of data





- the process are not properly synchronized while sharing the common variable "IN" hence it is leading to the problem of loss of data

[3] Deadlock

- if the process are not properly synchronized by sharing the common resource as by sharing the common variable. so it is also possible for Deadlock

- Definitions

1. critical section (C.S):
the portion of program text, where share variable or shared resource will be placed
ex:- count = count + 1;

2. non-critical section (non-CS):
the portion of program text, where independent code of the processes will be placed
eg: IN = (IN+1) mod N;

3. RACE condition
the final value of shared variable depends on the execution sequence of the processes. these type of condition is called as race condition

P → I
P → II
C → I
C → II
C → III
P → III

C → I
C → II
P →
P →
P →
C → III

count = 4 | count = 2

Note: to avoid the problem of race condition only one process should be present inside the critical section at any point of time

• Mutual exclusion fail's then race condition occur's.

Step 2 Condition to be followed to achieve syncronization:

1. Mutual Exclusion (M.E):
i. no two processes should present inside the C.S at any point of time
ii. only one process is allowed to enter into C.S at any point of time
↓
only tell's if two process are there in C.S or not or only one

2. Progress:
i. no process running out side the C.S, should block the other interested process from entering into C.S when C.S is free.
ii. if there is only one process trying to enter into C.S then it should be definitely allowed into C.S.
iii. if two or more processes are trying to enter the C.S then one process should definitely allowed to enter in C.S

~progress → Deadlock ✗
~finite time → Deadlock ✓

↓
C.S empty or not
iv. the decision must come in finite time that who will enter into C.S

3. bound waiting (Bw):

i. no process should have to wait forever to enter into C.S

ii. there should be a bound on getting chance to enter into C.S } after certain no of process it get chance

iii. Some processes indefinitely waiting to enter into C.S b/c C.S is always busy by some other processes. These situation should not arise. [with respect to no. of processes]

iv. If the bound waiting is not satisfied then it is possible for starvation.

5. No assumption related to H/w & process speed

v. no. of processes entering into their C.S after a process request for the same must be bounded

Step 3 Solutions:

1. Software Type:
   - a) Lock variable
   - b) Strict alternation or Decker algorithm

2. H/w Type
   - a) TSL instruction set
     - ↳ test & set lock

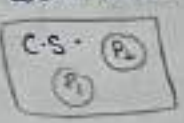
3. O.S Type
   - a) Counting semaphore
   - b) Binary semaphore

4. Programming language compiler support type:-
   - a) monitor's

- M.E fail then Race condition occurs
- progress fails then Deadlock may occur
  - ↳ progress (if finite time fails) in that case Deadlock will occur
- Bounded wait fail's then starvation may come

Analysis:

lock = 0 / 1



$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III

$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III
$P_2 \rightarrow$ IV
$P_2 \rightarrow$ V

$P_1 \rightarrow \overline{IV}$
$P_1 \rightarrow \overline{V}$

ME is not satisfied

e problem of race
one process should
e the Critical section
time

count=4

- Mutual exclusion fail's then race condition occur's.

o be followed to achieve syncronization:

on (M·E):
should pr
t ang poin
allowed to
oint of time
cess are
only one

[2] Progress:

- Bounded waiting & deadlock are 2 different Concept. So Bounded waiting doesn't imply no Deadlock. A System could Satisfy Bounded waiting & could be in deadlock as well.
- If a process is not using the critical section then it should not stop any other process from accessing it. (Independent of each other)

lock X
dlock ✓

that who will enter into C.S

3. bound waiting (B.W.):

i. no process should have to wait forever to enter into C.S

ii. there should be a bound on getting chance to enter into C.S } after certain no of process is get chance

iii. Some processes indefinitely waiting to enter into C.S b/c C.S is always busy by some other processes, these situation should not arise. [with respect to no. of processes]

iv. If the bound waiting is not satisfied then it is possible for starvation.

4. No assumption related to H/w & process's speed

v. no. of processes entering into their C.S after a process request for the same must be bounded

---

Step 3 Solutions:

1. Software Type:
   - a) Lock variable
   - b) Strict alteration or Decker algorithm

2. H/w Type
   - a) TSL instruction set
     - ↳ test & set lock

3. O.S Type
   - a) counting semaphore
   - b) Binary semaphore.

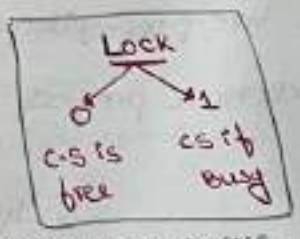
4. Programming language compiler support type:-
   - a) monitor's

---

1. Software type

a) Lock variable

I. Load Ri, m[lock]
II. cmp Ri, #0
III. JNZ to step (1)
IV. Store m[lock], #1
V. C.S
VI. Store m[lock], #0



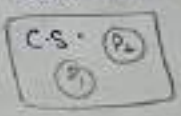
→ Respective process Registers

→ jump if it is non zero

→ Compare to 0

Analysis:

lock = 0 & 1



$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III

$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III
$P_2 \rightarrow$ IV
$P_2 \rightarrow$ V

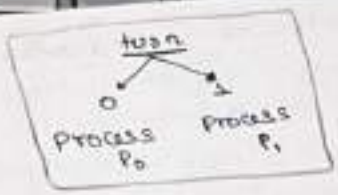
$P_1 \rightarrow \overline{IV}$
$P_1 \rightarrow \overline{V}$

ME is not satisfied

- we have prove that both the processes P_1 & P_2 are entering the C.S at the same time, hence ME is not satisfied & solution is consider to be incorrect

(b) strict alternation or packer algorithm :

(process takes "turn" to enter into c-s)



Process 'P0' code	Process 'P1' code

while(true){
non-cs();
while(turn!=0);
C.S
turn=1;
}
	while(true){
	no-cs();
	while(turn!=1);
	C.S
	turn=0;
	}

- we have proved that progress is not satisfied & solution is consider to be incorrect

Analysis :

turn = 0 || 1

M.E is satisfied

progress is not not satisfied

(turn is your) b/c chance is your & you don't want enter into C.S & you are also not letting me in the C.S

Important point

- preemption is just a temporary stop. the process will come back & execute the remain execution
- if there is any possibility of solution become wrong by taking the preemption then consider the preemption.
- if any solution have deadlock then progress is not satisfied

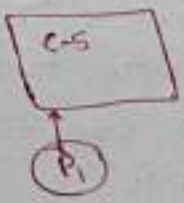
Qus!

P1	P2
while(S1==S2); C.S S1=S2	while(S1!=S2) C.S S2 = not(S1)

Analysis :

P1 : 0

P2 : 1



Progress failed

Peterson algorithm
(2 process solution)

```
#define N 2
#define True 1
#define False 0
int turn; // Shared variable used by both process (P0 & P1)
int interested[N];
void enter_Region(int process){
1. int other;
2. other = 1- process;
3. interested[process] = true
4. turn = process;
5. while (interested[other] == true && turn = process);
}
C.S
void leave_Region(int process){
    interested[process] = false;
}
```

P_0 is running at turn P_1

Analysis ①:-

Initially

interested[0] = false True

interested[1] = false True

turn = 0 1

Process P_0	Process P_1
other = 1	other = 0

C.S (P_0)

Analysis ②:- P_1 run at turn P_0

Initially

interested[0] = false True

interested[1] = false True

turn = ✗ 0

Process P_0	Process P_1
other = 1	other = 0

C.S (P_1)

- we have prove that all the 3 condition are satisfied in the above solution hence the solution is consider to be correct solution

So, ME is satisfied
also, progress is satisfied
Bound wait is also satisfied

Ques/ Assume that both the process P_0 & P_1 are trying to enter C.S at the same time then which process will entry in critical section (c.s) first

a) the process which execute statement '2' first
b) " " " " "3" first
c) " " " " "4" first
d) we can't say.

[2] H/w Type
↳ TSL Instruction set
(Test & set lock)

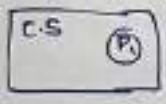
• TSL Register, flag : Copies the current value of flag into register & store the value of '1' into flag in a single atomic cycle without any preemption

• Entry section •

I. TSL Ri, m[flag]
II. comp Ri, #0
III. JNZ, to step (1)
IV. C.S
V. Store m[flag], #0

• Analysis:- •

Flag = 0 / 1



$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III
$P_1 \rightarrow$ IV

R_1 [0]

$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III

R_2 [1]

1. ME is satisfied
2. progress is satisfied
3. • Bound wait is not satisfied

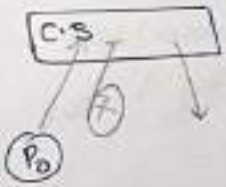
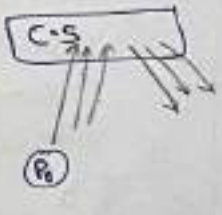
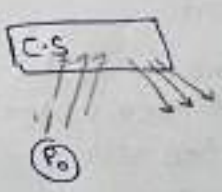
note If some process is in C.S then all the other process which are try to enter C.S will Repeatedly checking for I, II, IV

2. • there process are Busy in checking those instruct & waiting to enter C.S. these is called as Busy waiting

• Busy wait is also called as "Spin Lock" with the Busy waiting are waiting the cpu cycles (cpu time)

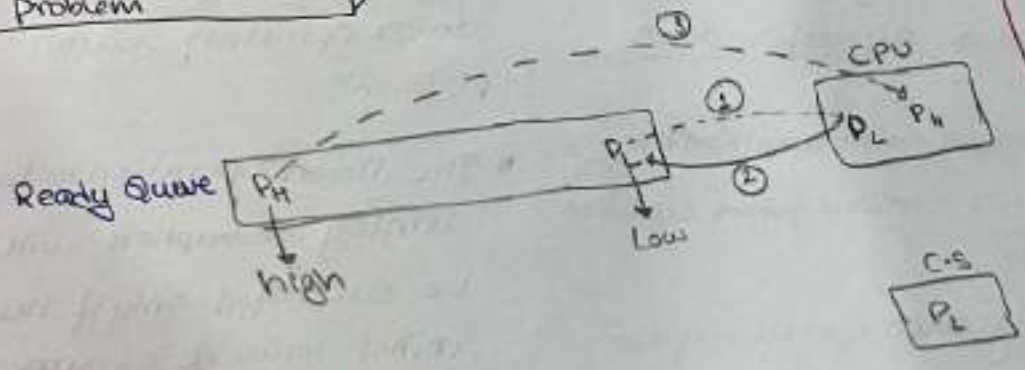
• avoid the problem for Busy waiting concept Concept of "Semaphore" is used

Solution	ME	Progress	B.W
• lock variables	×	✓	×
• Strict alternation (or) Deckers algo	✓	×	✓
• Petersons algo	✓	✓	✓
• TSL instruction	✓	✓	×

Strict alternation	Petersons algo	TSL instruction
Processes → P_0, P_1	Processes → P_0 P_1	Processes → P_0, P_1
progress (×)	progress (✓)	progress (✓)
		

• we can apply the above technique, only after checking deadlocks.

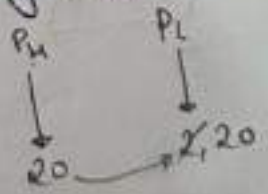
Priority Inversion problem



Live lock
P_L → ready
P_H → Run

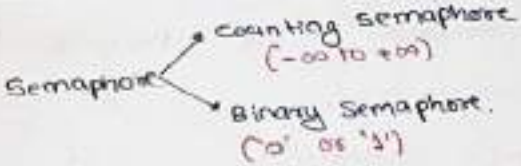
Solution:

Priority Inheritance



3] O-S Type

- Sempphore
- Semaphore is an integer variable used by various process in a mutual Exclusive manner to achieve synchronization.
- Improper usage of semaphore will also give wrong result.



- the two different operation performed of the semaphore variable

1] down() / wait() / P()

2] up() / signal() / V() / Release()

Counting Semaphore

```
down(semaphore S){
    S.value = S.value - 1;
    if(S.value < 0){
        Block the process & place it's PCB in the Suspended list
    }
}
```

```
up(Semaphore S){
    S.value = S.value + 1
    if(S.value ≤ 0){
        Select a process from Suspended list & wakeup();
    }
}
```

- after performing down operation if the process is not getting suspended then it is called as successfull down operation
- if it is successfull down operation then only process will continue further execution in the code
- after performing down operation if the process is getting suspended then it is called as an unsucessfull down() operation
- if it is unsuccessful down operation then processor will not continue further execution of the code

- positive value of semaphore indicator no of successful down operation you can perform
- The Down operation in the counting semaphore will be successful only if the inital value of Semaphore is

$S \geq +1$

- there is no unsuccessful up() operation, up() operation is always successful
- the process performing up() operation will definitely continue further execution.

Q.) consider a system were initially value of counting semaphore S = +17. then various semaphore operation like 23P, 14V, 13P, 7V, 2P then what is the final value of counting semaphore

⇒ +17 - 23 + 14 - 13 + 7 - 2

⇒ 0

Note: the -ve value of the counting semaphore indicates the no. of processes blocked (in the suspend list)

ex:

S: +2, 0, -1, -2

S-2 (P2 P3)

P1	P2	P3
P	P	P
✓	X	X

Q.) Consider counting semaphore variable S. the various semaphore operation like 20P, 12V are performed then what is the largest initially value of S. so that one process will remain blocked

S - 20 + 12 = -1 → no. blocked process

S = 7

Binary Semaphore

```
Down(Semaphore S){
    if(S.value==1)
        S.value=0;
    else{
        Blocked the process &
        placed it's PCB in the
        suspended list.
    }
}
```

```
up(semaphore S){
    if(suspended list is
                empty)
        S.value=1;
    else{
        select a process from
        suspended list and
        wakeup();
    }
}
```

- after performing Down() operation if the process is not getting suspended then it is called as successful down() operation
- If it is a successful down operation then the process will continue further execution in the code
- after performing down() operation if the process is getting suspended then it is called as unsuccessful down() operation
- if it is a unsuccessfull down operation then the process will not continue further execution in the code.
- Down() operation in they Binary Semaphore will be successfull if the initially value of the Semaphore is "1".
- there is no unsuccessful up() operation, up() operation is always successfull
- the process performing up() operation will definitely continue further execution

Ex: S = 1 0 1

P_1	P_2	P_3	P_4	P_5	P_6	P_7
P	P	P	V	V	V	V
✓	X	X	✓	✓	✓	✓

Successful ↳ unsuccessful

S.L [P_2] [P_3]

W-8
22

S = X, ϕ, X, ϕ, X (O)

9P, 14V, 6P, 8V, 3P, 2V

S.L ϕϕϕϕϕ

- Important points

(Applicable for both binary & counting semaphore)

1. every semaphore variable will have it's own suspend list
2. the down() & up() operation are atomic
3. if two or more processes are in the suspend list & there is no other process to wakeup() those processes then they are said to be involve in the deadlock
4. if more than one process is in the suspend list & every time when we perform 1 up() operation, one process will wakeup() from suspend list & these will based on first in first out
5. when the process is in the suspend list then it will be placed in the wait state & when we wake up the process then it will be brought into ready state

Note

```
Struct semaphore {
    int value;
    QueueType L;
};
Struct semaphore S;
```

- Semaphore is structure variable which takes integral value.

Ques] Each process P_i; $i = 1$ to 9 execute the following code

```
while (True) {
    P(mutex);
    C.S
    V(mutex);
}
```

mutex = X 0

S = X 9 X 8 X 1

The process 'P_{10}' execute the following code

```
while (True) {
    V P V(mutex);
    C.S
P   V V(mutex);
}
```

Analysis

mutex = X 0 X 1 X 2 X 3 X 0

what is the max no. of processes that may present inside the critical section at any point of time?

Note: Initial value of binary semaphore mutex = 1

a) 2 b) 3 c) 10 d) 3 e) 1

until you do VCD the process will remain in C.S

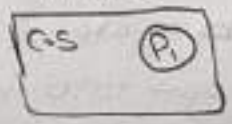
- Correct solution for 'n' process

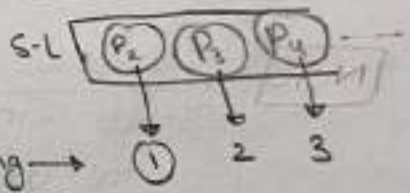
```
while (True) {
    P(mutex);
    C.S
    V(mutex);
}
```

initial value of binary semaphore mutex = 1;

1. ME ✓
2. P ✓
3. B.W ✓

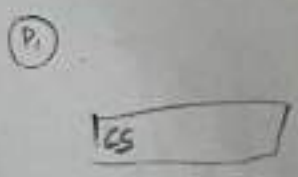
mutex = X, 0





waiting for process → 1 2 3

- after coming from suspend list P_2 will not executed the same instruction again it will execute from the next instruction in that case the next instruction is C.S.



Ques] Consider two concurrent processes 'P' & 'Q' executes their respective code S=0 T=1

Process 'P' code	Process 'Q' code

while (true){
w: P(T)
print('0');
print('0');
x: V(S)
}
	while(True){
	y: P(S)
	print('1');
	print('1');
	z: V(T)
	}

a) what should be the binary semaphore operation on w, x, y, z respectively & what must be the initial value of binary semaphore 'S' & 'T' inorder to get print o/p always as 00110011.....

×a) w=P(S), x=V(S), y=P(T),
z=V(T), S=1, T=1

× b) w=P(T), x=V(S), y=P(S),
z=V(T), S=1, T=1

× c) w=P(S), x=V(T), y=P(T),
z=V(S), S=0, T=1

✓ d) w=P(T), x=V(S), y=P(S),
z=V(T), S=0, T=1

Ques] which of the following will ensure that the o/p string never contain substring of the form 01^n0 or 10^n1, where n is odd?

Process 'P' code	Process 'Q' code

while (True){
w: ____
Print ('0');
Print('0');
x: ____
}
	while (True){
	y: ____
	Print('1');
	Print('1');
	z: ____
	}

a) w=P(S), x=V(S), y=P(T), z=V(T)
S=1, T=1

b) w=P(T), x=V(T), y=P(S),
z=V(T), S=1, T=1

c) w=P(S), x=V(S), y=P(S), z=V(S)
S=1

d)

10 B Pg-hr

16]

P_0	P_1	P_2
while(true)?	wait(S_1);	wait(S_2);
wait(S_0)	Release(S_0);	Release(S_0);
Print '0';		
Release(S_1);		
Release(S_2);		
}		

Initialized as
$S_0=1$, $S_1=0$, $S_2=0$.

Atleast twice

Analysis:

Case 1 [atleast 2]:

$S_0 - 1\ 0\ 1\ 0\ 1\ 0$
$S_1 - 0\ 1\ 0\ 1\ 1$
$S_2 - 0\ 1\ 1\ 0\ 1$

$P_0 \rightarrow P_1 \rightarrow P_0 \rightarrow P_2 \rightarrow P_0$

000

Case 2:

$S_0 - 1\ 0\ 1\ 1\ 0$
$S_1 - 0\ 1\ 0\ 1$
$S_2 - 0\ 1\ 0\ 1$

$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0$

00

1 1
0 0
0 0

17] initialized to 1. S_x & S_y
1 1

P_1	P_2
while true do	while true do?
L_1:	L_3:
L_2:	L_4:
$x = x+1$	$y = y+1$
$y = y-1$	$x = y-1$
$V(S_x)$	$V(S_y)$;
$V(S_y)$	$V(S_x)$;
}	}

- Classical problem of IPC (Inter process Communication)

1. Producer - Consumer semaphore

Producer | N = 8 | Consumer

in 8%8

out 0%2

Analysis:

mutex: 1

	Initial value	Produce item	Consume item	Produce & then Preempt & Consume item
Empty =	8	4	5	4 5
full =	0	4	3	2 3

8	8	8	8

itemp = 'A' B

itemp = 'A' y

```
#define N 8
Semaphore Empty = N;
Semaphore Full = 0;
Semaphore mutex = 1;
Void producer (void){
    int temp;
    while(TRUE){
        down(Empty);
        down(mutex);
        Buffer[in] = itemp;
        in = (in+1) mod N;
        up(mutex);
        up(full);
    }
}
```

Buffer check condition

Buffer Full

Full = 8

Empty = 0, -1

```
Void Consumer (void){
    int itemc;
    while(true){
        down(full);
        down(mutex);
        itemc = Buffer[out];
        out = (out+1) mod N;
        up(mutex);
        up(Empty);
    }
}
```

Buffer Empty

Full = 0, -1

Empty = 8

- empty is a counting semaphore variable represent the no. of slot empty in the buffer at any point of time.
- Full is a counting semaphore variable which represent #f of slot full in the buffer at any point of time.
- mutex is a binary semaphore variable used by producer-consumer to access the buffer in a mutual exclusive manner.

Qus] what happen when we interchange down(Empty) & down(mutex) in producer-consumer problem

a) no problem, the soln still work correct.

b) Both producer & consumer will access buffer together

c) some of the item produced will be lost

d) It is possible for deadlock.

Buffer is full :-

mutex = 1 0

Empty = 0 - 1

Full = 8 7

∴ it is possible for deadlock.

Qus] what happens if up(mutex) & up(full) are interchange.

mutex = 1 0

Empty = 0 1 2 3 4 5

Full = 7 6 5 4 3

∴ no problem, the Solution still work correct

from consumer

Qus] interchange down(full) & down(mutex)

Analysis:

mutex = 1 0 5-1

Empty = 8 7

Full = 0 5-1

∴ it is possible for deadlock

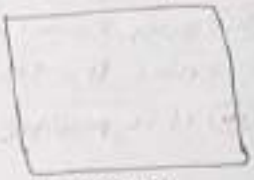
Ans] Interchange up(mutex) & up(Empty)

∴ no problem, the Solution still work correct

Readers-writers

Reader
$R_1, R_2, R_3 \ldots$

writer
$W_1, W_2, W_3 \ldots$



Database

Analysis ①: rc = 0 / 1
mutex = 1 ✗ 0 ✗ 0 ✗ 1
db = 1 ✗ 0

R → W ✗
R → R ✓

Analysis ②:
rc = 0 ✗ 1
mutex = 1 ✗ 0
db = 1 ✗ 0

W → R ✗
W → W ✗


```
int rc=0;
Semaphore mutex=1;
Semaphore db=1;
Void reader(void){
    while(true){
        down(    mutex);
        rc=rc+1;
        if(rc==1) down(db);
        up(mutex);
        D.B
        down(mutex);
        rc=rc-1;
        if(rc==0) up(db);
        up(mutex);
    }
}
```

→ till when is the reader inside the mutex

```
Void writer(void){
    while(true){
        down(db);
        D.B
        up(db);
    }
}
```

- 4 condition need to follow for synchronization

1) R → R ✓
2) R → W ✗
3) W → R ✗
4) W → W ✗

- rc is an integer variable which represent reader's count that is no. of reader present in the database at any point of time.

- mutex is Binary semaphore used by the reader in the mutual exclusive manner.

- db is a Binary semaphore variable used by reader & writer in mutual exclusive manner

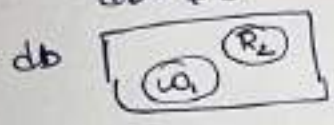
Ques what happen if we interchange

(a) down (mutex) & rc=rc+1 in reader's code

rc=0, 1, 2

mutex = 1, 0

db = 1, 0



∴ Both Reader & writer will enter the DB in same time

(b) what will happen when interchange if condition & up(mutex) in Reader's codes.

∴ Both Reader & writer will enter the DB in same time

(c) what happen when we interchange down(mutex) & rc=rc-1 in reader's code.

when R1 is execute & then R2 then deadlock is occuring.

Option

(i) no problem the soln still work correct

(ii) multiple readers are not allowed in DB

(iii) Both Reader & writer will enter the DB in same time

(iv) it is possible for Deadlock

rc = 0, 1, 2

mutex = 1, 0, 1

db = 1

rc = 0, 1

mutex = 1, 0

db = 1, 0

Pg no: 199
Ques: 19, 20

19]
```
void Reader (void){
  L1: down(mutex);
    if (w==1) {
      up(mutex); —①
      goto L1;
    }
    else {
      R=R+1;
      up(mutex); —②
    }
    down(mutex);
    R=R-1;
    up(mutex);
}
```

```
void writer(void){
  L2: down(mutex);
    if(R>=1 or w==1){ —③
      up(mutex);
      goto L2;
    }
    w=1;
    up(mutex);
    D.B
    down(mutex);
    w=0;
    up(mutex);
}
```

20] R=0 1
w=0 1
mutex=1 0 1 0

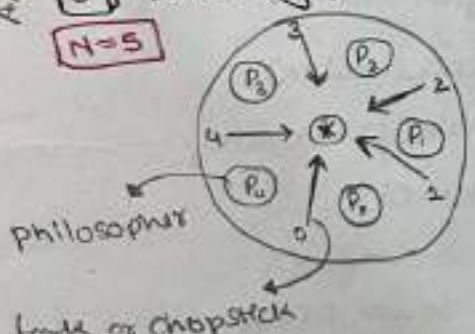
∴ multiple write can come at any point also ✱ reader

DB (w2) (w1) (R1)

Analog

3] Dining philosophers

N=5



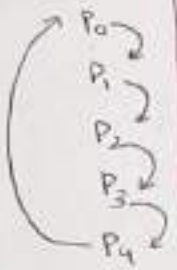

```
void philosopher(int i){ → philosophers no.
  while (True){
    thinking();
    take_fork(i); // take left fork
    take_fork((i+1)%N); // take right fork

    eats;
    put_fork(i); // put left fork back
    put_fork((i+1)%N); // put right fork back

  }
}
```

Note

- If all the philosophers are hunger at the same time then everybody will take there left fork 1st & when they try to take/attempt to take right fork then nobody will get the right fork & all the philosopher will wait on each other & they will go into Deadlock




```
#define N 5
#define THINKING 0
#define HUNGRY 1
# define EATING 2
# define LEFT (i+N-1) % N        // Left philosopher
# define RIGHT (i+1) % N         // Right philosopher
Semaphore mutex = 1;
Semaphore S[N];
// all S[i]'s are initialized
     to 0
Int State[N];
// an array to keep track of
every one state
```

```
void philosopher(int i)          // philosopher no.
{
  while(TRUE)
  {
    Thinking();
    take_fork(i);
    eat();
    put_fork(i);
  }
}
```

```
take_fork(int i)
{
  down(mutex);
  State[i] = HUNGRY;
  test(i);
  up(mutex);
  down(S[i]);
}
```

```
Put_fork(int i){
    down(mutex);
    State[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    up(mutex);
}
```

```
void test(int i){
    if (state[i]==HUNGRY &&
        State[LEFT]!=EATING &&
        State[RIGHT]!=EATING){
        State[i] = Eating;
        up(S[i]);
    }
}
```

- mutex is a Binary Semaphore used by philosopher in the mutual exclusive manner.
- S[N] is an array of Binary semaphore & initial all are assigned to "0".
- State[N] is an integer array used to keep track of every philosopher state. Initial all the philosophers are in thinking state.
- Analysis:

mutex = 1 ∅ 0

P_0 = T
P_1 = T
→ P_2 = T
P_3 = T
P_4 = T

S[0] = 0
S[1] = 0
S[2] = ∅ ≠ 0
S[3] = 0
S[4] = 0

Qus) assume that philosopher P_1 & P_3 are in Eating state. then philosopher P_2 is also trying to go into Eating state then which statement is above code is control not to go into eating state

a) if condition
b) test function
c) down(mutex);
d) down(S[i]);

Qus] what happen if we interchange up(mutex) & down(s[i]) in take_fork funcn.

Analysis :

mutex = 1, 0

P_0 = T	S[0] = 0
P_1 = E	S[1] = 0
P_2 = Y, H	S[2] = 0
P_3 = T	S[3] = 0
P_4 = T	S[4] = 0

} all are getting suspended, so it is getting in deadlock

a) no problem soln still work correct

b) more than 2 philosophers can go into eating state at once

c) Deadlock

d) none of the above

Qus] Assume P_1 is in Eating state & P_2 is also trying to go into eating state then it will be suspend. then what is the procedure & who P_2 will wakeup & go into eating state

Analysis: mutex: 1

- After P_1 go from eating state to next instruction

P_0 = T	S[0] = 0
P_1 = E	S[1] = 0
P_2 = Y H	S[2] = 0
P_3 = T	S[3] = 0
P_4 = T	S[4] = 0

S.L | P_2

will get into S L bcz down(s[i]) & wait for its left fork to be free

Qus) Each process P_i ; i!=0 to 4 & binary semaphore mutex m_0 to m_4 all are initialized to '1'.

Each process P_i executes the following code

```
down(m[i]);
down(m[(i+1) mod 4]);
C.S
up(m[i]);
up(m[(i+1) mod 4]);
```

Consider the following statement

1. M.E is satisfied
2. M.E is not satisfied
3. It is possible for deadlock

which of the above are True?

(a) only 2
(b) only 1
(c) only 1, 3
(d) only 2, 3

$m_0 = 1$
$m_1 = 0$
$m_2 = 0$
$m_3 = 0$
$m_4 = 0$

$P_0 = m_0 m_1$
$P_1 = m_1 m_2$
$P_2 = m_2 m_3$
$P_3 = m_3 m_0$
$P_4 = m_4 m_1$

Qus) Consider the following code used by the processes to access the common critical section

```
Shared boolean lock = false;
boolean key;
do{
  Key = True;
  while(Key == true)
    Swap(lock, Key);
  C.S
  lock = false;
}while(True);
```

note: swap function atomically swaps two [redacted] value by using call by reference

Consider the below statement

1. ME is satisfied
2. ME is not satisfied
3. It is possible for deadlock.

which of the above are true

(a) only 1
(b) only 1, 3
(c) only 3
(d) only 2, 3

Anaylsis

lock = ~~false~~ True

P_1	P_2	P_3
Key = ~~True~~ false	Key = True	Key = True

```
while(TRUE){
    Procedure_consumer.remove_item;
}
}
```

Concurrent programming

S_1: a=b+c;
S_2: d=e*f;
S_3: g=a/d;
S_4: h=g*i;

Read set = {b, c, e, f, a, d, g, i};
write set = {a, d, g, h};

Precedance graph:

Note

1. any 2 statement 'Si' & 'Sj' can be executed concurrently or parallelly if they follow below condition
   (i) $R(S_i) \cap w(S_j) = \phi$
   (ii) $w(S_i) \cap R(S_j) = \phi$
   (iii) $w(S_i) \cap w(S_j) = \phi$

Note

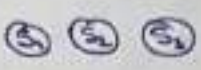
2. the real concurrent program is possible only on multiprocessor System
3. Concurrent have 3 different mean's
   - they can execute concurrently or parallely
   - they don't have any dependencies
   - any one can start 1st
4. Concurrent program will be written by using
   parallel → par begin - par end or co begin - co end ← Concurrent

Concurrently

```
begin
  S1;
  S2;
  S3;
end
```

Parallelly

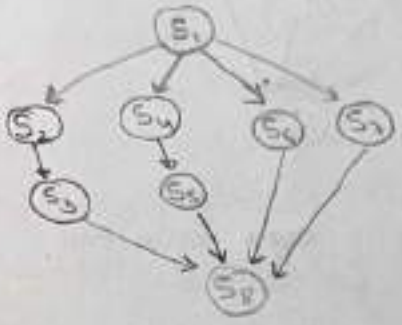
```
par begin
  S1;
  S2;
  S3;
par end
```



Question

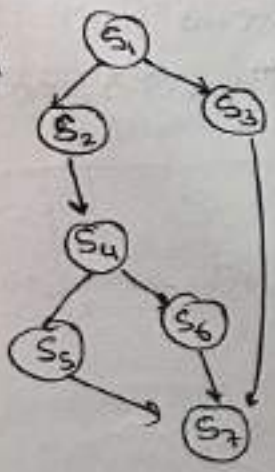
1

```
Begin
  S1;
  par begin
    begin
      S2
      S3
    end
    begin
      S4;
      S5;
    end
    S6;
    S7;
  par end
    S8;
  end
```



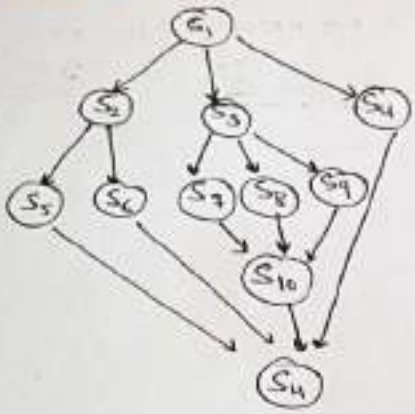
if nothing is given in starting or ending then by default assume begin-end

2



```
Begin
  S1
  par begin
    begin
      S2
      S4
      par begin
        S5
        S6
      par end
    end
    S3
  par end
    S7;
end
```

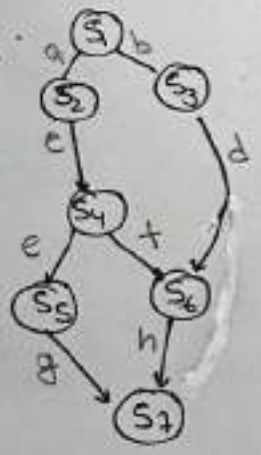
3




```
Begin
  S1
  Para Begin
    Begin
      S2;
      Para begin
        S5;
        S6;
      Para end
    end
    Begin
      S3
      Para begin
        S7;
        S8;
        S9;
      Para end
        S10;
    end
      S4;
  Para end
      S11;
end
```

- It is not possible to write the concurrent program for all the precedance graph using par begin & par end but it is very much possible to write with the help of Semaphore operation

4

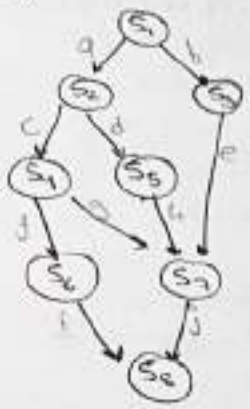


All the binary semaphore variable a, b, c, d, e, f, g, h are initialized to '0'

```
Para begin
    begin S1 , para begin V(a), V(b), para end, end;
    begin P(a), S2, V(c), end;
    begin P(b), S3, V(d), end;
    begin P(c), S4, para begin V(e), V(f), para end, end;
    begin P(e), S5, V(g), end;
    begin P(d), P(f), S6, V(h), end;
    begin P(g), P(h), S7, end;
Para end;
```

a = 0 1 0
b = 0 1 0
c = 0 1 0
d = 0 1 0
e = 0 1 0
f = 0 1 0
g = 0 1 0
h = 0 1 0

5 Marks



a = P(x) V(a)
b = P(x) V(b)
c = P(a) V(c)
d = P(a) V(d)
e = P(b) V(e)
f = P(c) V(f)
g = P(d) V(g)
h = P(e) V(h)
i = P(f) V(i)
j = P(g) V(j)

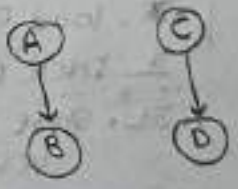
Parbegin & parend
b/c both a, b can be execute in any order

Par begin
begin S1, Parbegin, V(a), V(b), parend end;
begin P(a), S2 parbegin, V(c), V(d) parend, end;
begin P(b), S3, V(e) end;
begin P(c) S4 parbegin V(f), V(g) parend end;
begin P(d) S5 V(h) end;
begin P(f) S6 V(i) end;
begin P(g), P(h), P(e) S7 V(j) end
begin P(i), P(j) S8 end
parend.

6 Par begin
begin A; B; end
begin C; D; end
parend

Which of the following is a valid sequence of execution.

a) ABCD
b) CDBA
c) ACDB
d) CBDA
e) CADB
f) CABD
g) ACBD
h) DABC



7 Consider the following concurrent program

int x=0, y=0
Par begin
begin x=1; y=y+x; end
begin y=2; x=x+3; end
parend

x=0, y=0

x=1 → y=y+x

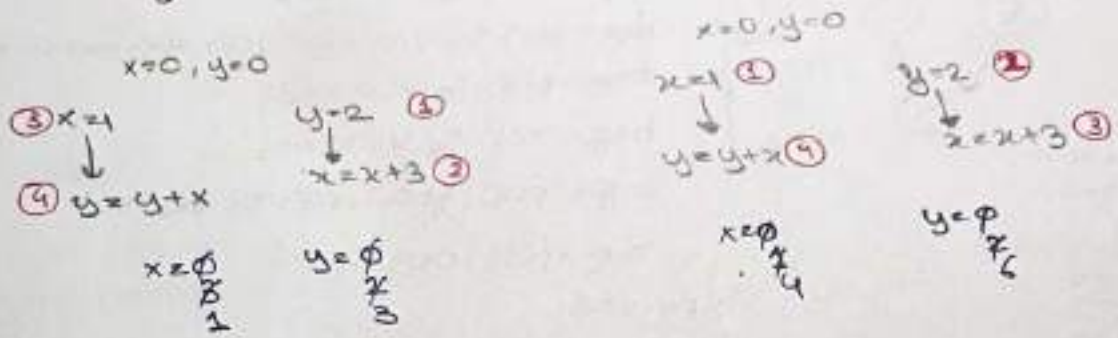
y=2 → x=x+3

What can be the final value of 'x' & 'y' after completion of the above Concurrent program?

① x=1, y=2 ② x=1, y=3 ③ x=3, y=6

Which of the above claims are possible?

a) 1,2 b) 2,3 c) 1,3 d) All 1,2,3



(Gate - 2013)

Pg: 180

Q: 25

a] -2

b] -1

c] +1

d] +2

W	X	Y
P(S); I. load Rw, M[x] II. INCR Rw III. Store M[x], Rw V(S); +1	P(S); I. load Rx, M[x] II. INCR Rx III. Store M[x], Rx +2	P(S); I. load Ry, M[x] II. DECR Ry (DECR-2) III Store M[x], Ry V(S); -2

Z
P(S) I. load Rz, M[x] II. DECR Rz III. Store M[x], Rz V(S); -4

Analysis:

S=2 ; x=0 -2 -4 +2

W → I
W → II

Rw {0, 1}

y → I, II, III

z → I, II, III

W → III

x → I, II, III

{-4, 2}

Rx

Ques] Consider the following Concurrent program

```
int count=0
void tally(){
    int i;
    for (i=1;i<=5;i++)
     count = count+1;
}
void main(){
    par begin
      tally();
      tally();
    par end
}
```

• what can be the min, final value of count after completion of both functions of tally()?

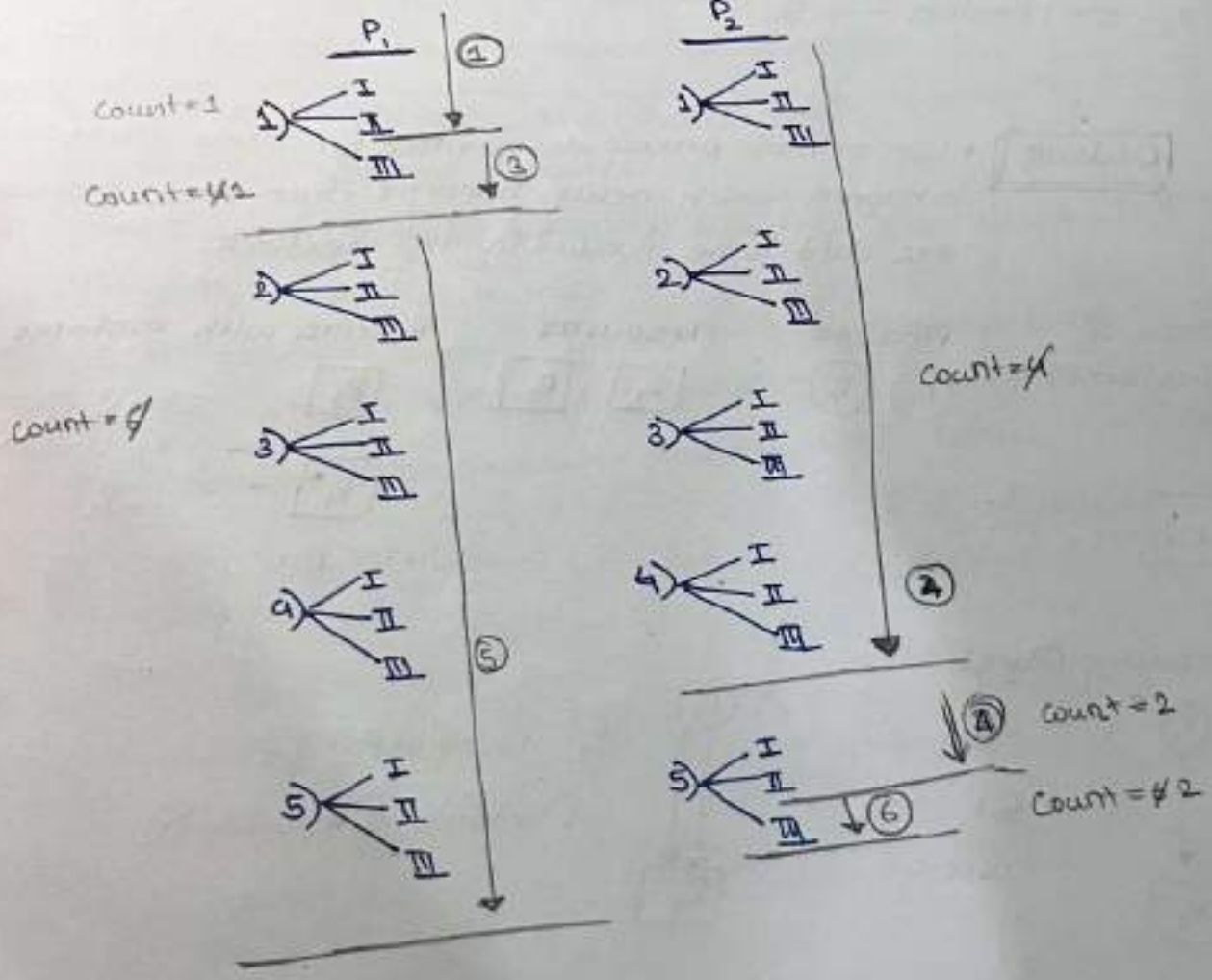
Note: Count = count + 1 , will execute on '3' instruction like load, increment & then store

I. load R1, M[count]

II. INCR R1

III. store M[count], R1

a) 1 b) 2 c) 3 d) 4 e) 5



Analysis:

count = 4

P_1: 1st iteration → I — 1 Row ☒ 1

P_1: 1st iteration → II

P_2: 1,2,3,4. iterations

P_1: 3rd iteration → III

P_2: 5th iteration → I — 2 Row ☒ 2

P_2: 5th iteration → II

P_1: 2,3,4,5 iterations

P_2: 5th iteration → III

Deadlock • two or more process are waiting for some event to happen which never happen's ithen those processes are said to be involved in the deadlocks.

• **Basic of Deadlocks:-**

Processes: P_1 P_2

Resources: R_1 R_2

Resource with instance: R_1 R_1 ⇒ single instance of resource 'R_1'

R_2: '2' Instance of Resource 'R_2'

Requesting Edges:

P_1 → R_1 : 'P_1' is Requesting for '1' instance of Resource 'R_1'

P_2 → R_2 : 'P_2' is requesting for '2' instance of Resource 'R_2'

Allocation Edge :

'1' instance of Resource 'R_1' is allocated to process P_1

'2' instance of Resource 'R_2' is allocated to process P_2

Note

the Resource Request & Resource allocation will be Represented by using Resource allocation graph.

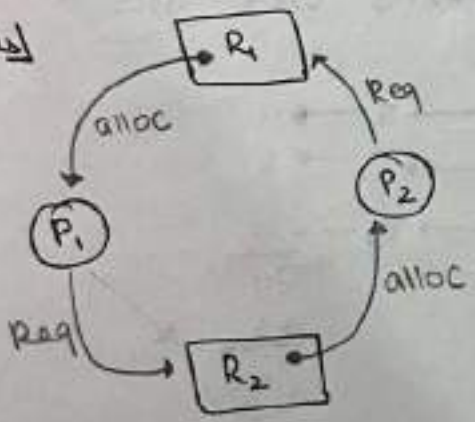
RAG = G(V, E) → Requesting & Allocation

V → Processes, Resource

- Resource Request/ Release life cycle

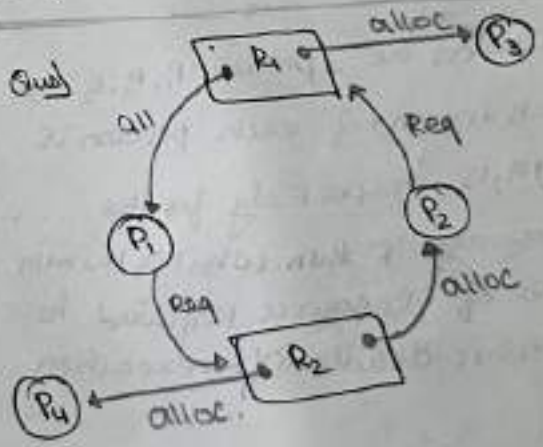
1. the process will request for the resource
2. O.S clearly validates the request of the process
3. If the request made by the process is valid then O.S will check for availibility of the Resource.
4. If the Resource is freely available then it will be allocated to process otherwise process has to wait.
5. If all the resource required for the start of execution are allocated then the process will go into Execution
6. Once execution of the process is completed then it Release all the Resource

Qus)



// Deadlock exist in the Resource Allocation graph(RAG)

Qus)



// No Deadlock exist in the RAG.

(Resource)

Qus] consider a system which have N processes, & 6 tape drives each process required 2 tape drive to complete there execution. then what is the max value of 'N' which ensure deadlock free execution

a) 2 b) 3 c) 4 d) 5

max 5 process

P_1	P_2	P_3	P_4	P_5
1	1	1	1	1
1	–	–	–	–

P_1	P_2	P_3	P_4
1	1	1	1
1	1	–	–

P_1	P_2	P_3
1	1	1
1	1	1

Qus] consider a system which have 3 processes & each process required 2 Resource to complete there execution then what is the min no. of Resource required to ensure Deadlock free execution

a) 2
b) 3
c) 4
d) 5

P_1	P_2	P_3
R_1	R_2	R_3
R_4		

= 4

Qus] Consider a system has N process and 6 resources each process required 3 Resource to complete there execution then what is max value of N which ensure Deadlock free execution

P_1	P_2
R_1	R_4
R_2	R_5
R_3	R_6

a) 2
b) 3
c) 4
d) 5

Qus] Consider the 3 process P_1, P_2, P_3 the peak demand of each process is 5, 9, 13 respectively for the Resource 'R' then what is the min no. of Resource Required to ensure deadlock free execution

a) 22 b) 23 c) 24 d) 25

→ $P_1 \longrightarrow 5 \longrightarrow 4$

$P_2 \longrightarrow 9 \longrightarrow 8$

$P_3 \longrightarrow 13 \longrightarrow 12$

24

+1

25

$5\left[\ P_1 \quad 9\left[\ P_2 \quad 13\left[\ P_3\right.\right.\right.$

23
Qus]
Page no 187
WB

$P_1 \rightarrow 4 \rightarrow 3$
$P_2 \rightarrow 3 \rightarrow 2$
$P_3 \rightarrow 7 \rightarrow 6$
$P_4 \rightarrow ?$

$20 - 11 = 9$

$P_4 = 9$

---

[1] necessary condition:
The deadlock may be possible or may not be possible

[2] Sufficient condition:
The deadlock never be possible
- The least upper bound which satisfies the condition is called as sufficient condition

Qus] P.g: 186
Q: If
'n' → processes
'm' → Resource,
$P_i \rightarrow S_i$

(a) $\forall P_i, S_i < m$

$m = 10$

$P_1 = 9$
$P_2 = 9$
$P_3 = 9$
⋮
$P_{100} = 9$

$9 < 10$

(b) $\forall P_i, S_i < n$

(c) $P_1 \rightarrow 5 \rightarrow 4$
$P_2 \rightarrow 9 \rightarrow 8$
$P_3 \rightarrow 13 \rightarrow 12$

$24 + 1 = 25$

$\sum_{i=1}^{n} S_i - n + 1 = m$

$\sum_{i=1}^{n} S_i = m + n - 1$

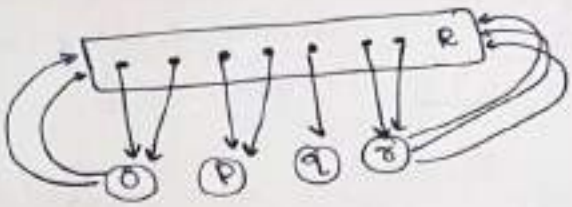
$\sum_{i=1}^{n} S_i < m + n$

- Sufficient condition:
in simple term:
if we provide certain amount of resource to a particular process then we can say that it is not going for deadlok

- necessary condition
in simple term:
the no of request mode by process for certain Resource, then we try to prove that is deadlock free or not.

Pg: 185
Q: 10

n=4(O, P, Q, R)



$x_o = 2$ $y_o = 2$
$x_p = 2$ $y_p = 0$
$x_q = 1$ $y_q = 0$
$x_r = 2$ $y_r = 3$

$y_p = y_q = 0$

(b) $x_p + x_q \geq \max y_k$, $k \neq q, p$ → it is a neccessary condition

↳ available Resource

ex: $2 + 1 \geq 3$

$3 \geq 3$

concept of deadlocks

1. Deadlock characterstics
2. ** Deadlock prevention
3. Deadlock avoidance
4. Deadlock detection
5. Deadlock Recovery

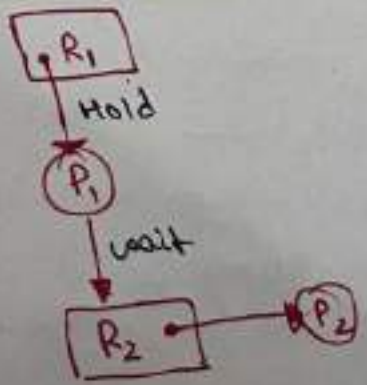
1 Deadlock characterstics (neccessary condition)

(i) Mutual Exclusion.

- the Resource has to be allocated to only one process or it is freely available
- There should be a one-one relation blw resource & the process.

(ii) Hold & wait

the process is holding resources & waiting for some other resource simultaneously

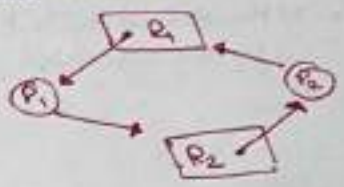


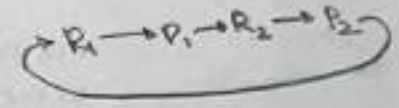
(iii) no preemption:

- the resources has to be "voluntarily" released by the process after completion of execution.
- It is not allowed to forcefully preempt the resource from the process.

(iv) Circular waits:

- the process are circular wait on each other for the resources.





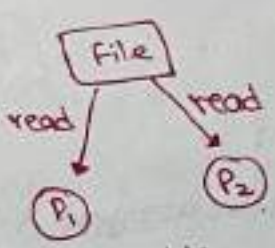
Note

- If all the above 4 condition are existing simultaneously in the system then it is possible for deadlock.

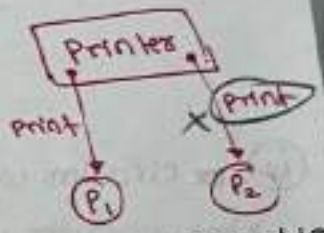
(2) Deadlock prevention:

(i) mutual exclusion:

- It is not possible to dissatify mutual exclusion always b/c of shareable & no-shareable resource.



Shareable Resource

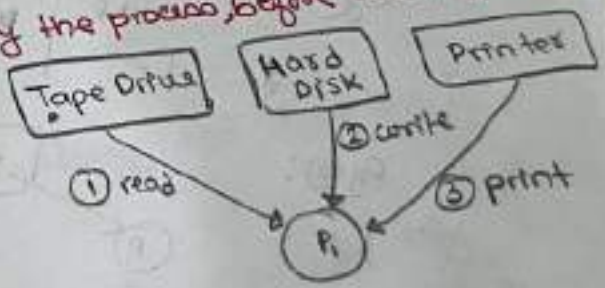


non-shareable resource.

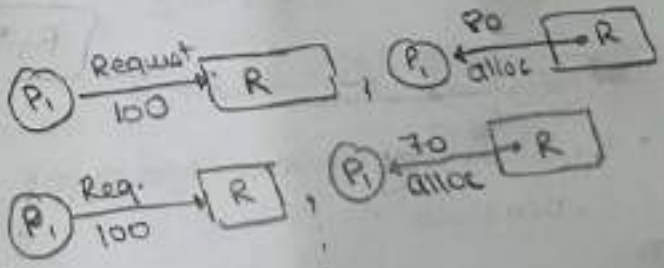
(ii) Hold & wait condition:

I] Allocate all the Resource required by the process before start of execution

Problem: Low device utilization or less resource utilization



II] The process should release all the existing resources before making the new request.



It is possible to go into Starvation

Starvation

(iii) No preemption

- the process 'P1' is requesting for Resource 'R1'
  - If the Resource 'R1' is free then it will be allocated to process 'P1'
  - the Resource 'R1' is not free & it is allocate to some other process 'P2'
    - If the process 'P2' is in the execution then process 'P1' has to wait
    - The process 'P2' is NOT in the execution it is waiting on some other Resource

⇓

Preempt the resource R_1 from process P_2 & allocate to process P_1

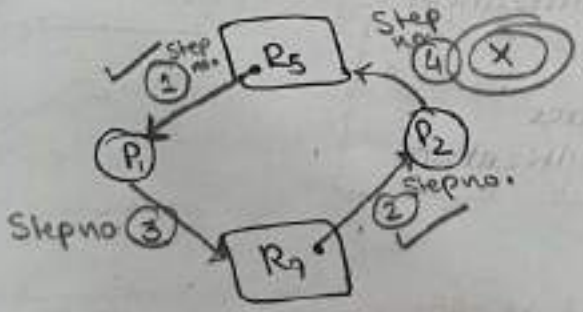
(iv) Circular wait:

- the resource will be assigned with the unique numerical no.
- the process can request for the resource only in the increasing (or decreasing) order of enumeration (numbering)

Increasing order

Eg (1): $P_1 \longrightarrow R_5$ ✓ | $P_1 \longrightarrow R_3$ ✗ | $P_1 \longrightarrow R_7$ ✓

Eg (2):



b/c P_2 is not allowed R_7 & then going for R_5

Note

- By dissatisfying any one of the above 4 condition we can prevent Deadlock

3 Deadlock avoidance

- Deadlock avoidance is implemented by using Bankers Algorithm.

Total available

A	B	C
↓	↓	↓
10	5	7

Resources: A, B, C

Processes: P_0, P_1, P_2, P_3, P_4

	max need			current allocation			current available			remaining need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0 →	7	5	3	0	1	0	3	3	2	7	4	3 → P_0
P_1 →	3	2	2	2	0	0	5	3	2	1	2	2 → P_1
P_2 →	9	0	2	3	0	2	7	4	3	6	0	0 → P_2
P_3 →	2	2	2	2	1	1	7	5	3	0	1	1 → P_3
P_4 →	4	3	3	0	0	2	10	5	5	4	3	1 → P_4
				7	2	5	10	5	7			

Total allocation

Safe Sequence: P_1, P_3, P_0, P_2, P_4

Remaining need = max need − Current allocation

- Current available = Total available − Total Current allocation

- If we can satisfy the remain need of all processes with the current available resource then the system is said to be in safe state otherwise system is said to be in unsafe state
- If the system is in unsafe state then it is possible for Deadlock
- the order in which we satisfy the remaining need of all the process is called as safe sequence
- safe sequence may not be unique, there may be multiple safe sequence.
- the unsafe state purely depends on behaviour of the processes
- the deadlock avoidance is less restrictive than the Deadlock prevention.

- each and ever time when the process is requesting for any resource then bankers algo will be implemented to identify whether the system is in the safe state or in the unsafe state.
- If the system is in the safe state then the request of the process will be granted & if the system is in the unsafe state then the request of the process will be denied & the deadlock will be avoided

W.B Out 2 Page no. 184

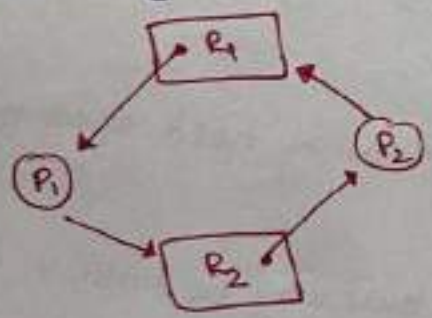
	max need	current Allocation	current available	Remaining need
	A B C	A B C	A B C	A B C
P_0 →	6 5 4	0 3 4	4 3 1	6 2 0
P_1 →	3 4 2	2 1 2	6 4 3	1 3 0
P_2 →	1 0 4	0 0 2		1 0 2
P_3 →	3 2 5	1 2 1		2 0 4

P: $P_1 P_0 P_2 P_3$ & $P_1 P_2 P_3 P_0$

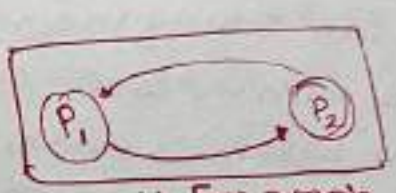
[4] Deadlock detection :-

(1) the Resource are of single instance type :

Run : cycle detection algo



$R_1 \rightarrow P_1 \rightarrow R_2 \rightarrow P_2$



wait For graph

- if all the resource's are of single instance type then the cycle in the RAG is necessary & sufficient condition for occuring of deadlock
- if all the resources are not of single instance type then cycle in RAG is just a necessary condition but not sufficient condition for occuring of deadlock.

(ii) the resource are of multiple instance type:-

	current allocation			current available			Remaining need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0 → P_0
P_1	2	0	0	0	1	0	2	0	2 → P_1
P_2	3	0	3	1	3	1	0	0	0 → P_2
P_3	2	1	1	5	1	3	1	0	0 → P_3
P_4	0	0	2	7	2	4	0	0	2 → P_4
				7	2	6			

P_0, P_2, P_3, P_5, P_4

- what happens if process P_2 is requesting for additional resource (0,0,1)

P_2
000 (old)
+ 001 (new)
001 (current)

	current allocation			Current available			Remaining need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0 → P_0
P_1	2	0	0	0	1	0	2	0	2 → P_1
P_2	3	0	3				0	0	1 → P_2
P_3	2	1	1				1	0	0 → P_3
P_4	0	0	2				0	0	2 → P_4

P_0 ⇒ unsafe ⇒ $P_1P_2P_3P_4$ deadlock

5] Deadlock Recovery

1] Killing the process :-

(a) Kill all the processes which are involved in the deadlock

(b) Kill one after the other

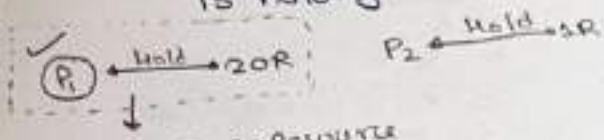
[i] Low priority

[ii] % of process completion

P_1 → 95%.

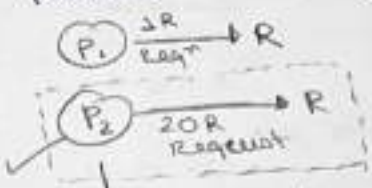
P_2 → 5%. → Kill these as it was just arrived & P_1 is about to complete so kill P_2

(iii) # of Resource the process is holding



By Kill P_1 20 Resource will be free rather than Killing P_2

(iv) # of Resource the process is requesting



will be Killed bc we don't know when the 20 Resource will be free

2] Resource Preemption

- the Resource will be preempted from the processes which are involved in the Deadlock & that preempted resource will allocated to some other processes, so that there is a chance of recovering the system from deadlock
- if above condition is followed then there is a possibility for process to go into Starvation

3] ostrich algorithm

→ Ignore the deadlock

note

- all deadlock state is always unsafe but all unsafe may or may not lead to the deadlock

Thread • light weight process

In these situation we may not coant to have different process

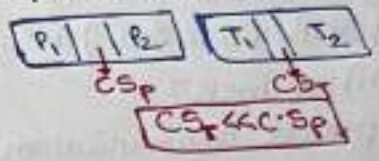
If we have different process then
- context switching
- different add. space

- Thread are about concurrency & parallelism.
  - ↳ Concurrent execution on single core system
  - ↳ parallelism on a multi-core system

- For the similiar task if we have different processes (instead of thread) then
  - IPC (inter process communcn) is required
  - process context switching is bit costlier than thread context switching.

- A process can have multiple thread or single thread.

- Context will be less
- no. of instn will be less
- faster context switch (CS)



Context
- might have opened some file
- some register value PC, R_1, R_2

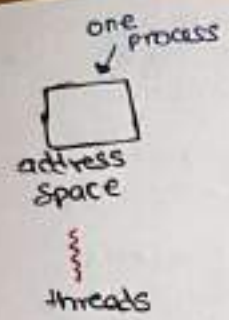
ex. what are the things that threads should not same?
- → Register value
- → Stack

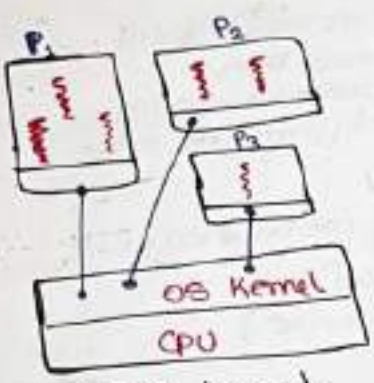
Should be shared
- ↳ code
- ↳ data
- ↳ files
- ↳ heap

- Type of Threads

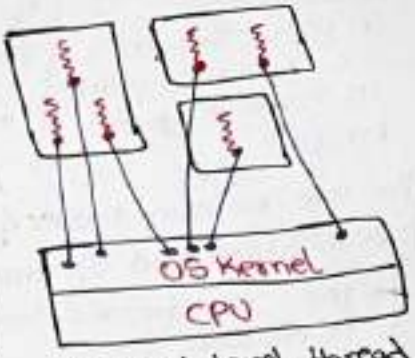
(1) User level thread
- when we make threads using funcn call
- we have TCB (Thread control block)

(2) Kernel level
- when we make thread using System call
- for each thread we have new PCB (TCB)
- since only kernel can create PCB hence System call is needed





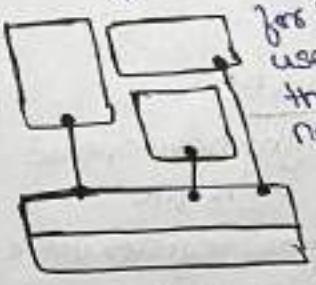
user level thread.



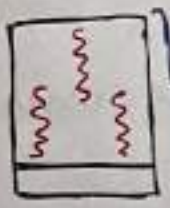
Kernel level thread
(Kernel is aware of each thread)

ex. why do we don't want different processes for similar task?
- Interprocess communication
- Context switch.

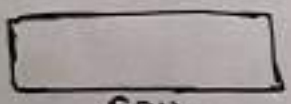
Kernel will see them in



for Kernel user level thread are not visible



thread in user space



CPU.

↳ Can we take help from Kernel Scheduler to Schedule the threads?

NO

- I (user) might be having my own preferences to schedules

↳ I can't take help from O.S Scheduler, So how I am Supposed to Schedule them? } I need to write my own Scheduler

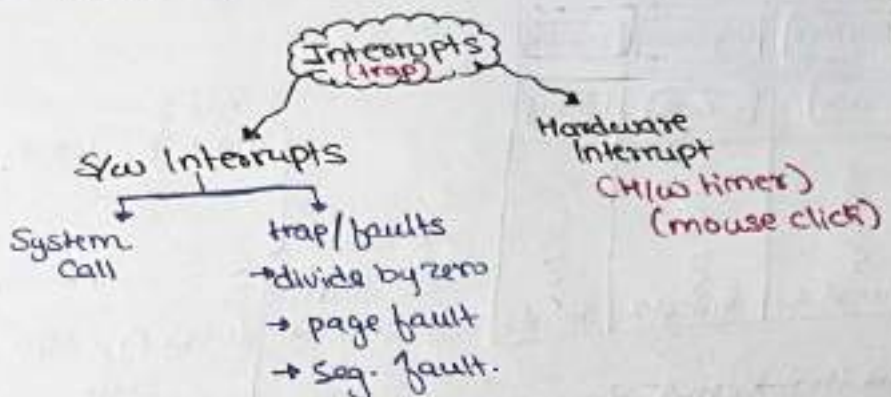
we have thread library (which has everything

there are libraries to help us.

↳ Scheduler

↳ thread manager.

- Can user level thread run Kernel code?
  yes, we can have system call in user thread.



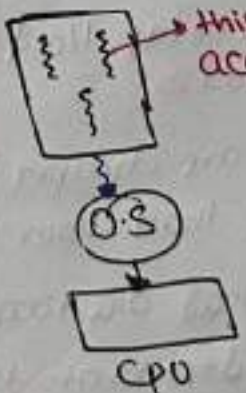
- P_1 is running on CPU
- now P_1 want to interact I/O device
- we are moving out P_1 from running to some other side)
- we will be having system call
- In that system call there will be instruction (exe. scheduler) that will schedule the O.S schedule
- os scheduler will schedule some other process.

- user level thread

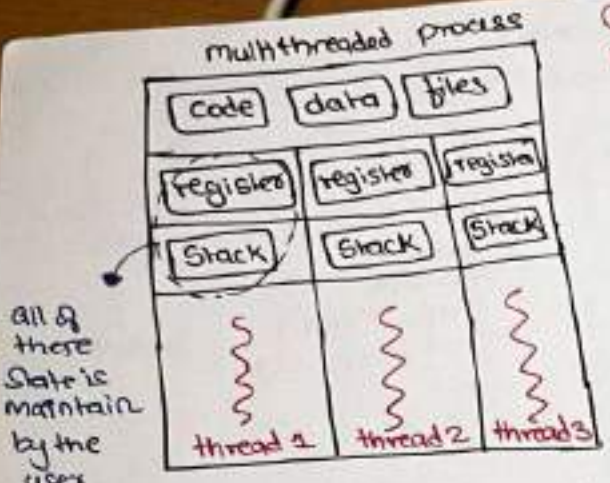
advantage

1. it is fast to create (blc no system call needed)
2. Kernel doesn't have to maintain different state of different threads (user code will do it) hence context switching is not needed
   - using libraries
3. we have our own scheduler here.
4. easy & simple to implment
5. required less context & no H/w support required

disadvantage

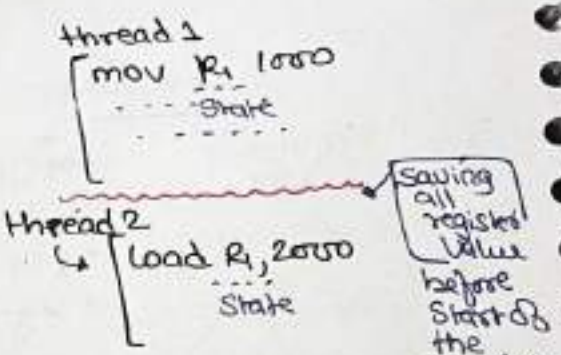
1. not visiable to Kernel (hence O.S can't make decision about schedule based on available hardware (cpu's)).
2. 

   os will put the complete process in the block state.
   (taking poor level of scheduling decisions).



All of there State is maintain by the user using the library, so that Kernel doesn't have to maintain the state.

② point explanation & advantage



- Kernel threads.
- O.S now manages thread & processes (address space)

↳ all thread are implemented in the Kernel

advantage

- if one kernel level thread is performing blocking system call [I/O operatn] then another thread will continue the execution

↳ possible to overlap I/O & computation inside a process.

- Kernel threads are cheaper than process (But costlier than user threads.)
- Kernel level thread are recognize by O.S & work as independent thread.
- Support to K.L.T (require H/w support) for

- Decreasing order of the cost

① create a process using fork-exec
② create kernel level thread
③ create use level thread.

1 > 2 > 3

disadvantage

- complicate to implement (difficult)
- have more context
- user level thread is faster than b/w kernel thread since a context switch is not required.

ex: The OS provides the illusion that it has it's own address space

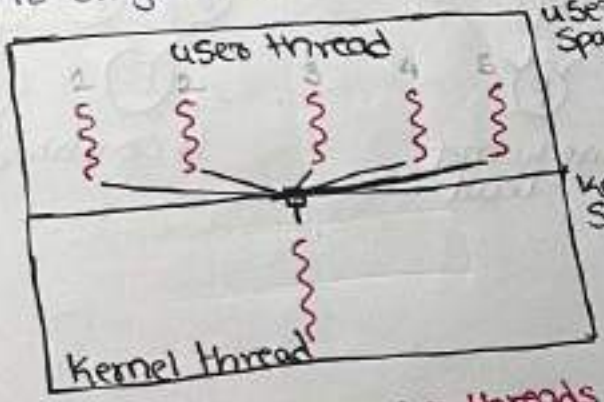
(no), O.S doesn't say every thread is having it's own heap/code

ex: with Kernel-level threads, multiple thread from the same process can be schedule on multiple CPU simultan^n.

(True)

ex: user level thread is blocked for I/O then Kernel of O.S will perform context switching to run another user level thread which is not block. (false) b/c Kernel doesn't have any information about user level thread.

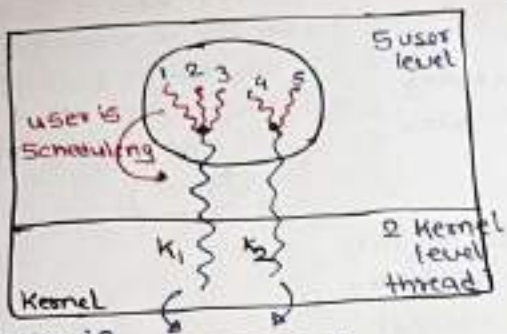
- Many-to-one
  - ↳ many user level thread mapped to single Kernel thread



↳ If we have only user threads then this <many to one> model is by default.

- one-to-one
  - ↳ Each thread map to Kernel thread

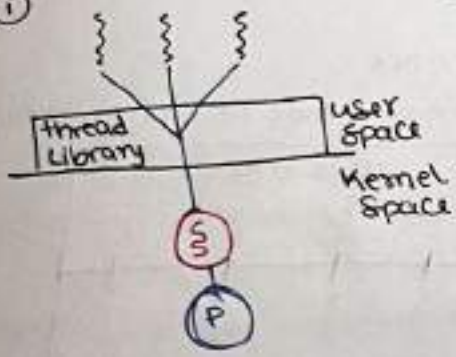
↳ If you have only Kernel level thread then it's a default case.



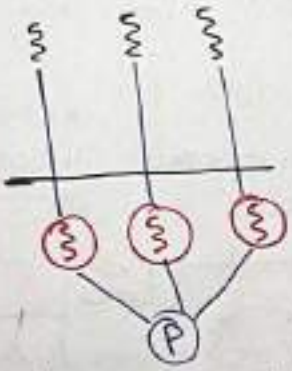
who is Scheduling is (0·5)

- if K_1 is running on cpu then one of the U_1, U_2, U_3 is running

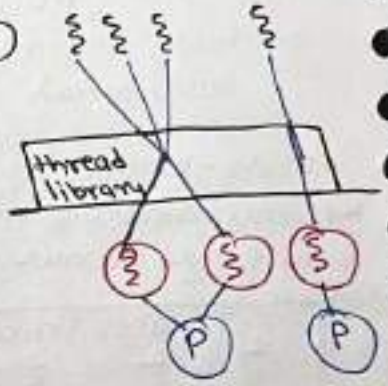
ex. how kernel will distinguish to b/w these 3 thread?
kernel doesn't even know

① 

↳ Pure user level

② 

↳ pure Kernel level

③ 

↳ combined

{ user level thread

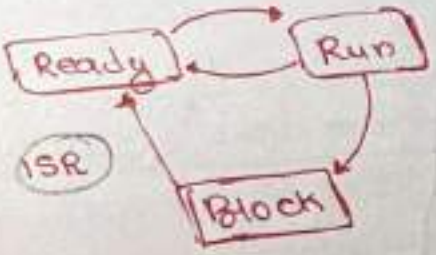
(3) Kernel level thread

(P) process.

- I/O operation: I/O process is categorized into two types

(i) Synchronous I/O:

- In synchronous I/O the process performing I/O operation will be placed in the block state till I/O operation is completed
- once I/O operation is completed an ISR will be initiated which bring the process from the block state to Ready state



(ii) Asynchronous I/O:

- In Asynchronous I/O call initiating the I/O request the handler function will be registered
- the process is not placed into block state & it continue to execute the remaining code after initiating the I/O request
- once the I/O operation is completed the signal mechanism is used to notify the process that data is available & register handler funct^n will be asynchronously invoked.

If large size of I/O is executing then all the other process which doesn't required I/O can be run parallelly

---

Handler funct^n

↳ Details related to I/O operation.

Memory Management

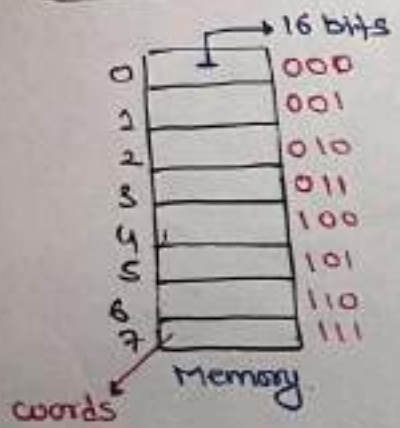
- functionality:- allocating & de allocating the memory to the processes
- goal: Efficient utilization of memory by minimizing the internal & external fragmentation.

Main memory
Physical memory
Primary memory
RAM

memory → ① Byte Addressable ex: 32KB
memory → ② word Addressable Ex: 32 kw

1 Byte = 8 bits
words = ?
↓
it can be anything (it would be mention in question)

$2^2=4$	$2^6=64$	$2^{10}=1024 \approx 10^3 = 1K$	
$2^3=8$	$2^7=128$	$2^{20}=1M$	$2^{50}=1P$
$2^4=16$	$2^8=256$	$2^{30}=1G$	
$2^5=32$	$2^9=512$	$2^{40}=1T$	



Capacity of Memory = # of words in memory * word size

ex: 8 * 16 bit
⇒ 8 * 2B
⇒ 16 Byte

Qus) Consider a System which have no. of words of 512 G each word having 128 bits then what is the capacity of memory in Byte.

⇒ $2^9 * 2^{30} * 2^7$
⇒ 2^6 TBits
⇒ $\frac{2^6}{2^3}$ TByte
⇒ 2^3 TB
⇒ 8TB

Ques) consider a system which has 64 M words & each word has size of 48 bits then what is the capacity of memory in Bytes.

capacity = $2^6 * 2^{20} * 48$ bit
$= 2^6 * 2^{20} * 6$ Byte

Ques) consider a system where 32 Binary Bits are used to represented all the word of memory & each word has size of 32 bits then what is the capacity of the memory in Bytes

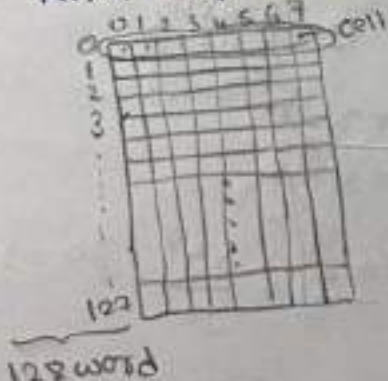
$2^{32} * 32$ bits
$2^{32} * 4B$
$2^{32} * 2^2 B$
$\Rightarrow$ 16 GB

RAM Chip Implementation

RAM chip size is 128B
Organize the main memory capacity of 16KB

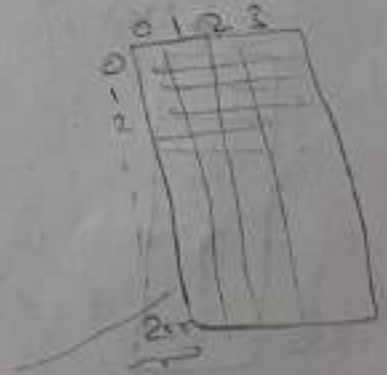
1. no. of RAM chip required?
2. Draw the memory organization map?
3. What is the size of decoder required?
4. Draw the memory organization map with the word size of 16 bits

RAM chip size is 128B

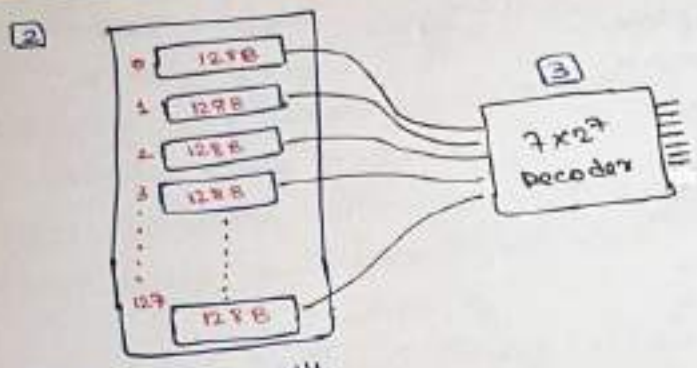


128 word

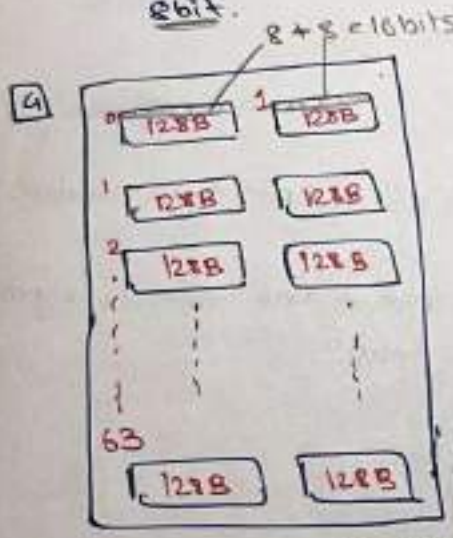
1. no. of RAM chip required
$\Rightarrow \frac{16KB}{128B} \Rightarrow \frac{2^4 * 2^{10}}{2^7} \Rightarrow$ 128



256 words



16 KB memory, with the word size of 8bit.



16KB memory, with the word size of 16 bits

$\frac{128}{2} = 64$ rows

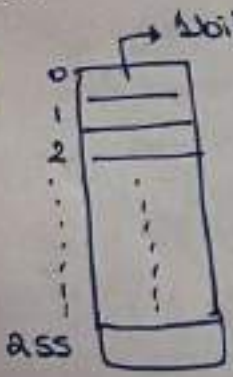
2 → Colums

64 x 2 = 128

these can't be modified

2 RAM chip size is 256 x 1bit organize the main memory capacity of 32MB.

(i)

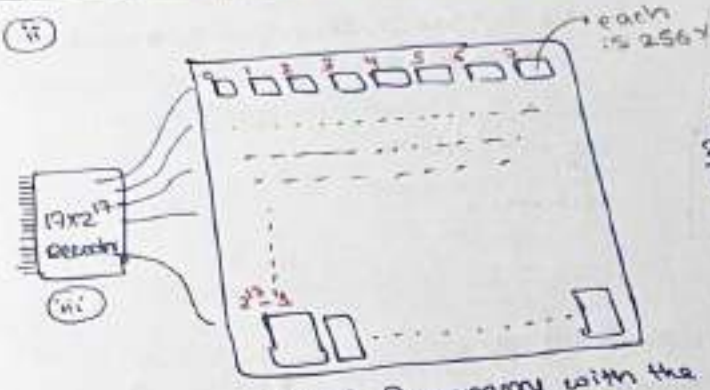


$\frac{32MB}{256 \times 1bit} = \frac{32 \times M * 8bits}{256 * 1 bits}$

$\Rightarrow \frac{2^5 * 2^{20} * 2^3}{2^8} = \frac{2^{28}}{2^8} = 2^{20} = 1M$

we can't write 1MB Counting Can't be done in Byte so its 1M

(ii)

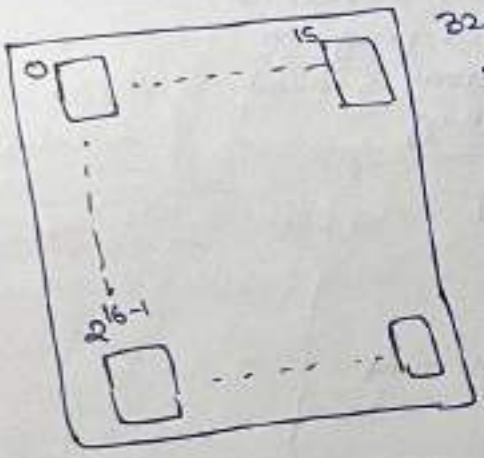


32 MB of memory with the word size of 8 bits

$\frac{2^{25}}{8} = \frac{2^{25}}{2^3} = 2^{17}$ rows

1 word = 1 Bit

(iv)



32 MB of memory with the word size of 16 bits

$\Rightarrow \frac{2^{25}}{16} = \frac{2^{25}}{2^4} = 2^{16}$ rows

columns.

(3) Ram chip size is 512 x 2 bit organize the main memory capacity of 16 MB → 8 bit ✓ kya nahi?

(i) $\frac{16\,MB}{512 \times 2\,bits} \Rightarrow \frac{16 \times 2^{20} \times 2^3}{2^9 \times 2}$

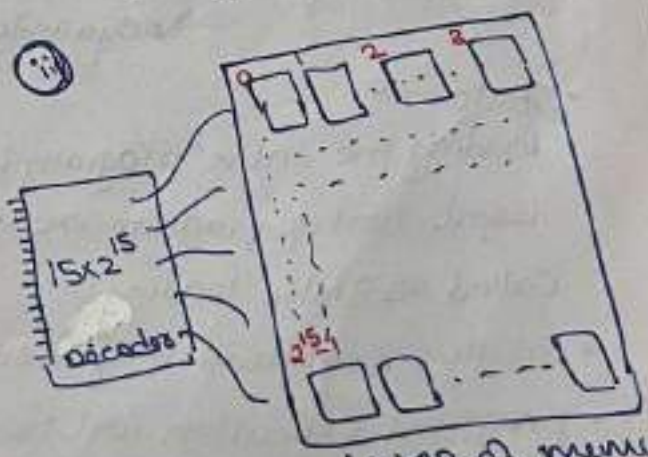
$\Rightarrow 2^7 * 2^{10}$

$\Rightarrow 128\,K$

$\Rightarrow 2^{17}$

1 word = 2 Bits

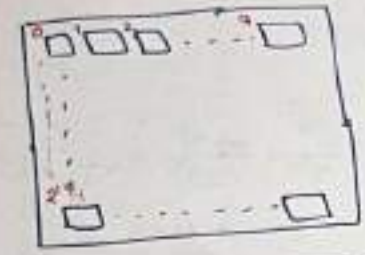
(ii)



16 MB of memory
4 word size of
4 bits.

$\frac{2^{17}}{2^2} = 2^{15}$ rows

(iv)



16MB of Main memory with word size of 8bit.

$\frac{2^{24}}{2^5}$ = 2^{19} rows

2^5 columns

- Loading, Linking & Address Binding :

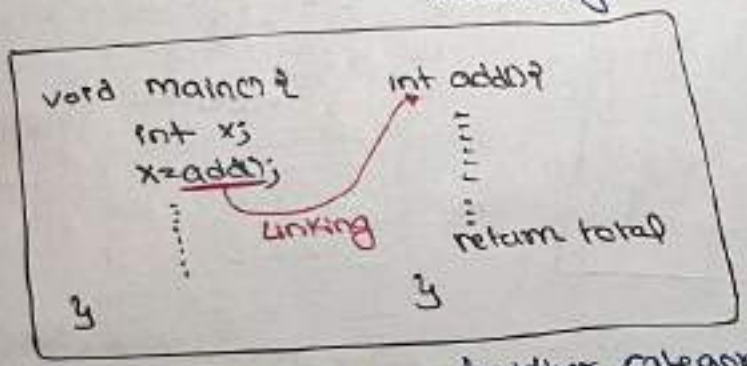
Address Binding

Loading

- Bring the program from 2nd memory to Main memory

Linking

- establishing the linking b/w all the module or all the functn of the program inorder to continue program execution is called as Linking



- Loading & Linking are further categorized into 2type → Static, Dynamic

Static

- loading the entire program into memory before start of program execution is called as Static loading
- Inefficient utilization of memory
- program execution will be faster
- if Static loading is used a/c Static linking will be applied.

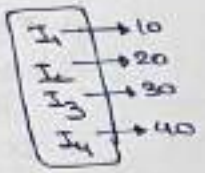
Dynamic

- loading the program into memory on demand is knowas Dynamic loading
- program execution will be slower
- memory utilization is efficient
- if dynamic loading is used A/c Dynamic linking will be applied.

• Address Binding (A·B)

Association of program instruction & data to the actual physical memory location called as address Binding (A·B)

Program = P_1



PC = 10 20 30 40

type of address Binding

1) Compile time A·B
2) Load Time A·B
3) Dynamic A·B or execution time A·B

• Compile time A·B

→ If the compiler is responsible of performing A·B then it is called as Compile time A·B

→ these type of address Binding is done before loading the program into memory

→ compiler required to interact with the O·S memory manager to perform compile time A·B

• Load Time AB

→ these time of A·B is done after loading the program into memory

→ these is done by O·S memory manager [loader].

• Dynamic A·B or execution Time A·B

→ the A·B will be posponded even after loading the program into memory

→ the program will keep on change the location in the memory till the time of program execution.

→ these type of address binding will be done by processor at the time of program execution

Note

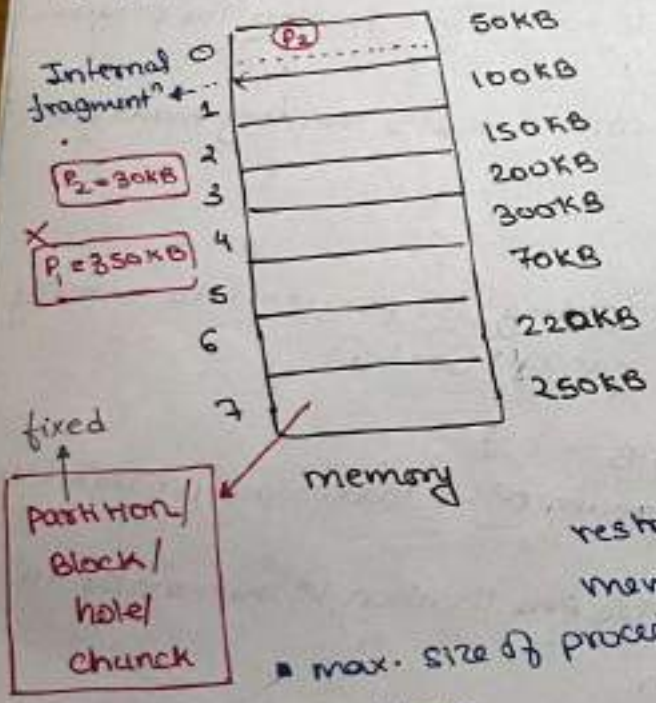
• majority of the O·S practical implement Dynamic loading, Dynamic linking & Dynamic A·B

ex: UNIX, window

Memory Management Technique (M.M.T)

- Contiguous
  - 1) fixed partition Scheme
  - 2) Variable partition Scheme
- non contiguous
  - 1) paging
  - 2) multilevel paging
  - 3) Inverted paging
  - 4) Segmentation
  - 5) segmented paging

- Fixed partition scheme :- (may also suffer from external fragmentation)

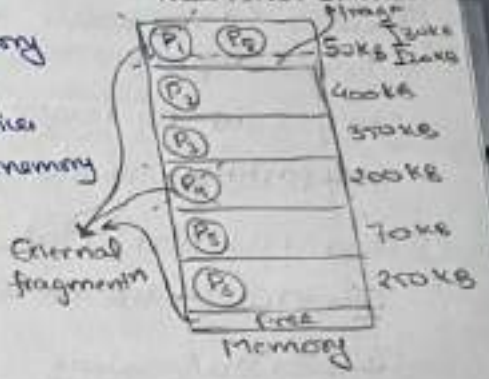


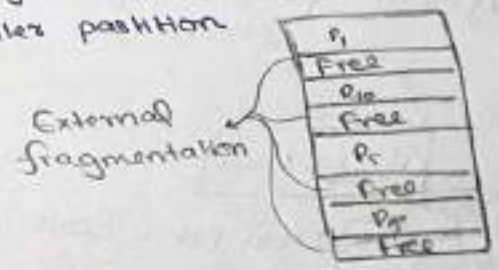
- In fixed partition scheme, memory is divided into fixed no. of partion
- fixed means no. of partition are fixed in the memory
- in every partition only one process can be accomdated
- Degree of multiprogramming is restricted by no. of partition in the memory
- max. size of process is restricted by max. size of partition

- every partition is associated with limit register

Limit Register
- ① lower limit:
  - Starting Address of the partition
- ② upper limit:
  - Ending Address of the partition

- variable partition scheme:
- In variable partition scheme, initial memory will single continous free block
- whenever the request by the process arrives Arc the partition will be made in the memory
- If the smaller process keep on coming then larger partition is divided into smaller partition





P_1=50KB, P_2=400KB
P_3=350KB, P_4=200KB
P_5=70KB, P_6=250KB

P_7=250KB ✗

P_8=30KB

- To avoid the problem of external fragmentn the following technique are used.

① Compaction:
- moving all the process to top or towards bottom to make the free available memory in a single continous space is called compaction
- compaction is undesireable to implement b/c it inter upts all the running processes in the memory

② Implementing non-contingnous M.M.T

Partition allocation Method

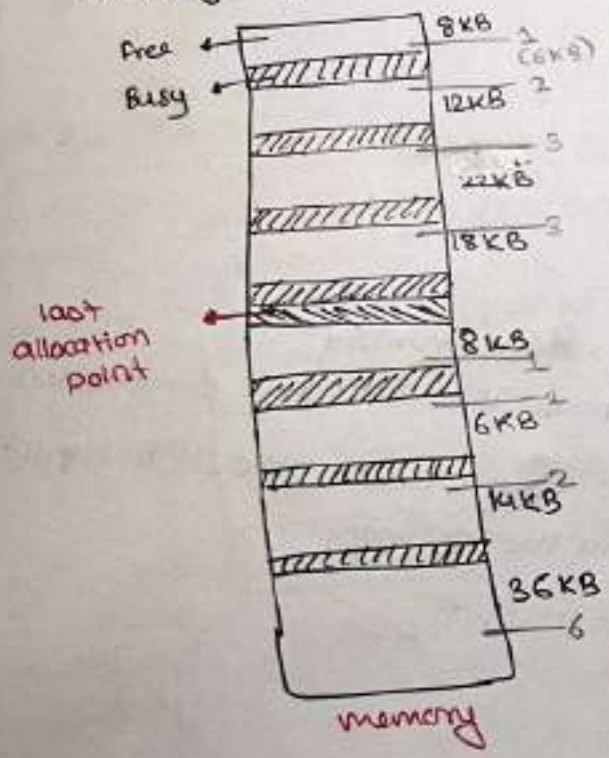
① First-fit: allocate the process in a partition which is first sufficient partition from top of memory

② Best-fit: Allocate the process in a partition which is smallest sufficient among the free available partitions.
- to find out the smallest sufficient it required to search all the free partition from the memory.

③ Worst-fit: Allocate the process in a partition which is largest sufficient among the free available partition.
- to find out the largest sufficient it requires to search all the free partition in the memory

④ Next fit: next-fit also work like 1st first but it will search for the 1st sufficient partition from last allocation point

Qus] Consider the following memory map.



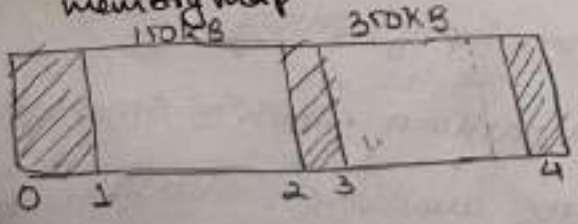
$P_1 \Rightarrow 16KB$

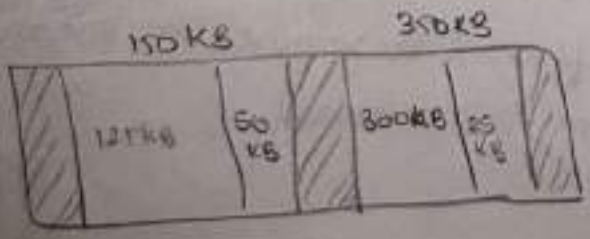
1] First Fit : 22KB { From the top of the stack}
2] Best Fit : 18KB
3] Worst Fit : 36KB
4] next Fit : 36KB. { From last allocation point}

Qus] How many successive request of 6KB will be satisfied by using first fit assuming variable partition scheme

19 ?

Qus] Consider the following memory map





the request from the processes are 300KB, 25KB; 125KB, 50KB respectively the above Request could be satisfied with?
(Assume variable partition scheme)?

a) Best fit but not first fit
b) First fit but not best fit
c) both
d) neither of both

• paging •

1] Logical Address Space (L·A·S) (can find by earth wires)
or Virtual Address space (V·A·S)
- Represented in the form of words or Byte
Eg:- 512KW or 512KB

2] Logical Address (L·A) or (generated by CPU)
Virtual Address (V·A) (Secondary memory)
- Represented in the form of bits
Eg: 32bit or 42bit

3] Physical Address Space (P·A·S) (main memory)
Represented in the form of words or Byte
Eg: 64MW or 64MB

4] Physical Address (P·A) (generated by main memory)
Represented in the form of bits
Eg: 26 bit

L·A > P·A

Ques] LA = 47bit
L·A·S = 2^{49} W
= $2^9 \times 2^{40}$ W
= 128 TWords

Q2] L·A·S = 32GW
LA = $2^5 \times 2^{30}$ W
L·A 35bit

Q3] P.A = 56 bits
P·A·S = 2^{56} B
= $2^6 \times 2^{50}$ B
= 64PB

Q4] P.A·S = 512MB
P.A = $2^9 \times 2^{20}$
⇒ 29 bits.

- technique of mapping, CPU generated logical address to physical address is called as paging

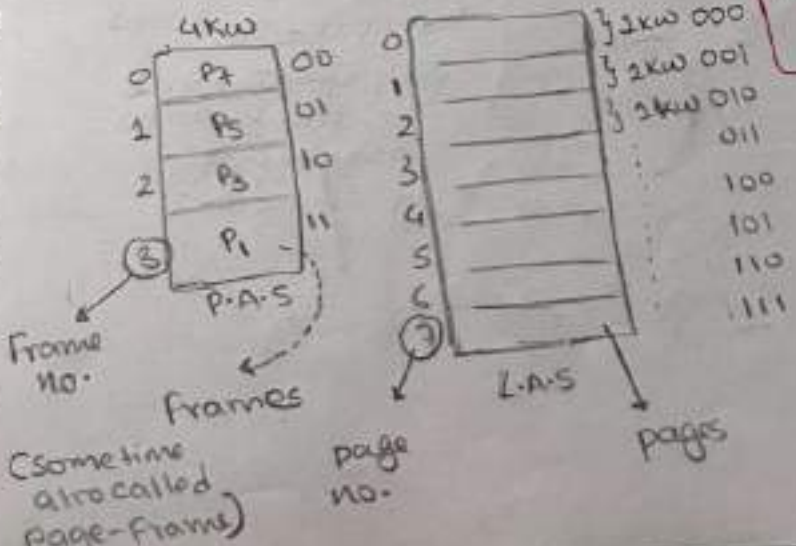
---

L·A = 13 bits
P·A = 12 bits
(Assuming word addressable)

L·A·S = 2^{13} = 8KW
P·A·S = 2^{12} = 4KW
page size = 2KW

- logical address space (L·A·S) will be divided into equal size pages.

$$\text{no. of pages} = \frac{\text{L·AS}}{\text{page size}} = \frac{8\text{KW}}{2\text{KW}} = 4$$



- P·A·S will be divided into equal size frame
- page size is always same as frame size

Page size = frame size

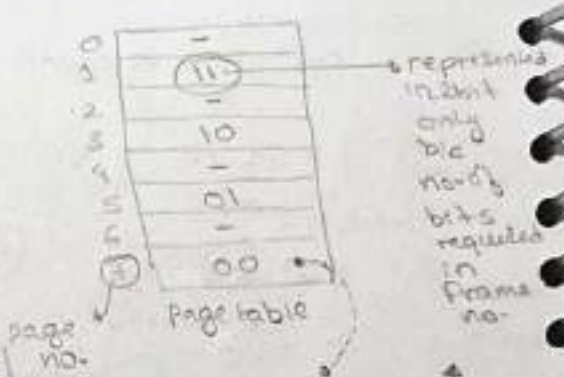
$$\text{no. of frame} = \frac{\text{P·A·S}}{\text{frame size}} = \frac{4\text{KW}}{2\text{KW}} = 4$$

- some of the pages of L·A·S will be brought into P·A·S

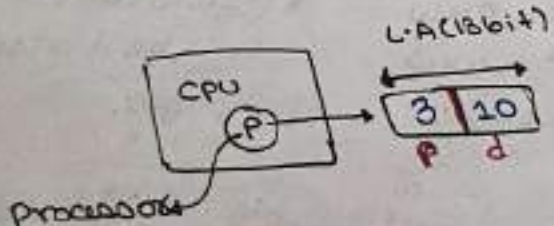
Page size = (1KW) → 2^{10}

Page size = 8KW or 8KB ↓ just representation

- whenever paging is applied the page table has to be maintained

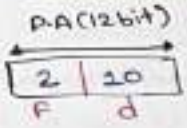


- no. of entires in pagetable = no. of pages in L·A·S
- page table entires definitely Contains frame no. bits.
- page table is also know as Address Translation table
- Frame no. bit also know as Translation bits



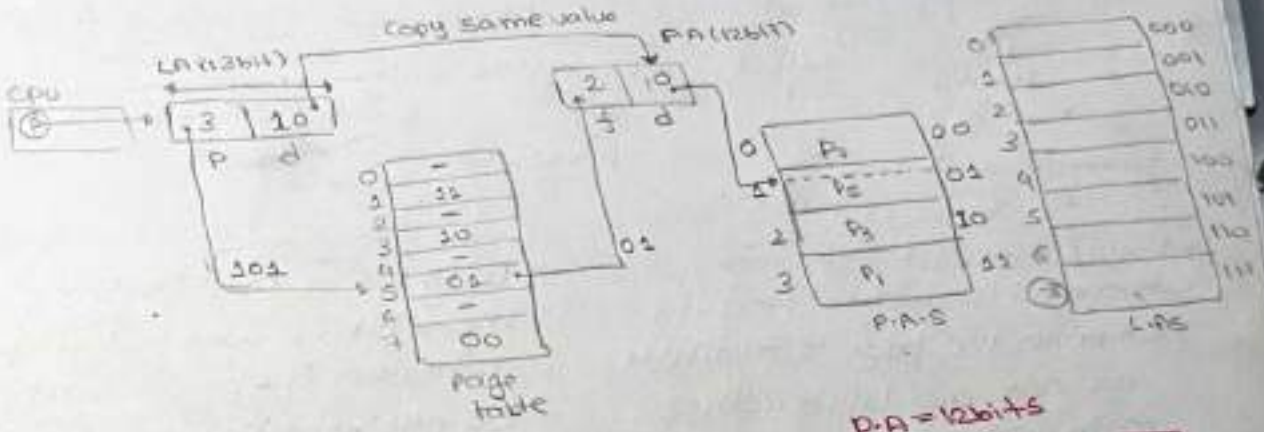
p: no. of bit required to represent page no. of L·A·S

d: no. of bits required to represent word no. of the page or also know as page offset

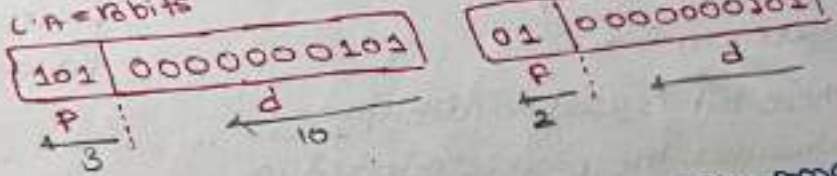


F: no. of bit required to represent frame no. of P.A.S

d: no. of bit required to represent word no. of the frame or frame offset



Eg:



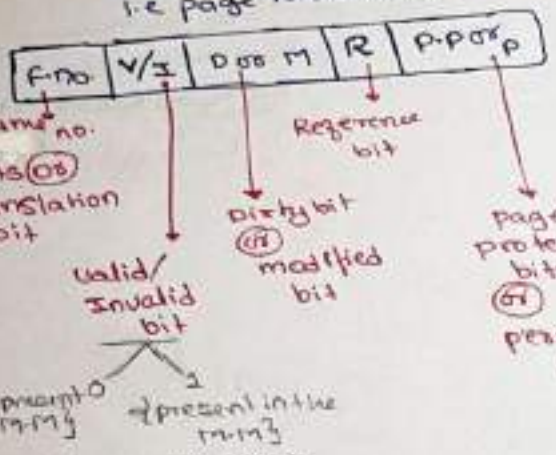
- Important point

1. Whenever the process is created paging will be applied on the process & page table will be created & the Base address of the page table will be stored in the P.S.B (process control block).
2. paging is w.r.t every process & every process will have its own page table.
3. page table of the process will be stored in the main memory
4. there is no external fragmentation in the paging
5. the internal fragmentation exist in the last page & internal fragmentation in the page is consider as $\frac{P}{2}$ (where 'P' is the page size)

half page

6. maintaining the page table is consider as overhead to the system
7. window practical follows page size of 4KB

i.e page table entire



Note

- page table entire definitely contain frame no bit but some time alongwith the frame no. it may also contain additional bit like as follow

- Valid/Invalid bit:
  these bit is used to identify wheather the page is available or not available in the main memory

- Referencen bit
  these bit is used to identify wheather the page is refered in the last clock cycle or NOT, how many time page is refreed.

{it doesn't have any fixed size}

- Dirty bit or modified bit
  these bit is used to identify wheather the page is modify or not modified by the process will accessing the page as part of execution.

{0 not modified} {1 modified}

- page protection bits or permission bits
  these bit is used for the protection & security from unauthorized access

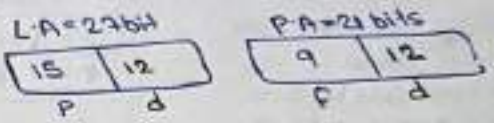
Ques) Consider a system which has a L·A = 27 bits.
P·A = 21 bits
Page size = 4KW
memory is word addressable
then calculate no. of page & no. of frame

L·A·S = 2^{27} bits = 128MW
P·A·S = 2^{24} bits = 2MW

* Page size = 4KW

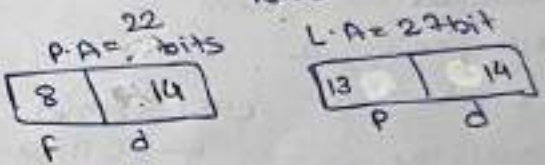
no. of pages = $\frac{128MW}{4KW}$ = 32K = 2^{15}

no. of frame = $\frac{2MW}{4KW}$ = 2^9



Ques) Consider a system which has no. of pages = 8K &
page size = 16KB
P·A = 22 bits
& memory is Byte addressable.
then calculate L·A & no. of frame?

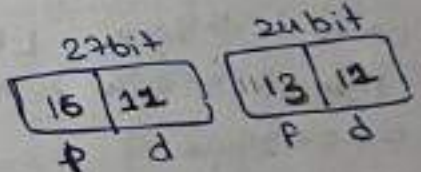
P·A·S = 2^{22} bits = 4MB
no. of frame = $\frac{4MB}{16KB}$ = 2^8



L·A·S = # page * page size
= 8K * 16KB
= 2^{13} * 2^{14}
= 2^{27}

Ques) Consider
L·A·S = 128MW
P·A = 24 bits
memory is word addressable
P·A·S is divided into 8K frame
then what is page size & how many pages in the L·A·S

L·A = 27bits
P·A = 24bits
P·A·S = 16MW
&
P·A·S = $\frac{16MW}{8K}$
= $\frac{2^{24}W}{2^{13}}$ = $2^{11}W$
= 2KW
=> 2KW



page size = frame size

His frame no.

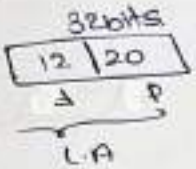
no. of pages = $\frac{LAS}{\text{page size}}$ = $\frac{128MW}{2KW}$ = 2^{16} = 64K

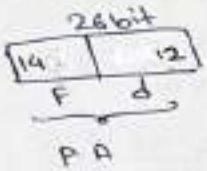
$\frac{2^{27}}{2^{11}}$ = 2^{16}

Q] Consider a system which have
L.A = 32bits
& P.A.S = 64MB
page size = 4KB
memory is
Byte addressable
then what is the
approximate size
of page table in
Byte.

L.A.S = 4GB
P.A = 26 bits
P.A.S = 64MB

no. of page = $\frac{L.AS}{\text{Page Size}} = \frac{4GB}{4KB} = 2^{20}$





page table size = $2^{20} * 14$ bits

$\rightarrow 2^{20} * \frac{14}{8} \rightarrow \approx 2B$

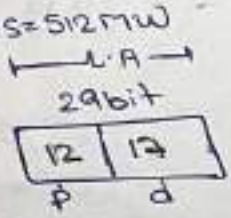
$\Rightarrow 2^{20} * 2B$

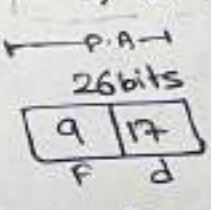
$\Rightarrow 2MB$

Que] consider the system
have (4K entries) &
L.A = 29 bits
then what is the
P.A, if system
has 512 frames
(memory is
word
addressable)

no of pages

LAS = 512MW





$\frac{512M}{4K}$

$2^7 K$
12 bits

no. of pages = $\frac{512M}{4K} = 2^{17}$

W.B
40
Page: 192

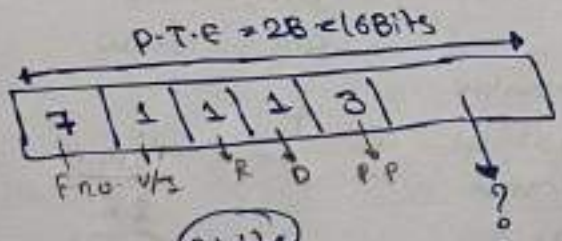
L.A = P.A = 2^{16} Bytes
page size = 512 Byte

L.A.S = 64KB
P.A.S = 64KB

no. of page = $\frac{64KB}{512B} \Rightarrow \frac{2^{16}}{2^9} = 2^7$

no. of frame

P.T.E = 2Byte
1 valid/invalid bit
1 dirty bit
3 bit for protection
1 bit for reference



16 - 13 = 3 bits

Imp

Ques] Consider a system which has L.A.S = 's' bytes & page size = 'p' byte.
page table entire size = 'e' byte
memory is (Byte addressable)
then what is the optimal value of page size by minimizing the memory overhead of maintaining the page table size & internal fragmentation in the paging.

memory overhead = P.T.S + Internal fragmentation in paging.

no. of page = $\frac{s}{p}$

$\Rightarrow \frac{s}{p} * e + \frac{p}{2}$

$\Rightarrow \frac{d}{dp}\left(\left(\frac{s}{p}\right) * e + \frac{p}{2}\right) = 0$

$\Rightarrow -\frac{s}{p^2} * e + \frac{1}{2} = 0$ $\Rightarrow p^2 = 2se$

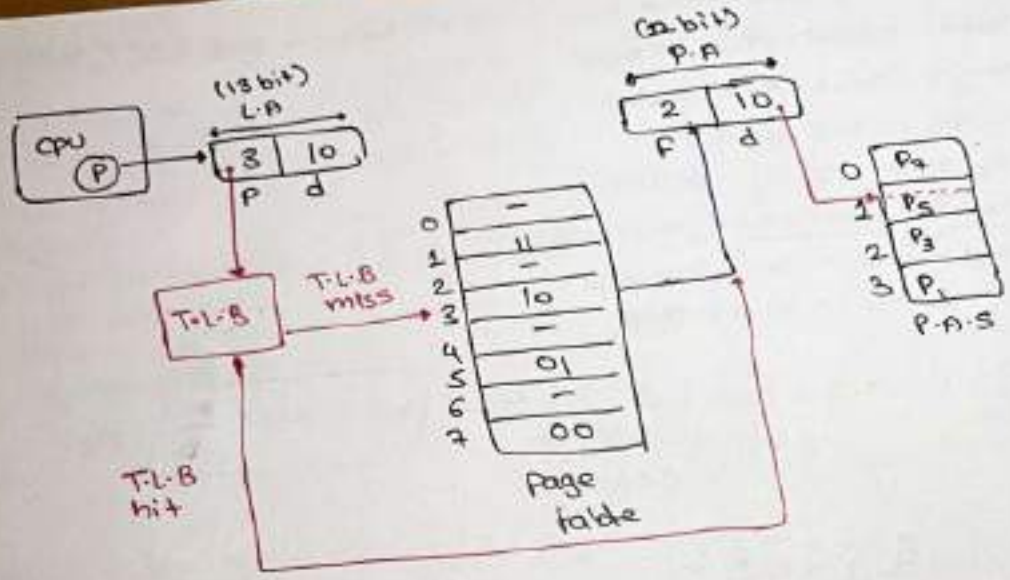
$p = \sqrt{2se}$

Performance of paging

1. main memory access time = 'm'
page table are store in the M.M.
then the formula for effective memory access time

E.M.A.T = 2m

2. the translation lookaside buffer (T.L.B) is added to improve the performance of paging.
3. TLB is a H/w device implemented By using associative Register
4. TLB access time will be very less compare to M.M access time
5. TLB will be placed before the page table
6. TLB contain recently referred page no. & corresponding Frame no.



- T·L·B access time = 'C'
  T·L·B hit ratio = 'x'
  then the formula for E·M·A·T with TLB

$$E\cdot M\cdot A\cdot T = x[C+m] + (1-x)[C+2m]$$

Qus] consider a system which has M·M access time = 100ns
& T·L·B access time = 20ns
& T·L·B Hit ratio = 95%
then what is E·M·A·T with TLB & without TLB.

with TLB = $\cdot 95[20+100] + \cdot 05[20+200]$

$\Rightarrow \cdot 95 \times 120 + \cdot 05 * 220$

$\Rightarrow 114 + 11$

$\Rightarrow$ **125ns**

without TLB:-

EMAT = 2×100

$\Rightarrow$ **200ns**

Qus] How much hit ratio required to reduce the E.M.A.T from 300ns without TLB to 250ns with T.L.B

T.L.B access time = 60ns

$2m = 300ns \quad m = 150ns$

$250 = x(60+m) + (1-x)(60+2m)$

$250 = 60x + 150x + 360 - 360x$

$150x = 110$

$x = \frac{110}{150} = \boxed{0.733}$

% of hit ratio $= \boxed{73.33}$

Qus] consider a system which have

L.A of 32 bits

L.A = 32 bits

Page size of L.A.S = 4KW

memory is word addressable

P.T.E size = 1W.

then what is page table size

$LAS = 2^{32}$ bits $= 4GW$

no. of page $= \frac{L.A.S}{\text{page size}} = \frac{4GW}{4KW} = 2^{20}$

P.T.S $= 2^{20} * 1W$

$\Rightarrow 1MW.$

• Multi-level paging.

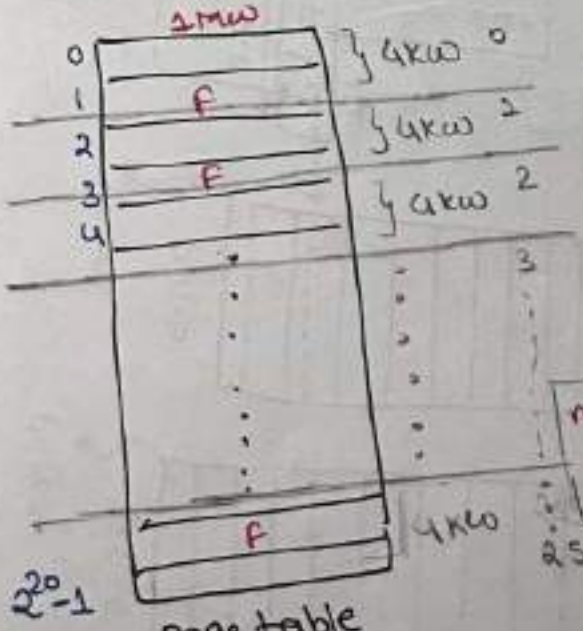
- to avoid the overhead of maintaining page table of large size for a process the concept of Multi-level paging is implemented
- In the multi-level paging the paging will be applied on the page table instead of bring the entire page table into memory the pages of page table will be brought into memory.

L.A = 32bit,

page size of L.A.S = 4KW

P.T.E size = 1W

$P.T.S = \frac{2^{32}}{2^{12}} * 1W = 2^{20}W = 1MW$

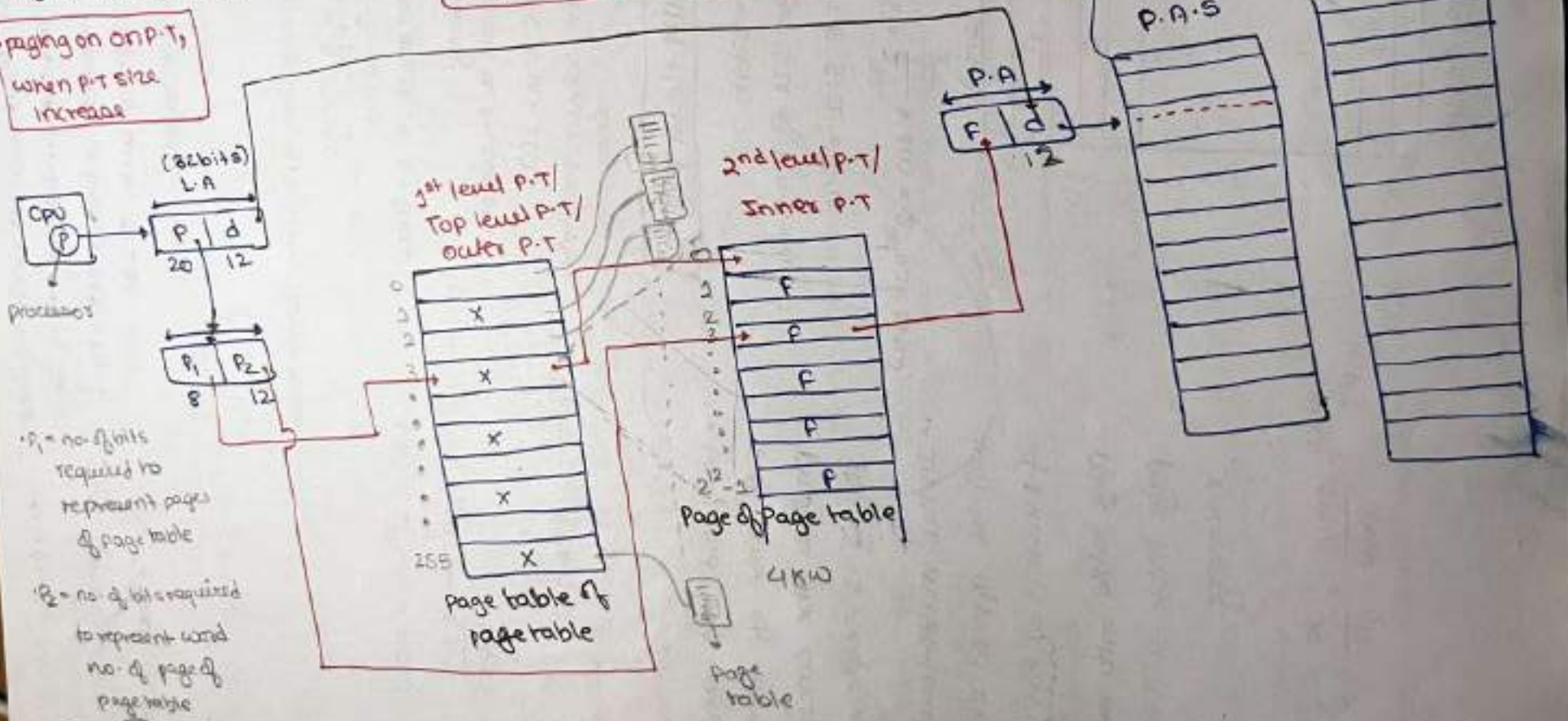


no. of pages in P.T $= \frac{1MW}{4KW} = \frac{2^{20}}{2^{12}} = 2^8 = 256$

Multi level paging :-

- paging on on P.T, when P.T size increase

X = B.A of page of P.T
page size of P.T = 4KW



- P_1 = no of bits required to represent page no of page table
- P_2 = no of bits required to represent word no of page of page table

(OR)

page offset of page table

(Ques) consider a system which have 2 level paging applicable.
page table has divided into 2K pages & each page is having 4K entries
the P.A.S = 64MW which is divided into 8K frame (words)
P.T.E size in both level is 2W. memory is word addressable

the calculate

① Length of L.A : 36 bit
② Length of P.A : 26 bit
③ 1st level page table size
④ 2nd level page table size (page of page table)

P.A.S = 64MW
P.A = 26 bits

no. of frame = P.A.S / frame size

no. of pages in page table = 2^{14} pages
each page = 2^{12} entries

$\Rightarrow \frac{2^{26}}{2^{13}} = 2^{13}$ no. of frame

P.A.S = 26 bit

13	13
F	d

L.A.S = 36 bits

23	13
P	d

③ 1st level page table = $2^{P_1} * P.T.E$ size

$\Rightarrow 2^{11} * 2W$

$2^{12} W = 4KW$

P_1	P_2
11	12

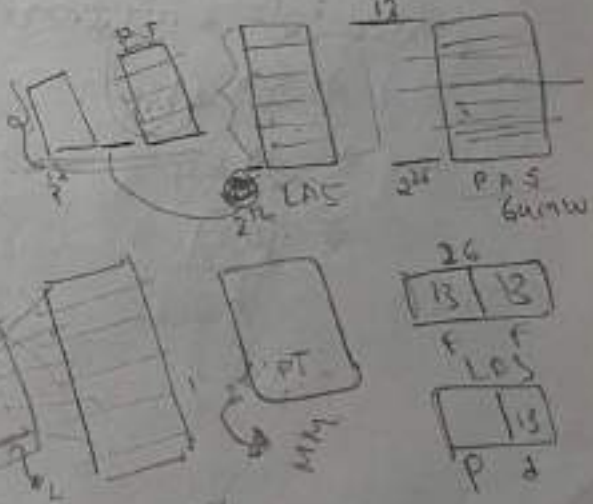
{ 2K pages, 4K entries }

④ 2nd level page table size = $2^{P_2} * P.T.E$ size

(page of P.T)

$\Rightarrow 2^{12} * 2W$

$\Rightarrow 2^{13} W = 8KW$



Q] Consider a sys. which has 2-level paging applicable P.T has divided into 8K pages & each page rohaving 16K entries memory is byte Addressable P.A.S is 256MB which is divided into 64KB frame. P.T.E.S in both level is 32bits then calculate length of *frame size

P.A.S / frame size

P.A.S = 256MB

P.A = 2^{28} = 28 bits

Frame size = P.AS / no. of frame = 256MB / 64KB = 4K

no of frame

$\Rightarrow 2^{12}$

① length of LA : 43bit

② length of P.A : 28bit

③ 1st level of P.T size : 32K

④ 2nd level of P.T size : 16K

(P.A) 28bits

12	16
f	d.

d: no of frame

L.A: 43bits

27	16
	d

13	14
P_1	P_2

③ 1st level index : 2^{P_1} * P.T.E Size

$\Rightarrow 2^{13}$ * 32bits.

$\frac{32}{8} = 4$

$\Rightarrow$

$\Rightarrow 2^{13} * 4$ Byte

$\Rightarrow 2^{15}$ B $\Rightarrow$ 32K

④ 2nd level index : 2^{P_2} * P.T.E Size

$\Rightarrow 2^{14} * 32$bit

$\Rightarrow 2^{14} * 4$

$\Rightarrow$ 16K / 64K

- performance of '2' level paging:

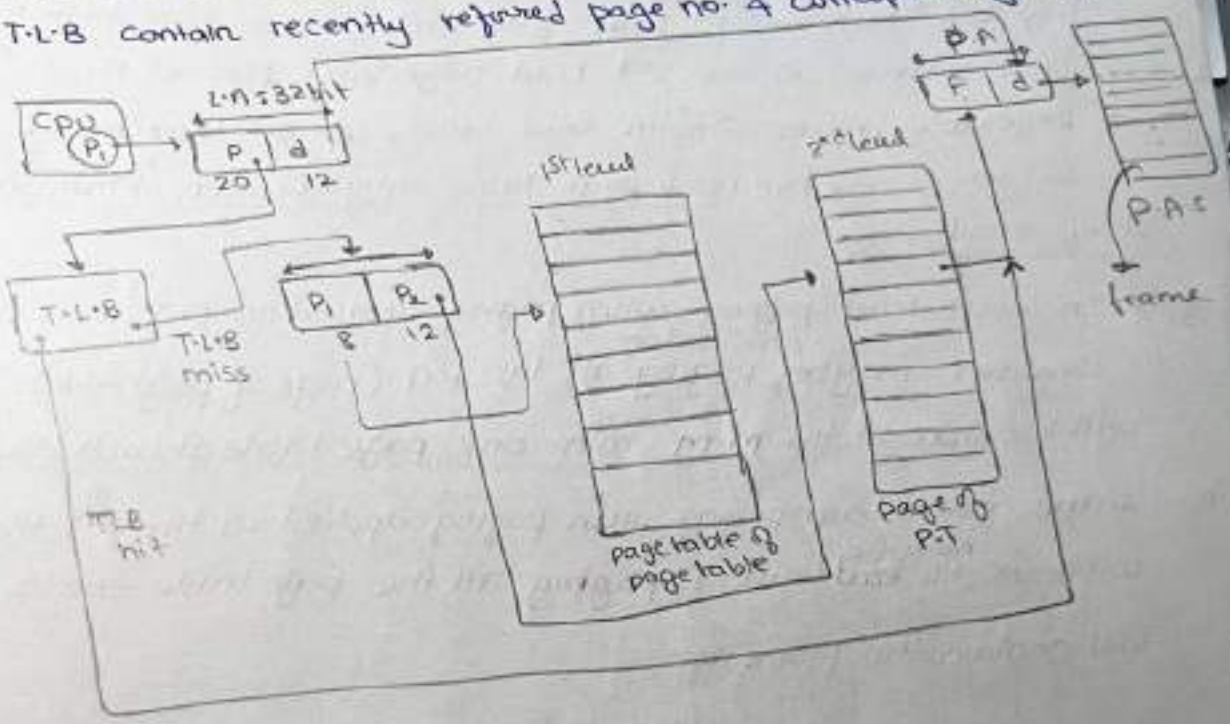
main memory access time = 'm'

page table are stored in the M.M

then Formula for E.M.A.T

E.M.A.T = 3m

- the T.L.B is added to improve the performance of paging
- T.L.B contain recently refered page no. & corresponding page no.



- TLB access time = 'c'
- TLB HA ratio = 'x'

then the formula.

for E.M.A.T,

E.M.A.T = x(c+m) + (1-x)(c+3m)

'n' level paging

E.M.A.T = x(c+m) + (1-x)(c+(n+1)m)

• Important point :

1. In the multi-level paging when the paging is applied on the page table then the final page table which we get is called as 1st level page table

2. In the multi-level paging when the paging is applied on the page table then the 1st level page table entire will contain, base address of the 2nd level page table & second level page table entires contain base address of 3rd level P.T & So on. ...., the final page table entry contain Frame no. of actual page

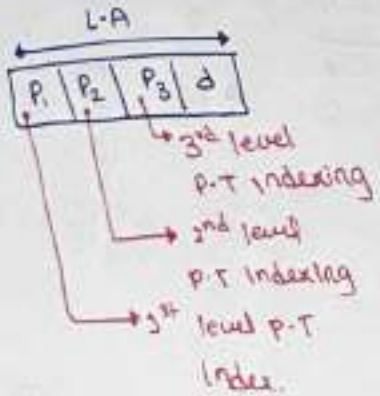
3. In multi-level paging when paging applied on P.T then whatever maybe the level of paging all the P.T (page of page table) will be store in the M.M (min. one page table at each level).

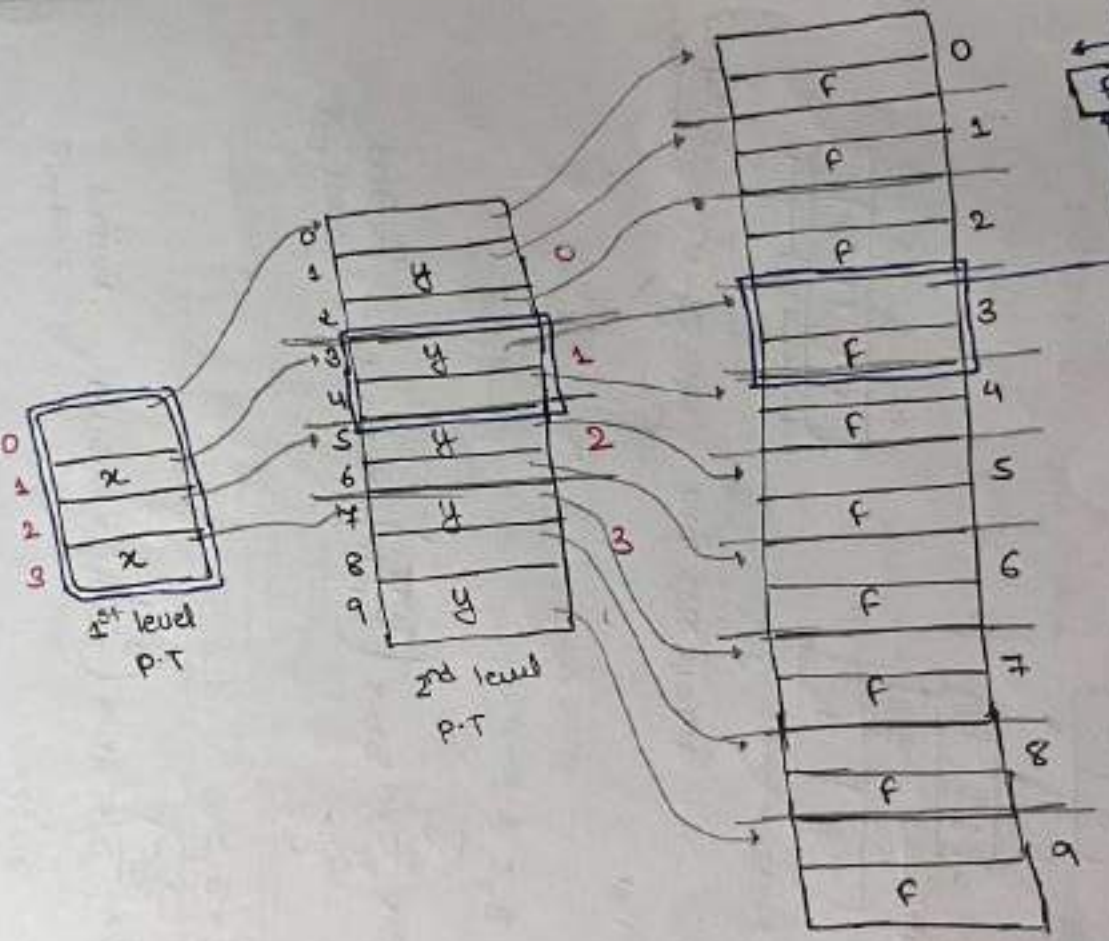
4. In the multi-level paging when paging applied on the P.T then whatever may be the level of paging all the page table entries will contain the frame no.

5. If page size is not mention in the problem the generally page size will be same in all the level.

The page table are Stored in Main memory → P.A.S → Frame

no. of bits in 'x' = no. of bits in 'y' = no. of bits in 'f'





32-B page no 19 | 25 (page size)

32bit virtual address

L.A = 32bit

L.A.S = 4GB

1024B pages

no. of page = LAS / page size

P.T.E 4B → size of page table entry

∴ 3level paging required

P.T.Size <= page size

1st time paging :- P.T.Size = $\left(\frac{2^{32}\,B}{2^{10}\,B}\right) * 4B$

page no

page size = 1KB

$\frac{2^P}{2^p} = 2^0$

$= 2^{22} \times 2^2 B = 2^{24} B$

$= 16MB$

16MB <= 1KB (F)

2nd time paging :- P.T.Size = $\left(\frac{16MB}{1KB}\right) * 4B \Rightarrow \frac{2^{24}}{2^{10}} \times 2^2 B$

$\Rightarrow 2^{16} B = 64KB$

64KB <= 1KB (F)

3rd time paging :- P.T.Size = $\left(\frac{64KB}{1KB}\right) * 4B = \frac{2^{16}}{2^{10}} * 2^2 B$

(T)

256B <= 1KB

$\Rightarrow 2^8 B = 256B$

1st level page table size

another method: LA = 32bits page size = 10bit

L.A = 32bits

P1	P2	P3	d
6	8	8	10

P	d
22	10

14	8

P3

6	8

P1 P2

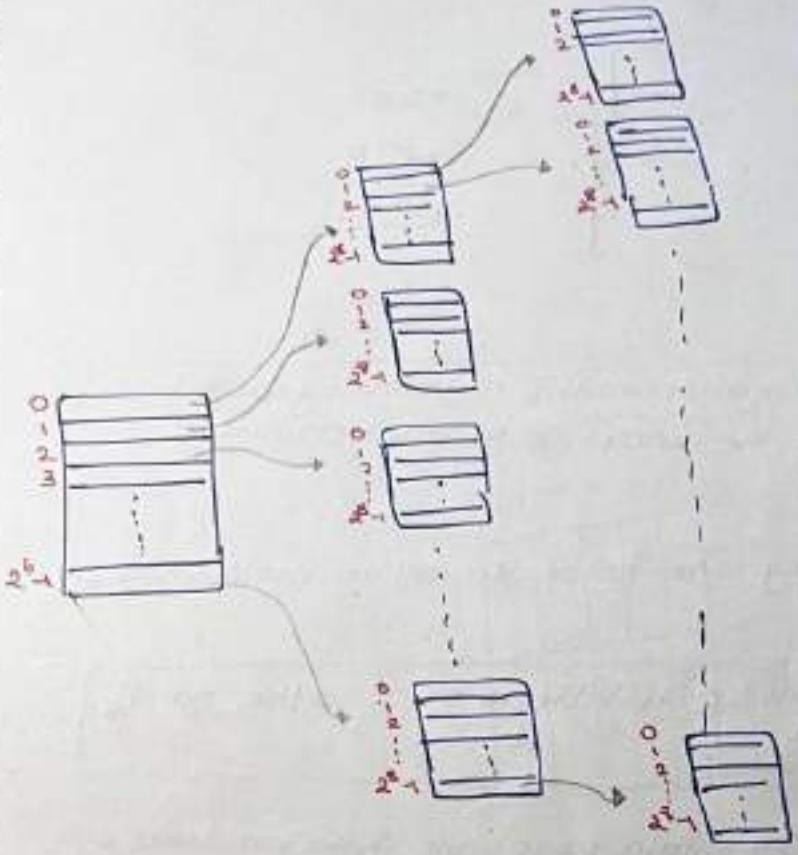
total no. of page's here

no. of page's * PTE = page table size of these level. & then dividing it by page table size will give you next page table page no.

page

$2^{22} * 4 Byte = 2^{22} * 2^2$

$\frac{2^{24}}{2^{10}} \Rightarrow 2^{14} \Rightarrow$ (14bit required)



- Total no. of pages table for a process:-

$(1+2^6+2^6*2^5)$

- Total memory required to store the page table of the process

$(1+2^6+2^6*2^5)*1KB$

Ques] Consider a system which has

L.A = 46 Bits

P.T.E size = 32 bits = 4B

Byte addressable

O.S uses 3 level paging

For Logical to physical address translation & 1st level page table size is exactly same as page size then what is the page size

L.A.S = 64TB

Page size = (xB)

1st time paging

$P.T.Size = \left(\frac{2^{46}B}{xB}\right)*4B = \frac{2^{48}}{x}B$

2nd time paging

$P.T.Size = \left(\frac{\frac{2^{46}B}{xB}}{xB}\right)*4B = \frac{2^{50}}{x^2}B$

3rd time paging $\Rightarrow$ P.T.Size $\left(\frac{\frac{\frac{2^{50}}{x^2}B}{xB}}{xB}\right)*4B = \frac{2^{52}}{x^3}B$

↳ 1st level P.T.Size

1st level paging = page size

$\frac{2^{52}}{x^3} B = xB$

$x^4 = 2^{52}$

$x = 2^{13}$

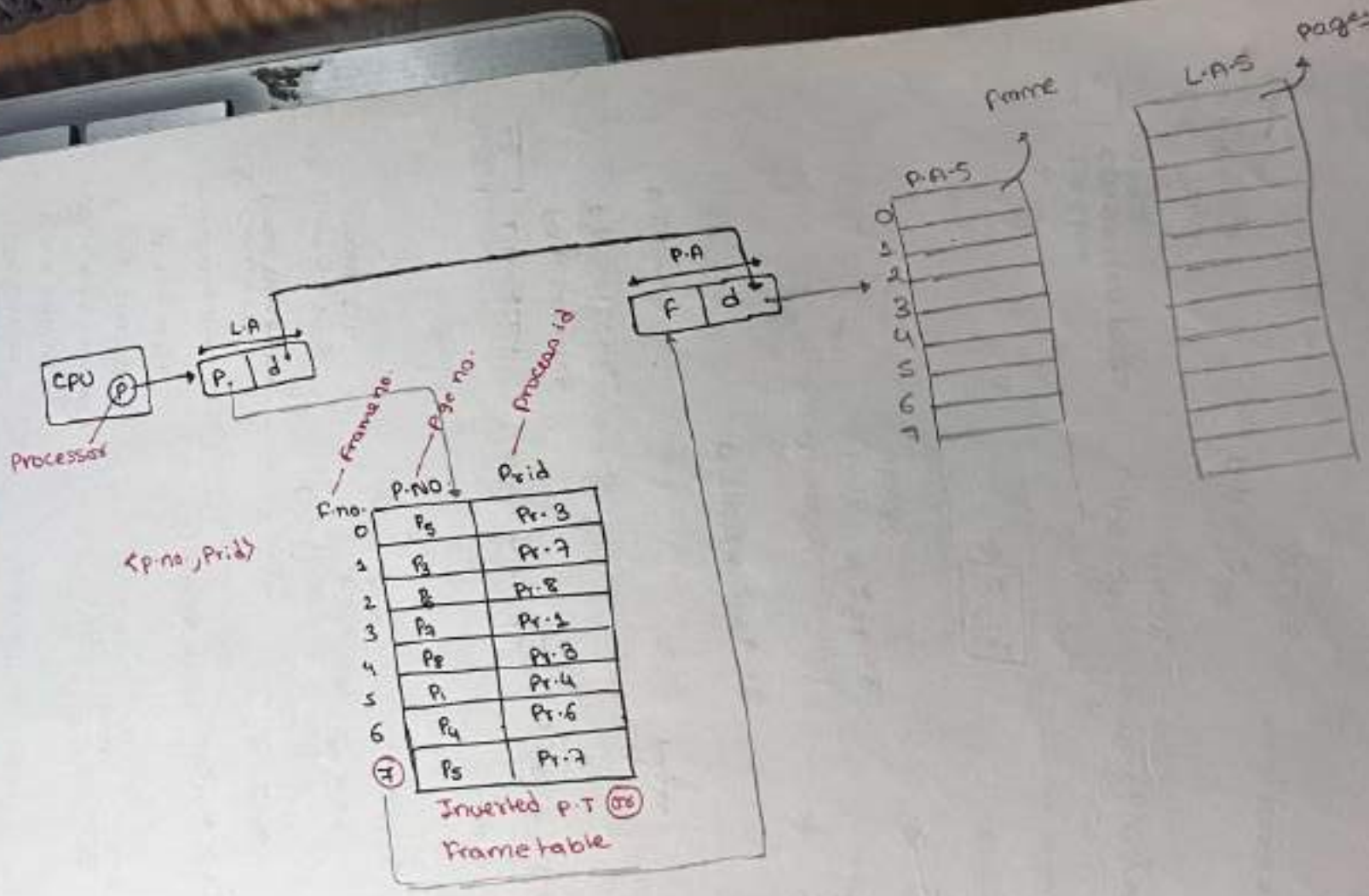
page size $= xB$

$= 2^{13} B$

$= 8KB$

Inverted paging

- To avoid the overhead of maintaining page table for every process. the concept of inverted paging is implemented.
- In the inverted paging only one page table will be maintained for all the process
- no. of entires in the inverted page table is same as the no. of frame in the P.A.S
- the memory required to maintain page table of the processes will be less but the search time for the corresponding page of the process will be more


CPU
P
Processor
LA
P.
d
P.A
f
d
Frame no.
Page no.
Process id
P.NO
Pr.id
F.no.
0
1
2
3
4
5
6
7
P5
P3
P8
P9
P1
P4
P5
Pr.3
Pr.7
Pr.8
Pr.2
Pr.3
Pr.4
Pr.6
Pr.7
Inverted P.T
Frame table
<p.no, Pr.id>
P.A.S
Frame
L.A.S
Pages

Qu] consider a system which have

L.A = 34 bits

P.A = 29 bits

(Byte addressable)

& page size = 16KB

P.T.E = 8B

then calculate

① Conventional page table size

② inverted page table size

L.A.S = 2^{34} ⇒ 16GB

P.A.S = 512MB

no. of pages = $\frac{L.A.S}{\text{page size}} = \frac{16GB}{16KB} \Rightarrow 2^{20}$

① Page table size = $2^{20} * 8B$

⇒ 8MB

② Inverted table size ⇒ $\sum^{15} * 8B$

⇒ 256KB

W.B 23] L.A = 32bit

Page size = 4KB

P.A.S = 128KB

P.T.E = 4B

L.A.S = $\frac{4GB}{4KB} \Rightarrow 2^{20}$ (no. of pages)

P.A.S = $\frac{128KB}{4KB} \Rightarrow 2^{5}$ (no. of frame)

P.T.size = $2^{20} * 4B$

Inverted page table = $2^5 * 4B$

$\Rightarrow \frac{2^{20} * 4B}{2^5 * 4B}$

$\Rightarrow \frac{2^{15}}{1}$

$\Rightarrow \boxed{2^{15} : 1}$

W.B 6] L.A-Shad pages 4KB pages

L.A = 32bit

P.A = 30bit

L.A.S = 2^{32} ⇒ 4GB

P.A.S = 2^{30} ⇒ 1GB

no. of pages = $\frac{4GB}{4KB} \Rightarrow 2^{20}$ (page no.)

no. of frame = $\frac{1GB}{4KB} \Rightarrow \frac{2^{30}}{2^{12}}$

$\Rightarrow 2^{18}$

I.P.T size = no. of entires in I.P.T * P.T.E size.

↓

no. of frame

$\Rightarrow \left(\frac{2^{30}}{2^{12}}\right) * (pno + 12bits)$

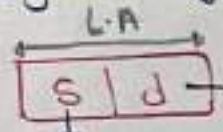
$\Rightarrow 2^{18} * (20 bits + 12bit)$ → no. of bit in page no.

$\Rightarrow 2^{18} * 32bits$

$\Rightarrow 2^{18} * 48 \Rightarrow 2^{18} * 2^2 B$

$\Rightarrow 2^{20} B$

Segmentation

- paging doesn't follow users view of memory allocation
- to achive user view of memory allocation the concept of segmentation is implemented
- in the segmentation L.A.S will be divided into various segments
- Segment of L.A.S will varies in the size
- Segment of L.A.S will be brought into P.A.S
- 

→ no. of bit required to represent word no. of segment
(oo)
Segment offset

no. of bit required to represent Segment no.

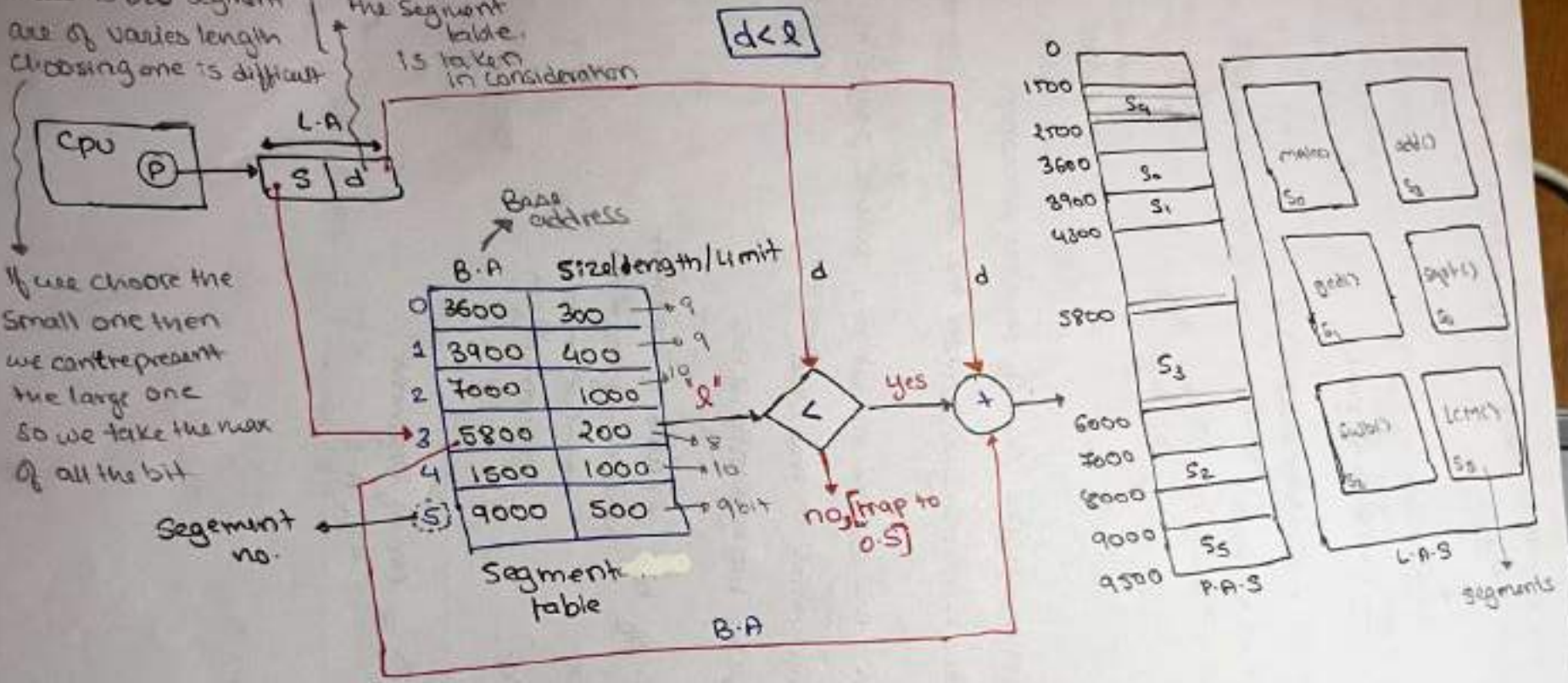
- no. of entires in the segment is = no. of segment in L.A.S

• Segmentation

Why are we doing this is b/c segment are of varies length choosing one is difficult

max of all the bit that are in the segment table. is taken in consideration

If we choose the small one then we can't represent the large one so we take the max of all the bit



Eg: s=3, d=210

d<L | 210<200 (F)

→ Trap to O.S

Eg: s=3, d=190

d<L | 190<200 (T)

B.A + d → P.A.S

5800 + 190 → 5990

- Variable size segment are brought from L.A.S to P.A.S. so it is similarly behaving like variable partition scheme hence segmentation still suffer from external fragmentation.

Segmented paging

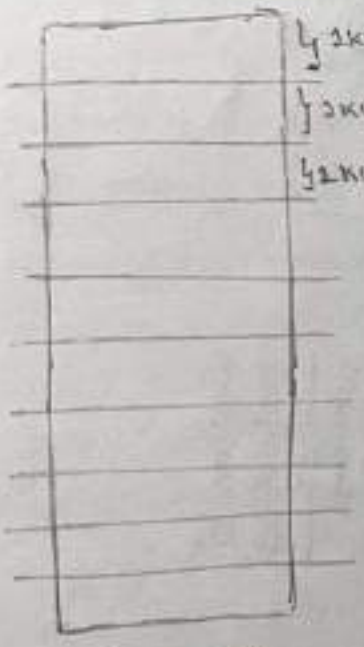
- to avoid the overhead of bringing large size segment to M.M the concept of segmented paging is implemented

L.A = 32 bits

<s,d> = (16,16)

Segment size = 2^{16} w = 64Kw

- In the segmented paging the paging will be applied on the segment instead of bringing the entire segment into M.M. the pages of segmented will be brought into memory.

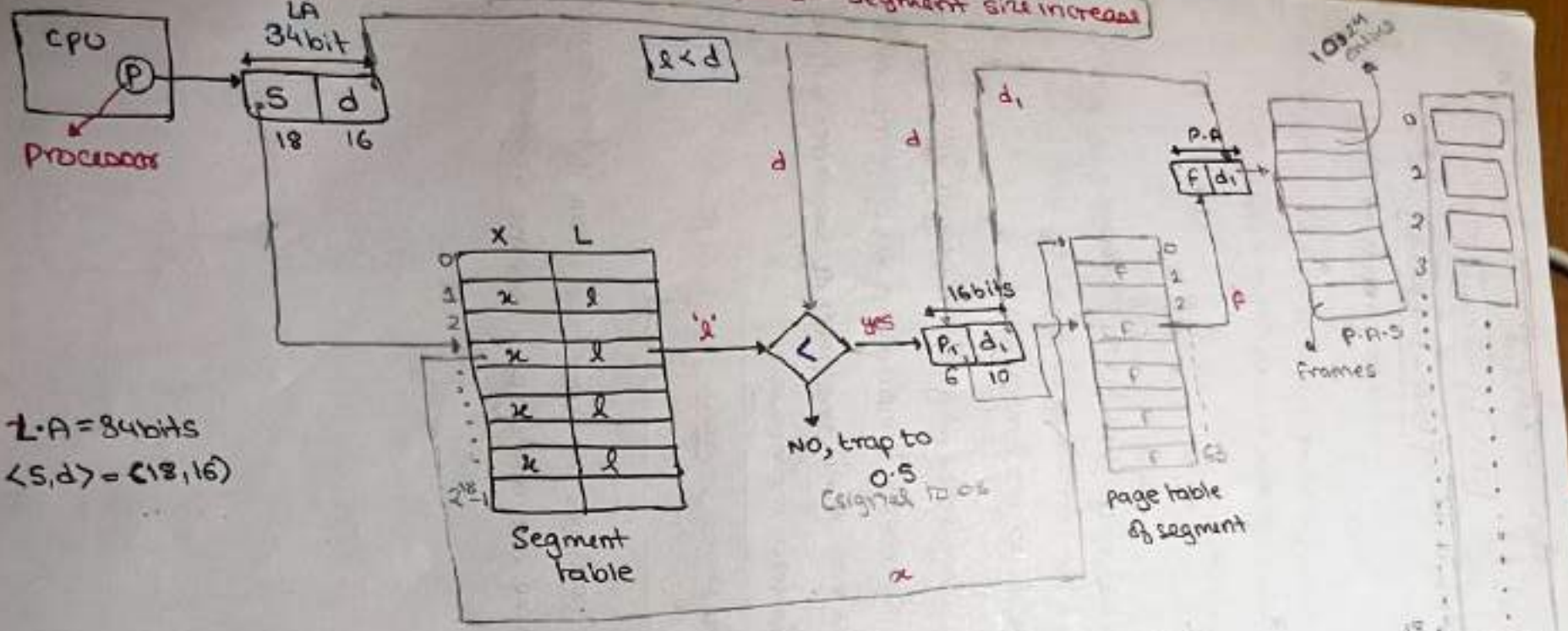


Segment (64Kw)

no. of pages on segment = 64Kw / 1Kw = 64

- no. of entries in the page table of the segment is same as no. of pages on the segment

2. Paging on segment, when segment size increases



L.A = 34bits

<S,d> = (18,16)

P_1 | d_1 → no. of bits required to represent coord no. of page of segment (or) page offset of segment

no. of bits required to represent page no. Segment

- page size of Segment is same as frame size of P.A.S

x = B.A of P.T of respective Segment page size of Segment = 1KW

Qus) consider a system using segmented paging architecture segment is divide into 1K pages & each page is have size entires. the segment no. required is 17 bits. frame no. requires 18 bits. memory is word addressable P.T.E size 2W then calculate

① length of L.A:

② length of P.A:

③ page table size of segment:

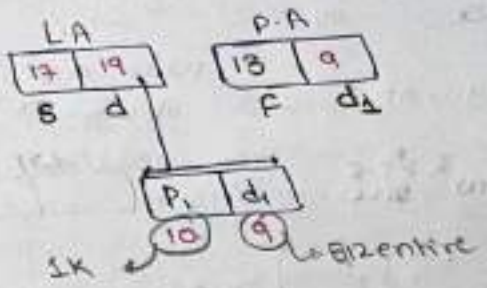
↓

no of entires in the P.T of segment * P.T.E size

(no. of pages on segment

⇒ 1K * 2W

⇒ 2KW



Qus) Conside a system using Segmented paging architecture

Segment is divided into 8K pags, each page is having 2K entires

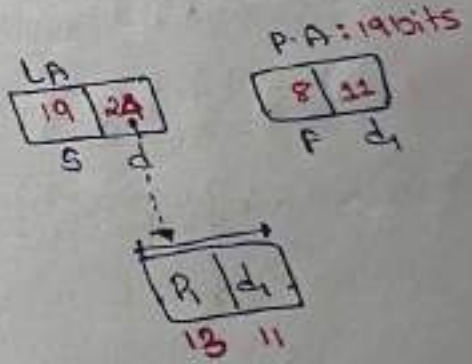
Segment no. requires 19 bits.

memory is byte addressable

the P.A.S is 512 KB &

P.T.E size 8 bits.

then calculate



① Length of L.A : 43 bits

② Length of P.A : 19 bits

③ page table size of segment. : 8K * 8 bits ⇒ KB / 8KB

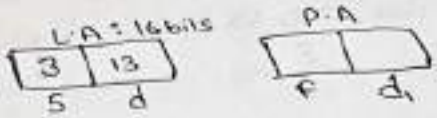
④ no. of frames in the P.A.S : 8 bits

page no. 191

W.B

28

L.A.S = P.A.S = 2^{16} Byte



L.A.S = 64KB

P.T.E = 2B

Segment Size = 2^{16} Byte

Page size of segment = 2^x B

P.T size of segment

= no. of entries in P.T of segment * P.T.E size

↓

no. of page on segment

↓

$\frac{\text{segment size}}{\text{page size}}$

$\frac{2^{13}\ B}{2^x\ B} * 2B = 2^{14-x}\ B$

PT size of Segment = frame size

$2^{14-x} B = 2^x B$

$2^{14} = 2^{2x}$

$2x = 14$

$x = 7$

Page size of segment = 2^x B

= 2^7 B

= 128 B.

- Virtual memory:
- Virtual memory give an illusion to the programmer, that the program of larger size than actual physical memory.

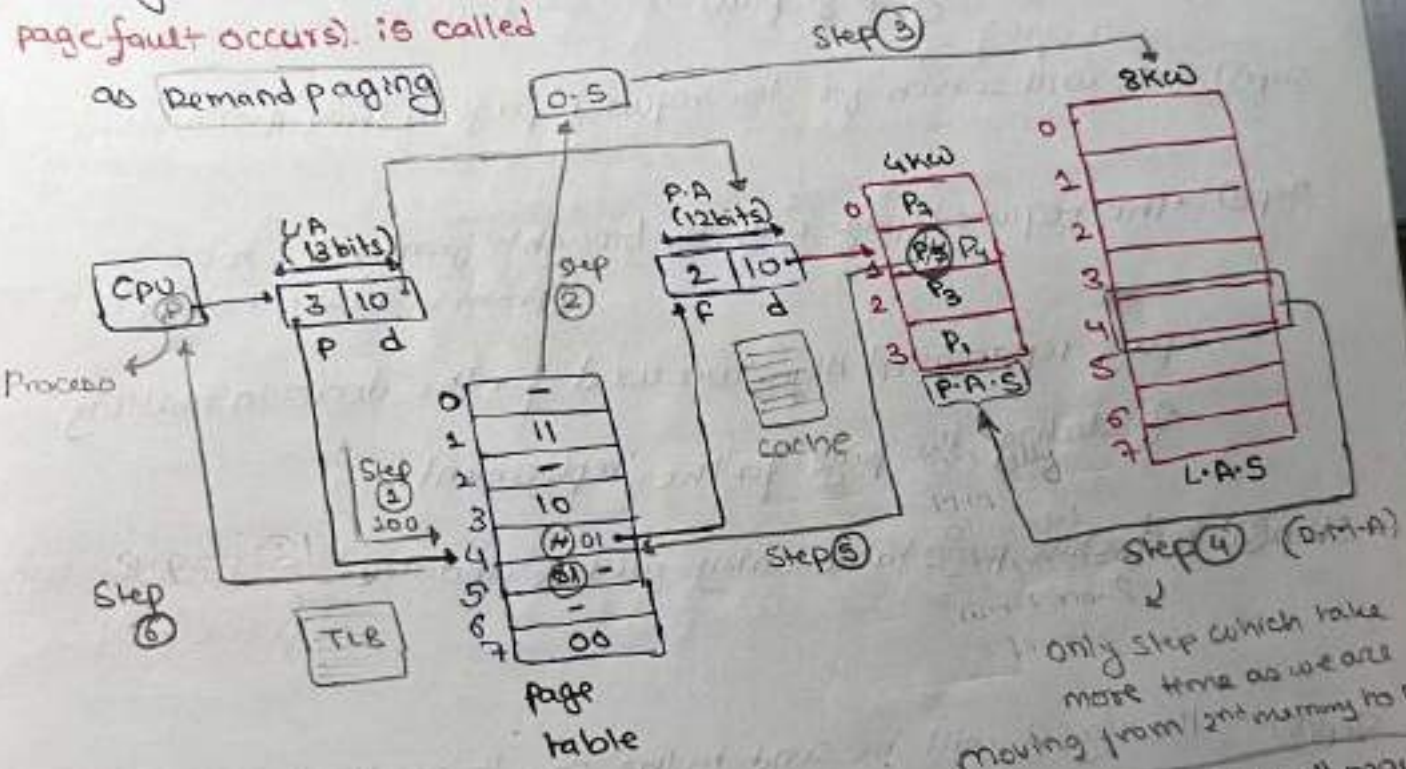
Program size : 8KW

P·A·S : 4KW

Virtual memory is implemented using

- Demand paging (practically implemented)
- Demand segmentation

- Demand paging: Loading the page into memory on demand (whenever page fault occurs) is called as Demand paging



only step which take more time as we are moving from 2nd memory to MM

(Practically possible)

1) L·A > P·A → Reason: It is not possible to satisfy the need of process as it may be larger. So bring the whole process in the P·A which is not possible practically, as the size of MM will be less than secondary memory.

2) CPU → L·A

L·A·S: size of the process w.r.t to no. of pages, Initially all are in virtual memory

P·A·S: actually MM allocated to process w.r.t no. of frame only some space is allocated to a particular process so that more no. of process can came [increasing the degree of multi-programming]

• why cpu generate logical address

↳ To execute the whole process which is present in the L·A·S CPU generate L·A, in order to comple execution of process CPU definitly required all the of L·A·S

∴ when cpu generate address it generates w.r.t size of process = size of L·A·S

Step ②: the cpu is trying to execute the page in the M·M but the page is not currently present in the P·A·S this is know as page fault. (valid or invalid bit will be zero).

Step ③: the program execution will be stop & signal will be send to the o·s regarding page fault (process will go from running to wait state)

step ④ : os will search for the required page in the L·A·S (virtual memory)

step ⑤ : • the required page will be brought from L·A·S to P·A·S (from virtual memory to M·M)

• page replacement Algo. are used for the decision making of selecting the page for the 'replacement.

Step ⑤: respective page table entry will be updated [valid/Invalid bit & frame no.]

Step ⑥ the signal will be send to the cpu to continue program execution, cpu will access the required page in the M·M & Continue program execution (process will brought from wait state to ready state)

- Demand paging is of 2 type
  - Pure demand paging (practical implemented)
    - Initially no page loaded in the P.A.S
  - Pre fetched demand paging (Pre-paging)
    - Initially some page loaded in P.A.S

- Frame allocation policy:
  - equal allocation
  - proportional allocation (practical implemented)

Note

- the main purpose of virtual memory is by allocating less memory to the process. we are trying to execute large size process & more no. of process indirectly we are trying to increase degree of multi-programing
- If the virtual memory is remove from the system the efficient implementation of multi-programing & multi-user are no longer possible.

Page fault Service Time (P.F.S.T)

- the time taken to service the page fault is called as page fault service time
- page fault service time includes the time to perform all the above [6] step

page fault service time = 's'

main memory access time (M.M.A.T) = 'm'

page fault rate = 'p'

- then formula for effective memory access time (EMAT).

page fault time Hit time will be negligible

$E.M.A.T = p*s + (1-p)*m$

includes all time

$s >> m$

ms ns

Q] Consider a system which has MMAT = 35ns.

PPST = 175ns

page hit ratio = 75%.

E·M·A·T = ·25 * 175ns + ·75 * 35ns

⇒ 44ns.

W·B
Page no·197
20

P·P·S·T = 's' = (i+j)

M·M·A·T = 'm' = (i)

Page fault Rate = 'p' = $\left(\frac{1}{k}\right)$

EMAT = p * s + (1-p) * m

⇒ $\frac{1}{k}(i+j) + \left(1-\frac{1}{k}\right) * i$

⇒ $\frac{i}{k} + \frac{j}{k} + i - \frac{i}{k}$

⇒ $i + \frac{j}{k}$

W·B
page no·199
Q.37

⇒ $\frac{1}{10^6} * 10ms + \left(1-\frac{1}{10^6}\right) * 20ns$

⇒ $\frac{1}{10^6} * 10 * 10^6 ns + \left(1-\frac{1}{10^6}\right) * 20ns$

⇒ 10ns + 19.9999ns

⇒ 29.9999 ns ≅ 30ns.

W·B
Page no·196
13

E·M·A·T = p * s + (1-p) * m

modified 70% 20ns

not modified 30% 8ns

2ms = ·P(·7×20 + ·3×8) + (1-P)×1

P×16.4 + 1 - P = 2ms ⇒ P = $\frac{1}{15.4}$ = 0.064

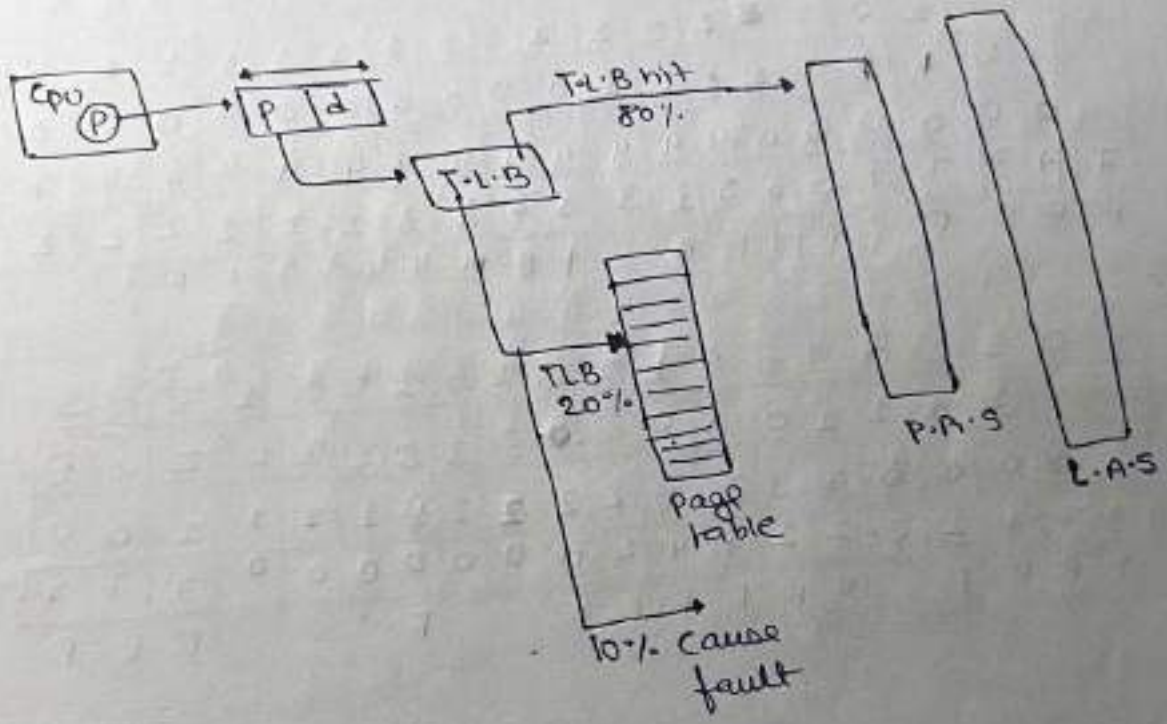
W B
Page no
Pg.9
21

S = 200MS
M = 10MS.
80% references are found in the TLB

4 remaining 20% cause page faults.

E·M·A·T = 0·80(0 + 10MS) + 0·20(0 + 0·30 * 200MS + 0·90 * 10MS) = 13·9 MS

TLB hit ratio
TLB time
MMAT
TLB miss
TLB time
Page fault
P·F·S·T
NO·P·P
MMAT



- Page Replacement algorithms:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

page no. referred by the process in the L.A.

Note
- no. of frame allocated to the process is decided by instruction set architecture

1- First IN First OUT (F.I.F.O):

no. of frames: 4

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
			2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
		1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	7	7	7
7	7	7	7	7	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2
F	F	F	F		F		F			F			F	F			F		

Q] 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

		(2)	1	1	1	(0)	0	0	(3)	3	3	3	3	(2)	2	2	2	2	(2)
	(0)	0	0	(0)	(3)	3	3	(2)	2	2	2	(1)	1	1	1	1	(0)	0	
(7)	7	7	(2)	2	2	2	(4)	4	4	(0)	0	0	0	0	0	(7)	7	7	
F	F	F	F		F	F	F	F	F	F		F				F	F	F	

(15)

Q] 1,2,3,4,1,2,5,1,2,3,4,5.

		3	3	3	2	2	2	2	2	4	4
	2	2	2	1	1	1	1	1	3	3	3
1	1	1	4	4	4	5	5	5	5	5	5
F	F	F	F	F	F	F			F	F	

(9)

			4	4	4	4	4	4	3	3	3
		3	3	3	3	3	3	2	2	2	2
	2	2	2	2	2	2	1	1	1	1	5
1	1	1	1	1	1	5	5	5	5	4	4
F	F	F	F			F	F	F	F	F	F

(10)

- Belady's anamolys : By increasing the no of frame to the process no of page fault should decrease but instead they are increasing
- these problem is known as Belady's anamolys
- which all algorithm face Belady's anamolys? & why?
- optimal page Replacement : In the event of page fault replace the page which is not used for longest duration of time in future

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
		1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
F	F	F	F		F		F						F				F		

Total page fault = 8

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F			F			F				F		

total page fault = 9

- LRU page Replacement : In the event of page fault, the Replace the page which is least Recently used

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		(1)	1	1	(3)	3	3	(2)	2	2	2	2	2	2	2	(7)	7	7	
	(0)	0	0	0	0	0	0	0	(3)	3	3	3	3	3	(0)	0	0	0	0
(7)	7	(7)	(2)	2	2	2	(4)	4	4	(0)	0	0	(1)	1	1	1	1	1	1
F	F	F	F		F		F	F	F	F			F		F		F		Hit

most recently

least recently

sequence:- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

with 4 frame

	4	2	3	0	3	2	1	2	0	1	7	0	1
	2	2	2	2	2	2	2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	3	3	3	3	3	3	3	7	7	7
F				F			F				F		

F F F F F F

Total page fault = 8

- LRU Page Replacement: In the event of page fault, Replace the page which is most Recently used

Frame = 4.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	0	0	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	0	0	0	3	0	4	4	4	4	4	4	4	4	4	4	4	4	4	
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	
F	F	F	F	F	F	F	F			F	F	F		F					

12

Frame = 3

3+13

=16

W-8
Page no. 197
12

Page size = 100 records

Records #	page #
0-99	0
100-199	1
200-299	2
300-399	3
400-499	4
500-566	5

Address :- 100, 200, 400, 499, 510
530, 560, 320, 220, 240,
200, 320, 370.

Reference String :- 1, 2, 4, 4, 5, 2, 2, 3,

will not contain continues same page

7 7 4 5 7 3
F F F F F F

note

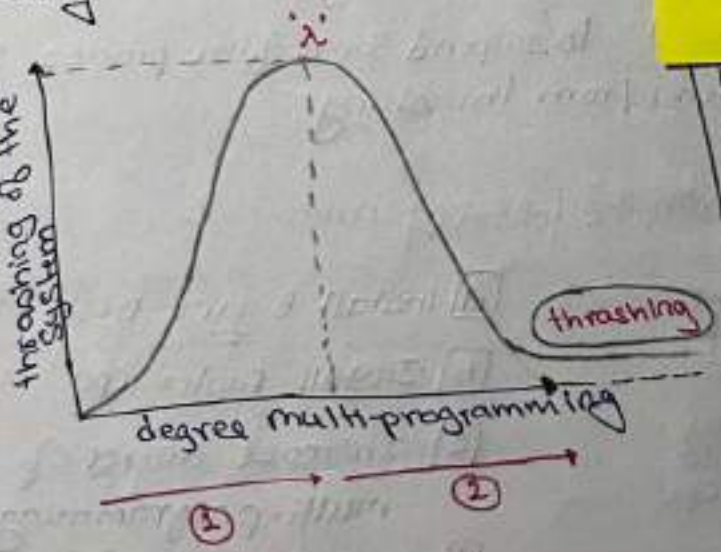
- reference string will never contain continues same page Eg: 1, 2, 3, 4, 4, 3, 2, 1 ✗
  1, 2, 3, 4, 3, 2, 1 ✓

- working set model.
- It is based on assumption of locality
- It use Δ, working set window
- Idea to examin recent (Δ) page replacement.
- If a page is in active use, it will be in working set window & if it is no longer being used it will be dropped from working set.

- If there is only one frame allocated to process then every page refer. red w... be a page fault

Eg:

- Thrashing:-



Each p...
1 = 4 frames

#f processes 400
Each process 'p' get = 1 frame
2

Note

- In the initial degree of multiprogramming upto the point of 'λ' the throughput of the system is very high & the system resource are utilised perfectly
- If we further increase degree of multiprogramming after the point of 'λ' the throughput of the system will drastically fall down the system will spend more time only in the page replacement. the time taken to complete execution of a process will increase this situation in the system is know as Thrashing.

note

- working set (t,k) at an instant of time t is the set of pages used by k most recent memory reference in the last t time unit.

- Causes of thrashing
  1) the high Degree of multiprogramming
  2) Lack of frames (M.M)
- Recovery of thrashing
  1) do not allow the system to go into thrashing, instruct the L.T.S (Long term schedulers) not to bring process into memory after the point of "λ"
  2) if the system is already in the thrashing then instruct the M.T.S(Mid term schedules) to suspend some of the process so that we can recover the system from thrashing.

Que) Consider the system with the following parameter.

CPU utilization → 10%
Paging Disk → 90%
} thrashing

then which of the following factor will improve the CPU utilization

time spent on Page Replacement.

(a) Install Bigger Disk : NO
(b) Install Faster CPU : NO
(c) Increase degree of multi-programming : NO
(d) Decrease degree of multiprograming : yes
(e) Install more M.M : yes
(f) Decrease page size — NO
(g) Increase page size — LIKELY

File System:

File: collection of logically related entities (records).

Attribute: file will have various attribute

- Name
- Type
- Size
- location
- creation time
- last modified date
- owner
- permissions
- passwords

→ File context

- all the attribute of the file is known as file context
- File context will be store in the (F·C·B)(File Control Block)

- Various operation on the file
  1) Create
  2) open
  3) write
  4) edit
  5) delete
  6) close
  7) save
  8) save as
  9) read
  10) rename
  11) cut
  12) copy
  13) paste
  14) print
  15) send/share
  16) compress

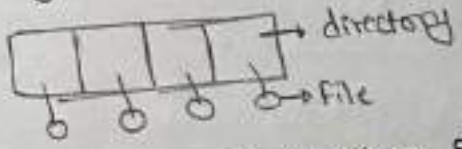
- Type of file
  - .doc
  - .txt
  - .obj
  - .bat
  - .exe
  - .pdf
  - .png
  - .xlsx
  - .ppt
  - .jpg
  - .mp3
  - .mp4
  - .mkv/.avi...
  - .c/.cpp/.java...

- file have various access methods.
  1) Sequential access
  2) Random access

Note
For the better classification of file file will be store in the directory

Directory Structure

1) Single level directory

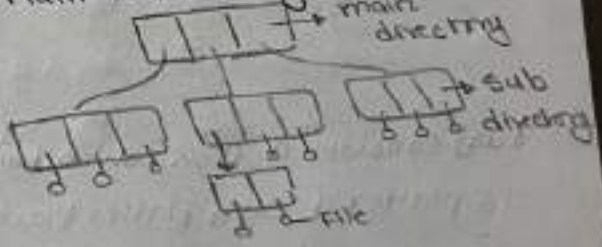


- Implementation of directory structure is easy & simple
- Two file can't have the same name
- Search time for the file will be more

Disadvantage of multi-level directory

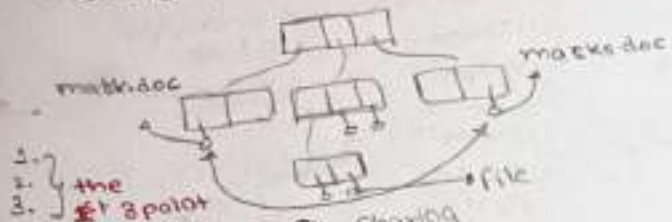
- If the some file exist in the two different directory then if one file is update the other file has to be updated A/c other wise inconsistency

2) Multi-level directory



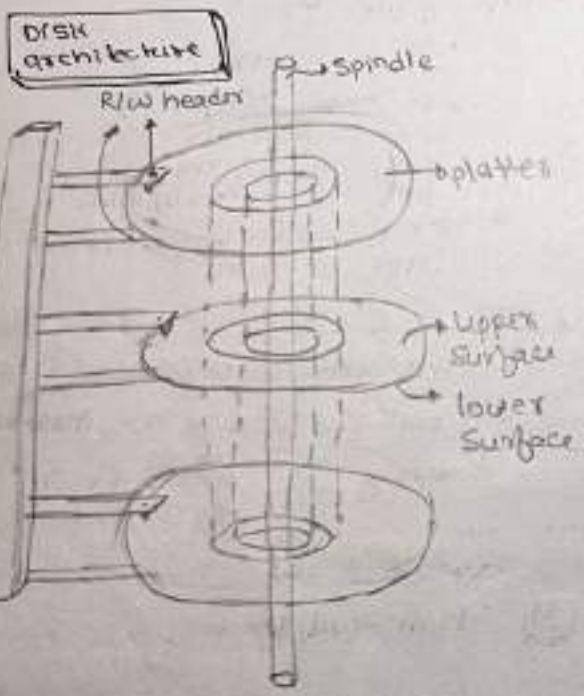
- Implementation of the directory structure is difficult or complicated
- better classification of the file as per the criteria.
- search time of the file will be less b/c we classified so it will take less time

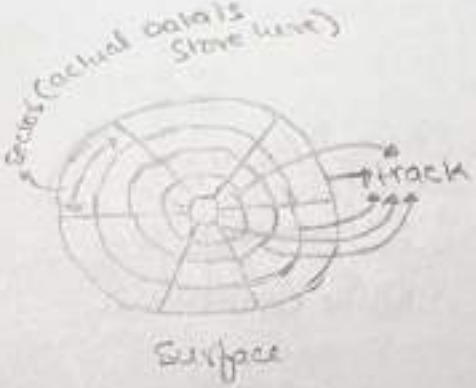
- Acyclic graph directory

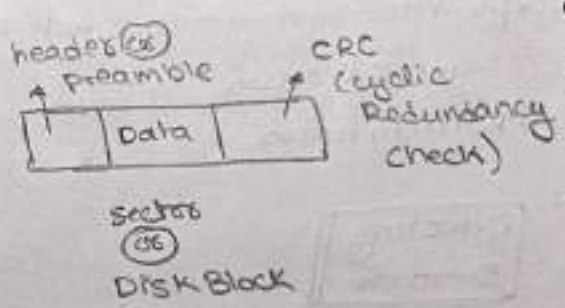


1.
2.
3. } the 3 point is same as multi-level directory
4.

4th point] If the same file exist in two different directory then if one file is updated then other file will be updated automatically by using file sharing Concept







Ques] Consider a Disk which has a 16 platter & each platter have 2 surface & each surface is divided into 8K track & each track is further divide into 512 sector & each sector can store 32 KB data then calculate

① Capacity of Disk
② no. of bits required to identify the specific sector of Disk

① $\Rightarrow 2^4 \times 2^1 \times 2^{13} \times 2^9 \times 2^5 \Rightarrow 2^{42} \Rightarrow$ 4TB

② $\Rightarrow 2^4 \times 2^1 \times 2^{13} \times 2^9 \Rightarrow 2^{27}$

27 bit

Disk Space Allocation method

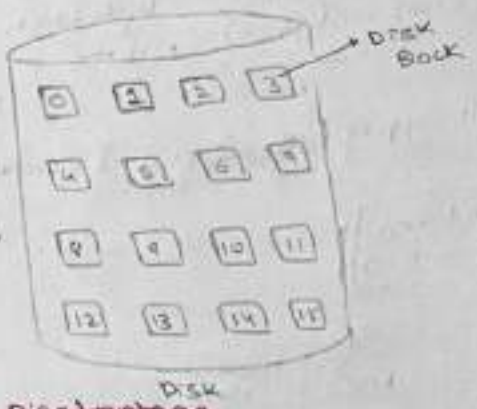
- contignous allocation:

File	Starting D·B·A	Size
abc·doc	2	4
xyz·doc	9	5

disk block address

w.r.t to no. of disk Block

0, 4 offset value are from here



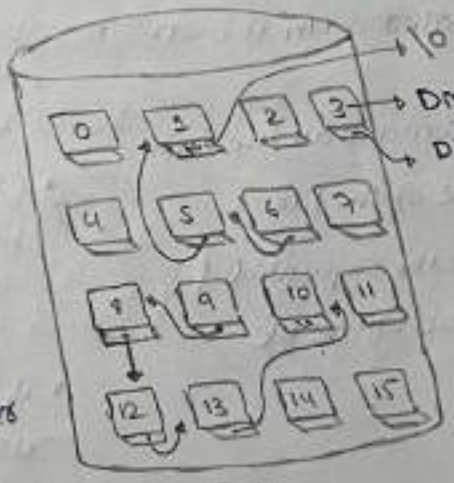
1. In the contignous allocation whenever the file is created these block are allocated in the continous manner
2. ever file is associated with the 2 parameters (a) Starting DBA (b) size.
6. It support both sequential & Random access of file

- Disadvantage
3. Increasing the file size may not be possible always
4. It is suffering from external fragmentation
5. Internal fragmentation may exist in the last Disk block of a file.

Random access = Starting D·B·A + offset

Linked allocation (non-contignous)

File	Starting D·B·A	Ending DBA
abc·doc	2	1
xyz·doc	9	10



\0 (null ptr)

Disk Blocks

Disk Block address

pointer to the next D·B of file

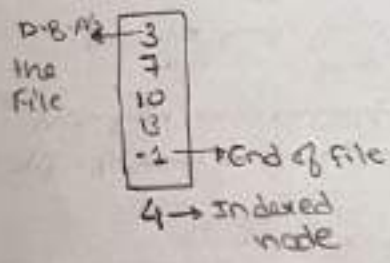
- in the linked allocation coherence the file is created the disk block are allocated in a non-contignous manner
- ever file is associate with 2 parameter → Starting D·B·A → Ending D·B·A
- Increasing the file size is always possible if the free disk block is available
- there is no external fragmentation
- internal fragmentation exist in the last disk block of the file

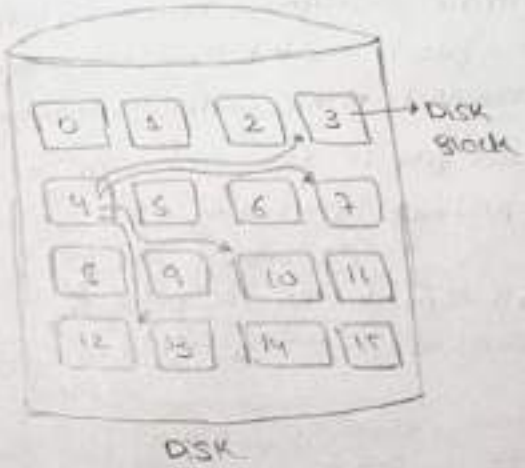
- problem of Linked allocation (non-contignous)
  - there is a overhead of maintaining the pointer in every disk block
  - if the point of any disk block is lost then file will be truncated

- It support only sequential access of the file

Indexed Allocation

File	Indexed node
abc.doc	4





- in the index allocation every file is associated with it own index node
- Index node contain all the disk block address of the file
- It support both sequential & random access of the file

- there is no external fragmentation but internal fragmentation may exist in the last disk block of the file
- if the file is very large one disk block may not be sufficient to store all the disk block address. (if happen it become Linked allocation).
- if the file is very small then it is waste of using entire one disk block to store the disk block address

- Unix - I-Node Implementation (An Extension of Indexed allocation)

File	I-Node
doc-doc	27

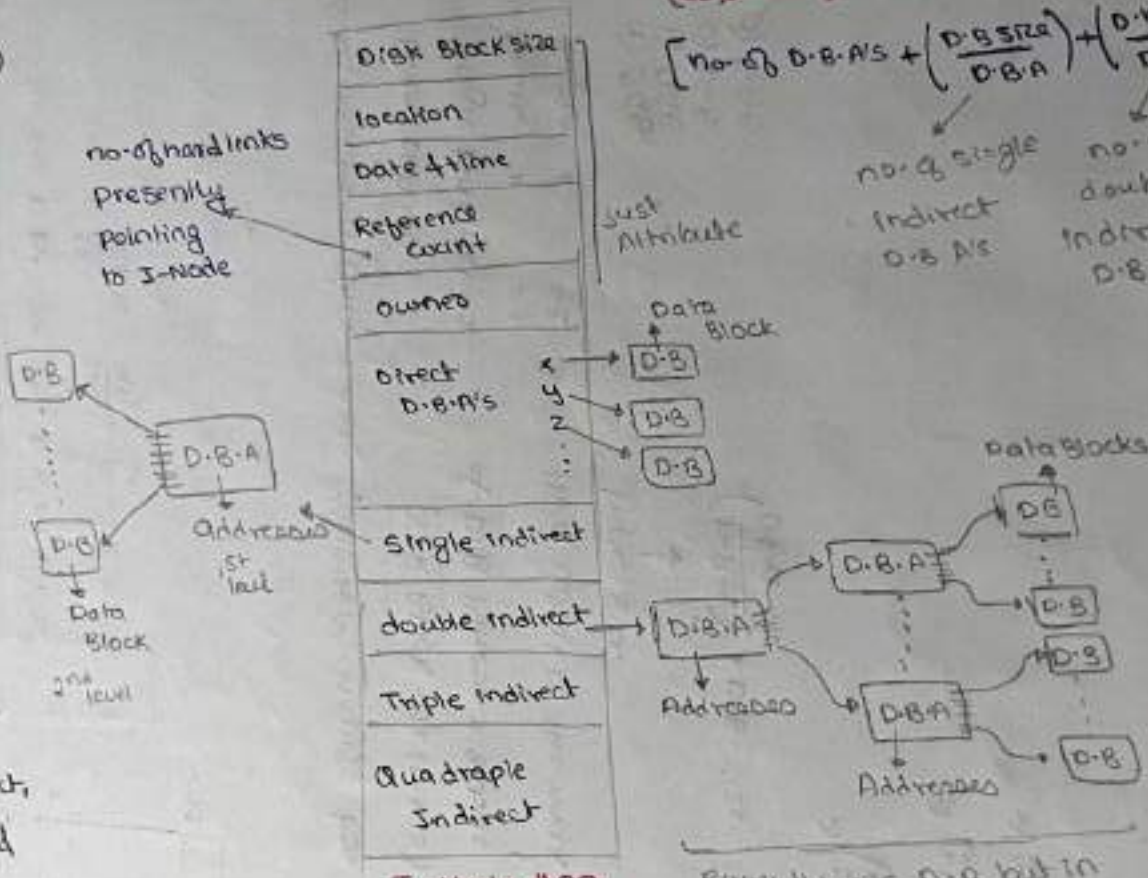
- Unix follow I-Node way of Implementation in order to allocated the disk space to the file
- Every file is associated with its own INode
- Whenever the file is created depending on the size of the file, file will be always stored from Direct D·B·A to single Indirect, double Indirect, triple Indirect & so on..

Total size of file system (or) max file size =

$$\left[\text{no. of D·B·A's} + \left(\frac{\text{D·B size}}{\text{D·B·A}}\right) + \left(\frac{\text{D·B size}}{\text{D·B·A}}\right)^2 + \cdots\right] * \text{D·B size}$$

no. of single Indirect D·B·A's

no. of double Indirect D·B·A's



• If we assume Disk Block size = 1KB

Then ① File size (F.S) ≤ 10 KB

→ It will be stored in direct D.B.A's

② F.S > 10KB · 4 F.S ≤ 110KB

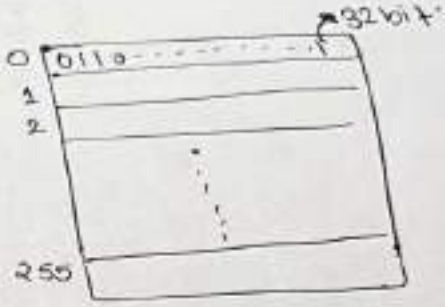
→ First 10KB will be stored in Direct D.B.A

→ next 100KB, will be stored in single indirected.

D.B.A Size / D.B.A = no. of DBA's one disk Block can contain

Ex: D.B. Size = 1KB D.B.A = 32bits

$\frac{1KB}{32bits} = \frac{1KB}{4B} = \frac{2^{10}}{2^2} = 2^8 = 256$



Ques) Consider a unix I-node which is having

Direct DBA = 10

Single Indirect = 1

Double " = 1

Triple " = 1

Disk Block size (D.B.S) = 1 KB

& D.B. Address = 32bits

then what is the max file size possible using triple size

a) 2GB
b) 4GB
c) 8GB
d) 16GB

$\left(\frac{D.B.S}{D.B.A}\right)^3 * D.B.\ size \rightarrow \left(\frac{1KB}{32bit}\right)^3 * 1KB$

$\Rightarrow \left(\frac{2^{10}}{4}\right)^3 * 1KB$

$\Rightarrow (2^8)^3 * 1KB$

$\Rightarrow 2^{24} KB \Rightarrow$ 16GB

page no. 205
W-B
Q6

5-direct ptr
3-single indirect
2-double "
D·B·S=4KB
D·B·A=10B

$\Rightarrow \left[5+3\left(\frac{4KB}{10B}\right)+2\left(\frac{4KB}{10B}\right)^2\right]*4KB$

$\Rightarrow 5+120+3200*4KB$

$\Rightarrow 3325*4KB$

$\Rightarrow 13300000$ byte

Q7] 10 direct ptr
1 double indirect
1 triple "
1 quadruple "
D·B·S=1*B

$\left(\frac{D.B.SIZE}{D.B.A}\right)^4 * D.B\ SIZE = 4TB$

$\left(\frac{1KB}{xB}\right)^4 * 1KB = 4TB$

$\left(\frac{2^{10}}{x}\right)^4 * 2^{10} = 2^2 * 2^{40}$

$\frac{2^{40}}{x^4} * 2^{10} = 2^2 * 2^{40}$

$x^4 = 2^8$

$x^4 = (2^2)^4$

$x = 4B \Rightarrow$ So, $x = 32bit$

- Disk Free Space Management

Size of Disk = 20MB

Disk Block size = 1KB

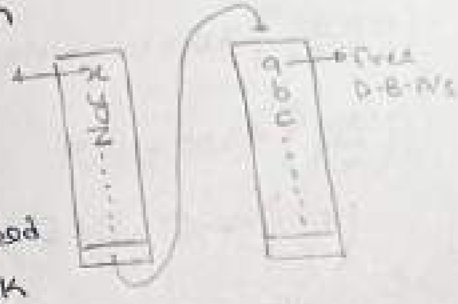
D.B.A = 16 bits

no of Disk Block available on the given Disk = $\frac{\text{Size of Disk}}{\text{D.B. Size}}$

$\Rightarrow \frac{20MB}{1KB} = \frac{20 \times 2^{20}}{2^{10}} = 20 \times 2^{10}$

$\Rightarrow$ **20K**

① Free list Approach



- In free list Approach Some Disk Block are used to store the free disk block Address.

→ no. of D.B.A's possible to store '1' D.B

$\frac{\text{D.B. Size}}{\text{D.B.A}} = \frac{1KB}{16 bits} = \frac{1KB}{2B} = \frac{2^{10}}{2^1} \Rightarrow 2^9 = 512$

→ 512 D.B.A ⟷ we can store in 1 D.B

So, to store 20K D.B.A's ⟶ ?

$\frac{20K}{512} = \frac{20 \times 2^{10}}{2^9} = 20 \times 2$

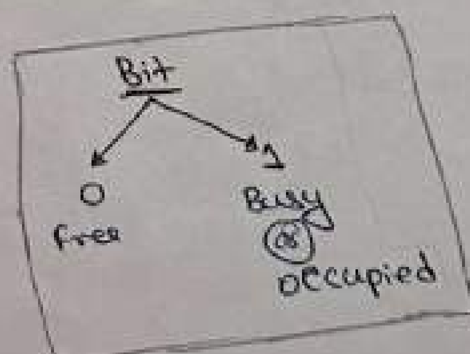
$= 40$ //

② Bit map Approach

Free Disk Block:

1	2	3	4	5	6	7	8
1	0	1	1	0	1	0	0
1	1	0	0	1	0	0	0
0	0	0	0	1	1	1	0
0	0	1	0	1	0	0	0
1	1	0	1	1	1	0	1
0	1	1	1	1	0	1	0

Bit map

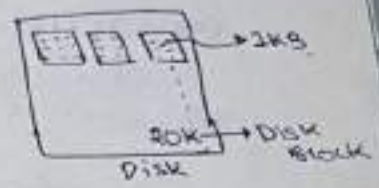


these also Disk Block only

* In Bit map Approach every disk block will be mapped with 1 Binary Bit.

Dno. of Disk Blocks we can map in 1 D.B

D.B Size = 1KB = 1K * 8bit = 8K bits



2) 8K Disk Block ⟷ one can map to 1 DB

to map 20K Disk Blocks ⟶ ?

$\frac{20K}{8K} * 1 = \frac{20}{8} = 2.5$ (3)

W.B 16

Harddisk Capcity : 40MB

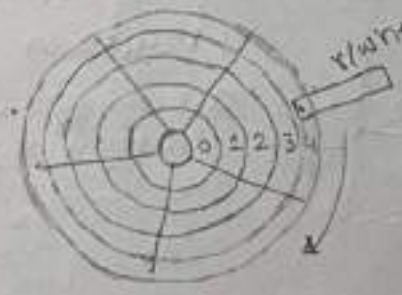
D.B.S = 2KB

D.B.A = 3B

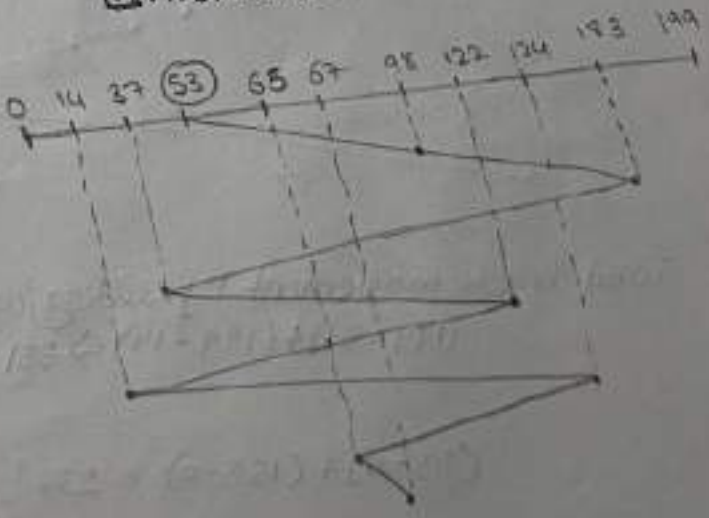
Disk Scheduling

Track Request :- 98, 183, 37, 122, 14, 124, 65, 67

Goal: To minimize the avg. seek time of the Disk



1. First Come First Serve (F.C.F.S)



④ Total Track movement made by Read/write header (seek time):-

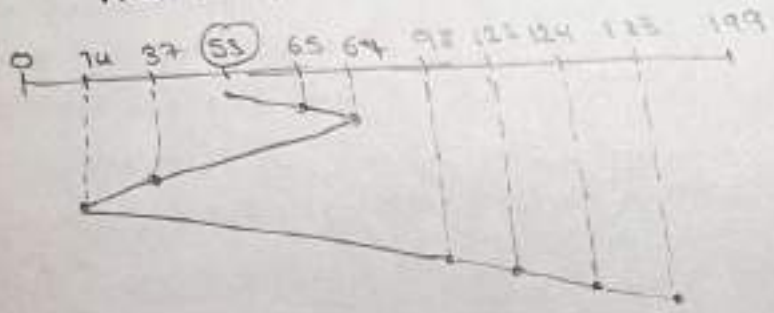
(98-53)+(183-98)+(183-37)+(122-37)+(122-14)+(124-14)+
(124-65)+(67-65)
+85+ +85+105+110+59+2 ⇒ 640

⇒ 640 * 1ms ← seek time

Total seek time ⇒ 640 ms

[2] Shortest seek time First (S.S.T.F):

Nearest Track next:



Sequence = 98, 183, 37, 122, 14, 124
65, 67

- Total track movement by Read/write header.

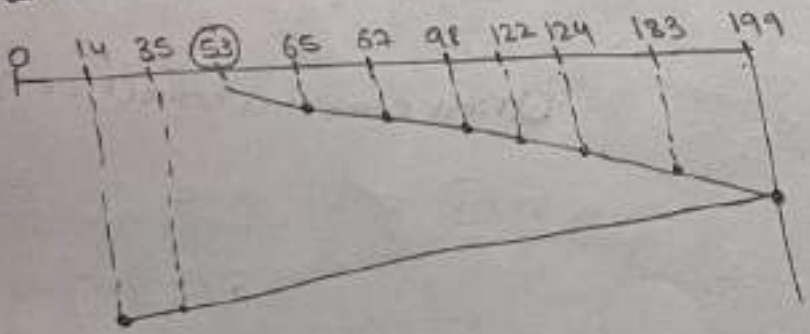
(67-53)+(67-14)+
(183-14)

14+53+169

⇒ 67+169

⇒ 236

[3] SCAN:- (Elevator): need to specify direction



Sequence: 98, 183, 37, 122, 14,
124, 65, 67

Assuming: it moving in right direction

-ve left direction

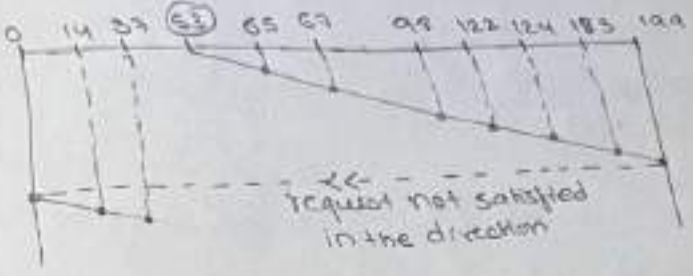
+ve right direction

Total track movement by Reading/write header

(199-53)+(199-14) ⇒ 331 } if it move right

(53-0)+(183-0) = 236 } if it move left

* C-SCAN :
↓
Circular



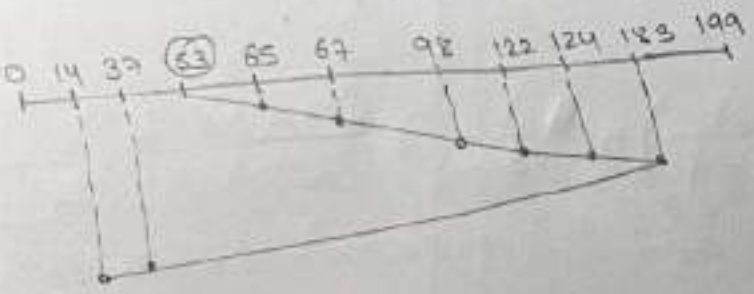
Total track movement made by R/W header:-

(199-53)+(199-0)+(37-0)

⇒ 382

* LOOK : (Look for last pending request in the direction of R/W header)

sequence: 98, 183, 37, 122, 14, 124, 65, 67



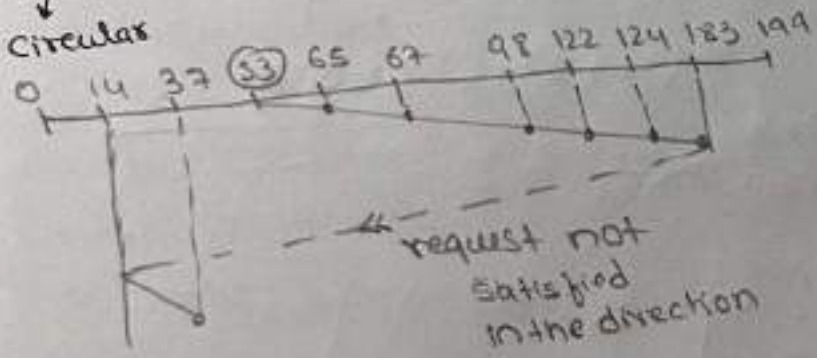
Total Track movement made by r/w header

(183-53)+(183-14)

⇒ 130 + 169

⇒ 299

* C-LOOK:
↓
Circular



Total track movement made by r/w header

(183-53)+(183-14)
+(37-14)

⇒ 299 + 23

⇒ 322

Chapter: 5

PYQ

5:12

10 direct pointer
one single
one double
one triple
Disk block size: 1KByte
Disk Block address: 32bit
48bit integer is used
max possible file size?

$\Rightarrow \left\lfloor \frac{DBS}{DBA} \right\rfloor = \frac{2^{10} \times 8 \text{bit}}{2^5 \text{bit}}$

$\Rightarrow \frac{2^{13}}{2^5} = 2^8$

$\text{max file size} = \left[10 + (2^8) + (2^8)^2 + (2^8)^3\right] * DBS$

$\Rightarrow \left[10 + 2^8 + 2^{16} + (2^{24})\right] * 1KB$

Highest effect overall

$\Rightarrow 2^{24} * 2^{10}$

$\Rightarrow 2^{34} B$

5:14

Data block size: 1024Byte

10 direct
1 single indirect
1 double indirect
1 triple indirect

each block can contain address for 128 blocks.

$\frac{D \cdot B \cdot S}{D \cdot B A}$ = no. of Disk Block address Store inside one Block

128 or 2^7

$\text{max file size} = \left[10 + (2^7) + (2^7)^2 + (2^7)^3\right] * 1024Byte$

highest effect

$\Rightarrow (10 + 2^7 + 2^{14} + (2^{21})) * 2^{10} B$

$\Rightarrow (16 + 2^{21}) 2^{10}$

$\Rightarrow 2^{31}$

$\Rightarrow 2GB$ (Approx).

5.23

300 GB disk
use d escriptor with
8 direct
1 indirect
1 double indirect

D·B·S = 128B
D·B·A = 8B
max. possible file size

$\left(\frac{D \cdot B \cdot S}{D \cdot BA}\right) = \left(\frac{128B}{8B}\right) = \frac{2^7}{2^3} = 2^4$

$\text{max file size} = \left[8 + (2^4) + (2^4)^2\right] * 128B$

$= 8 + 2^4 + 2^8 * 2^7$

$\Rightarrow 2^7 B (2^3 + 2^4 + 2^8)$

$\Rightarrow 2^{10}B + 2^{11}B + 2^{15}B$

$\Rightarrow 32KB + 2KB + 1KB$

$\Rightarrow$ 35KB

5.30

12 direct
1 single
1 double
indirect

D·B·S = 4KB
D·B·A = 32bit long

max. file size

$\left(\frac{D \cdot B S}{D \cdot B \cdot A}\right) = \frac{4KB}{32bit} \Rightarrow \frac{2^{11} \times 2^3}{2^5} bit = 2^9$ or 512

$\text{max file size} = \left[12 + (2^9) + (2^9)^2\right] * 4KB$

$\Rightarrow (12 + 2^9 + 2^{18}) * 2^{11} B$ wrong

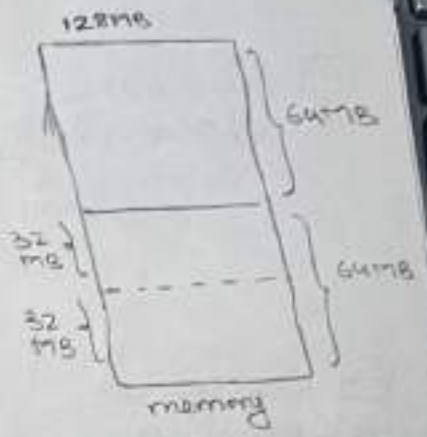
$\frac{DBS}{DBA} = \frac{4KB}{4B} \Rightarrow 2^{10}$

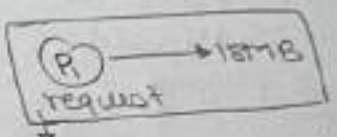
$\text{max file size} = \left[12 + (2^{10}) + (2^{10})^2\right] * 4KB$

$\Rightarrow (12 + 2^{10} + 2^{20}) * 2^{11} B$

* Buddy System:

1. In the buddy system initially the memory will be single continous free block.
2. whenever the request by the process arrives then memory will be divided into "2 half block"
3. If the request by the process is to small then lower block of the memory is further divided into "two half block" again
4. In the buddy system the memory will be allocated from lower block to higher block





after the arrival of P_1 the memory of 128MB is divide into 2 half 64MB, 64MB further P_1 is still small So again it is divided into 32MB, 32MB but now we can't further divided it as if we divide the value become 16MB, 16MB & P_1 can't be store here.

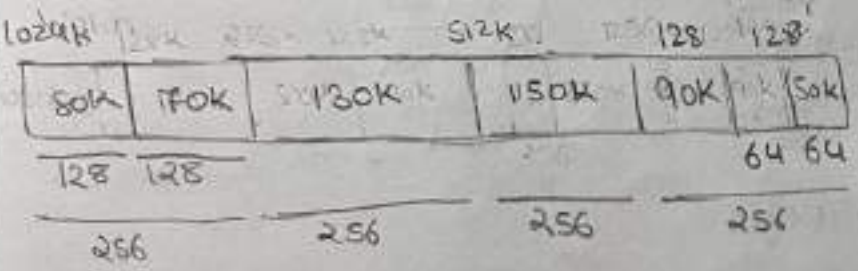
- P_1 will be located at the lower & any other process which come will be put higher; ie we go lower to higher

W.B Page no 189 Q3

1MB memory using buddy system

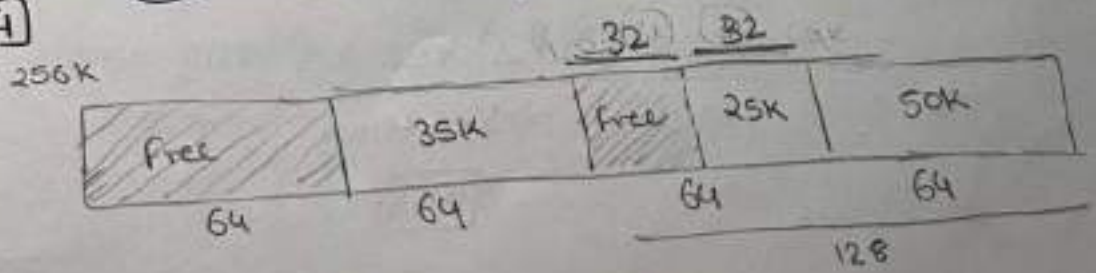
Request: 50K, 150K, 90K, 130K, 70K 80K, 120K, 180K, 60K.

higher ↓ lower ↓



So, (120K) is the 1st to fail due to lack of resource

4]



32K & 64K are left free

W.B solution
Chapter: 6

22 given address:

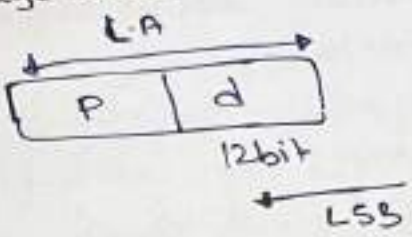
$0x\ 304F48AC5345$

$(0011\ 0100\ 0101)_2$

↓

(837)

Page size = 4KB → 2^{12}



27 Read the 4th Imp. point in Multi-level paging.

P.A = 36bit

Page size = 4KB

$\#f$ of frame $= \frac{2^{36}}{2^{12}} = 2^{24}$

24, 24 | 24

32 2 level paging with TLB

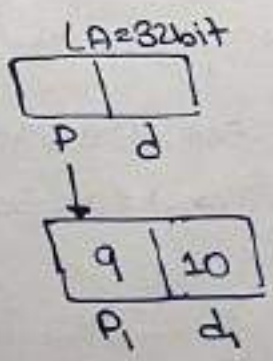
L.A = 32bit

Memory access time = 100ns

TLB access time = 15ns

P.T.E at 1st level = 2Byte

4 2nd level = 4Byte



Page table divide into 512 pages

4 each page size = 1K.

1st level PT size + 2nd level P.T size

$\Rightarrow 2^{P_1} * PTE + 2^{P_2} * PTE$

$\Rightarrow 2^9 * 2B + 2^{10} * 4B$

$\Rightarrow 1KB + 4KB$

$\Rightarrow$ 5KB

33 TLB = 95% (hit ratio)

link to above question

EMAT = .95(15ns + 100ns) + .05(15ns + 3×100ns)

⇒ .95 * 115ns + 0.05 * 315ns

⇒ 109.25 + 15.75

125ns

34

L.A = 32bit

Page Size = 4KB

P.A = 29bit

Page fault rate = 2%

Page fault Service = 20 millisec

& memory access time is: 2 microsec.

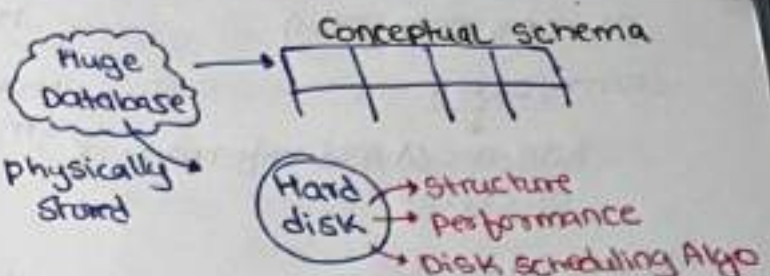
P.TE contain
1 valid/invalid bit
1 reference bit
1 modified bit
3bit for page for protection

if PTE is multiple of 3Byte

approx. PT size &

EMAT

• Magnetic Disk



CPU ---→ Closest memory to CPU is cache memory

every action/execution is done by CPU

CPU has it's own memory (register) (fastest)

Primary memory

(Secondary/permanent memory) (non-volatile Storage)

Slowness increasing & capacity increasing.

If power goes Still Data Remain intact

main memory } physical Devices

DISK

mostly we study it in logical way b/c every cell has same access time

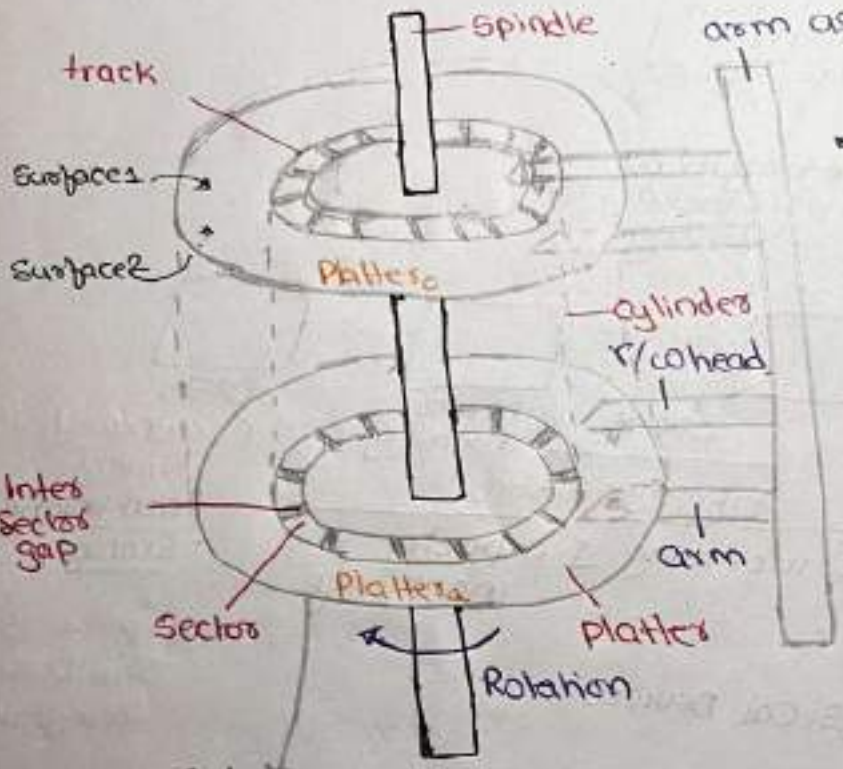
we care about physical Structure of DISK because Different Sectors have Different Access time

main memory (RAM)

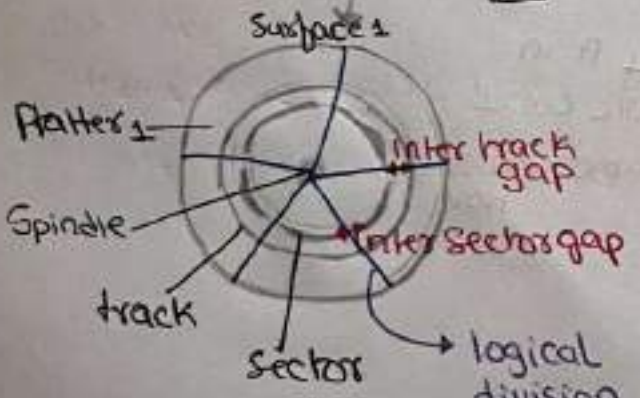
- SRAM, DRAM
  - bits are stored in form of charge
- Random access

Magnetic Disk

- bit are stored in form of magnetic poles
  - N
  - S
- Random Access
- Smallest unit in DISK is Sector for Read & write



- all the head will move simultaneously but only one of them will Read or write at any point of time
- outermost cylinder (cylinder -0)



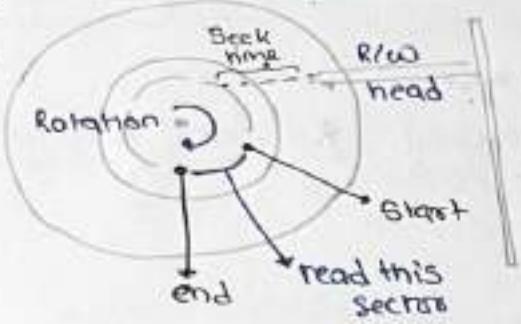
does head touch the surface?

- no, there is a gap (micro gap) otherwise there will be fire it will (corrupt it)
- R/W is happening with the help of magnetism.

- Reason for gap

① due to magnetism if u don't keep the gap then data will corrupt
② there can be miss alignment of there is no gap

} gap is created using magnetism



① Putting R/w head on Desired track
Seek time

② Rotate starting point of Desired Sector under R/w head.
Rotational latency

③ Rotate complete Desired sector under R/w head: Transfer time

- every sector will contain same amount of data (in every track/platter)
- If you are close to to spindle then you will have a small sector gap, & if far from spindle then you will have more sector
[waste of space but easy to manage]

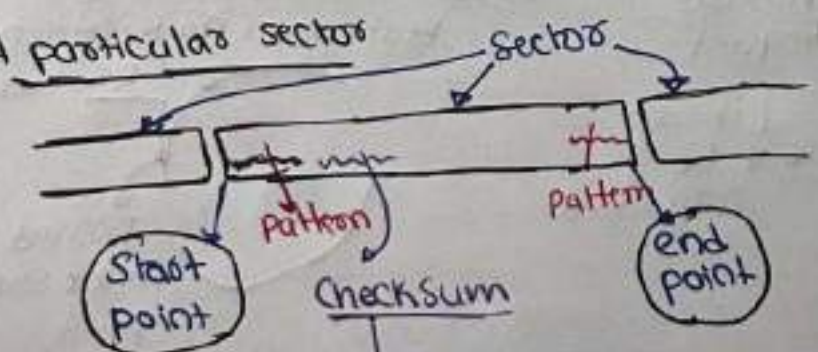
Block → Some consecutive sectors (not a property of disk)
(property of DBMS)
↳ will decides how many consecutive sector to be considered as one block.

Sector Size → property of Disk (can't change)

Size of disk block → Can be set when disk is initialized as a multiple of the sector size.

Cylinder → A cylinder contains one track per platter surface

- A particular sector



Checksum
↳ To check whether Sector Data is correct or not

- In any sector, Apart from the Actual Data, every extra info is called Disk formatting (or) formatted data.

• Capacity of a Disk

ex: How much data can we save on a disk

platter × 2 (Recording Surface) × #tracks/Surface × #Sector/Track × sector size

Surface

#cylinder: #track per surface

• Disk performance parameters

• access time for particular sector.

Seek time + Rotational latency + transfer time.

• Disk access time

Seek time + Rotational latency

• Throughput or Data transfer Rate.

Data per second can we read/write

(Byte)

ex:

• 1000 cylinders, 10 sector/track

• Head Assembly At cylinders 0. Initially

• Head move 10MS/cylinder

• Disk rotate 100 times/second

• Controller, Bus perfect, no previous Request

Avg. Time to Read A Randomly chosen Byte is. } it can be anywhere

① Seek time

↳ Avg. seek time

0 1 2 999

| | | ..... |

Initially Avg. = 500

So, 500 × 10MS = 5ms

② Rotational latency

Best case: no Rotation

• Worst case : one whole Rotation Avg. case : half rotation

Rotational latency
↳ Avg. RL : $\frac{1}{2}$ (time taken for one Rotation)

$\Rightarrow \frac{1}{2} \times 10ms$

1 Sec ⟶ 100 Rotation

1 Rotation ⟶ $\frac{1}{100}$ sec $= \frac{10^{-3}}{10^{-3} \times 100} =$ 10ms

Avg. RL = 5ms

③ transfer time (one sector)

given: 10 sectors per track

Read one track ⟶ 1 Rotation: 10ms

Read 10 sectors ⟶ 10ms

Read 1 sector ⟶ 1ms

Read Random byte:

Total time: Avg. Seek time + Avg. RL + Transfer time

$\Rightarrow$ 5ms + 5ms + 1ms $\Rightarrow$ 11ms

note

• we consider Avg. Seek time usually / Avg. RL

• given in question that Seek time it means Avg. Seek time

• assume Avg. time to Read A Byte on cylinder '5' is

Seek time: $5 \times 10ms = 50ms = 0.05ms$

Avg. Rotational latency: $\frac{10ms}{2} = 5ms$

Transfer time: 1ms per sector

Total time: 6ms + 0.05ms = 6.05ms

ex:
Sector size: 512B
2000 track/surface
50 sectors/track
5 double sided platter
avg. seek time 10ms

(1) (i) track capacity in Byte
sector/track × sector size
⇒ 50 × 512B
⇒ 25,600B

(ii) surface capacity
tracks/surface × Track size
⇒ 2000 × 50 × 512B

(iii) DISK capacity
surface capacity × # surface
⇒ 2000 × 50 × 512 × 10 B

(2) no. of cylinders:
track/surface
⇒ 2000

(3) give ex. of valid B.S. is 256 valid B.S? 2048?

B.S = n × sector size
{n ∈ Natural no.}

so, n × 512B
no value × 512 = 256 ✗
4 × 512 = 2048 ✓

(4) max. Rotational delay
in disk platter rotate 5400 rpm

Rotation speed: 5400 RPM
RPM: Rotation → minute

60sec ⟶ 5400 Rotations

$\frac{60}{5400}$ sec ⟵ 1Rotation

1Rotation ⟶ $\frac{60}{5400}$ sec × $\frac{10^3}{10^{-3}}$ ms

⟶ $\frac{60\,000}{5400}$ ms

⟶ 11.11 ms.

Aug. Rotal. Delay
⇒ $\frac{(\frac{600}{54})}{2}$ ms

don't do these ✗
bcz these Aprox. ← $\left[\frac{11.11}{2}\right]$

⑤ one track of data can be transferred per revolution | then const is transfer rate

#Byte/sec

1 Rotation → $\frac{60}{5400}$ ms → one track Data → 50 × 512B

in $\frac{60}{5400}$ ms → 50 × 512B

⇒ 1 sec → $\frac{50 \times 512 \times 5400}{60 \times 10^{-3}}$

⇒ 2250 KB/sec

ex:
6 platters,
advertised avg. Seek time: 4ms
rotation speed: 15,000 rpm
512B sector with
500 sector / track
track to track to access time is 1ms
read a file consisting of 2500 sector. for a 1.28MB.
total time for the transfer.

Total time: 4ms + 5×2ms + 5× one Rotation time

⇒ 4+10+5×4ms

⇒ 34ms

---

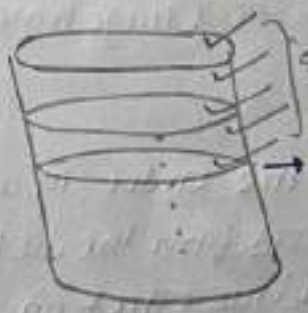
1 Rotation → $\frac{60\,sec}{15000} = \frac{60000}{15000}$ ms

⇒ 4ms

① Assume that the file is stored consecutively on a cylinder (5 surface)

500 sector per track

file: 2500 sector ⇒ in a single cylinder.



→ #sectors per Cylinder = 12×500 = 6000 sector.

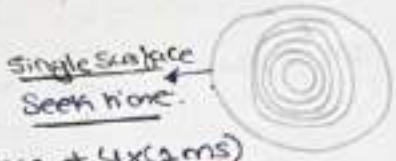
Total time: Seek time once + 5×Avg. RL + 5× one Track Transfer time

b/c once you fix R/w head then consecutively it will be fix on other platter too

So you don't have to move it again

ex. Assume that the file is stored consecutively on a (5 track surface

file: 2500 sectors; 500 sectors/track



Total time: Single Surface

⇒ 8ms + 5×Avg. RL + 5×Transf. time

⇒ 8ms + 5×2 + 5×4

⇒ 38ms

4ms + 4×(2ms) (Track to track)

1st seek time will be Avg.

⇒ 8ms

ex. Assume that the file is stored randomly in sectors throughout the disk

file → 2500 sectors → Random sectors

⇒ 2500 × [4ms (seek time) + 2ms (R-L) + 4ms/500] ⇒ 2500 × 6.008ms

⇒ 15.002sec

⇒ 15.000sec. (Huge time)

- It is clear that the order in which sectors are read from the disk has a tremendous effect on I/O performance

- Closeness of sectors

Two sectors: Decreasing order of closeness

1. Same track → Single Rotation
2. Same cylinder → Single seek
3. Adjacent cylinder → Seek time less

Data is stored:

① single sector
② single track
③ single cylinder
④ next cylinder

Sector Addressing

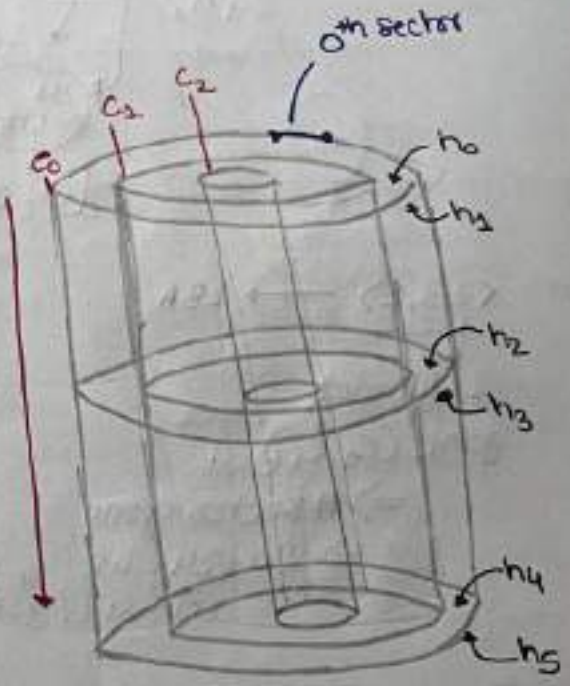
two type

① LBA (Logical block addressing)
↳ Integer

② CHS (cylinders-head-sector)
<C,H,S>

By default Track no. start at 0,
track 0 is the outermost
& highest track is nxt to spindle

ex: 3 platters
6 Recording surface
3 cylinders,
4 sector/track
Sector will have Address



#surface = #heads

Start count from 0:

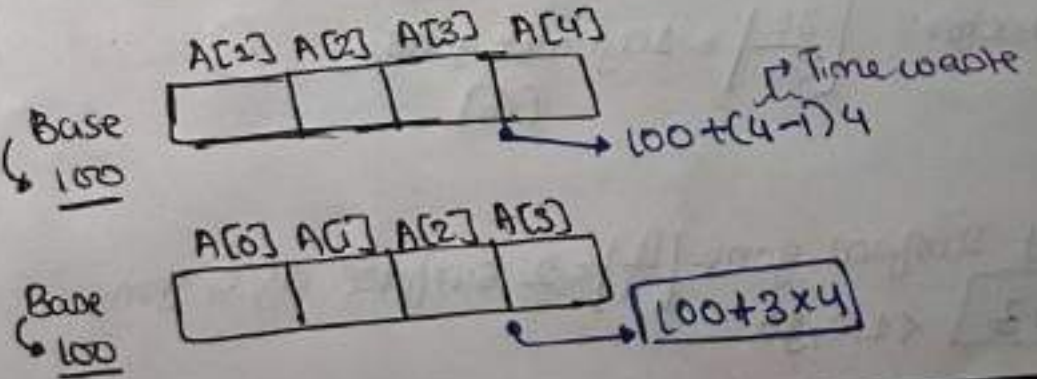
Benefits:

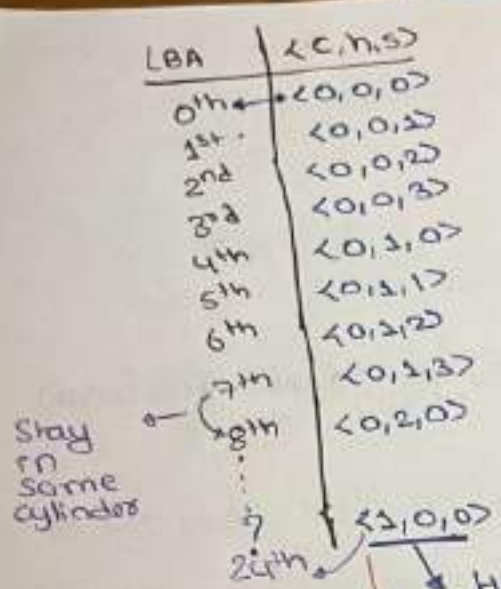
0
1
2
3
(4) → before it How many? = 4 no.
5

how many cyclinder

0
1
2
(3) → 3 cyclinders
4

so here we don't have to waste time on subtraction





when count starts from 0

- (y) no. of Hulo things have gone then you are at (hulu y)
- if you are at hulu y then y hulu have gone

ex: y no. of sectors things have gone then you are at sector y

- if you are at sector y then y sectors have gone

How many cylinders gone? = 1 → C_0

How many sector gone ⇒ 6×4=24

24 Sectors

[0, 23] gone

ex: <2,3,2> → LBA

2 cylinder gone (C_0, C_1)

⇒ # sector gone

⇒ 24+24=48 sector (0-47) gone

3 surface gone in S_2

⇒ # sector gone

3×4=12 sector gone

2 sector gone (0,1)

Total how many sector gone ⇒ 48+12+2

⇒ 62th sector

[0-61] gone

ex: LBA: 35th sector → <C,h,s>?

1 cylinder: 24 sectors $\left\lfloor \frac{35}{24} \right\rfloor$ = 1 cylinder gone. [C_0]

I am at (C_1)

In C_1 how many surface gone: $\left\lfloor \frac{11}{4} \right\rfloor$ = 2 surface of C_1 gone

I am at $C_1, S_2, 3$ <1,2,3>

formula: only if counting start from 0th

<C,h,S> ⟶ LBA?

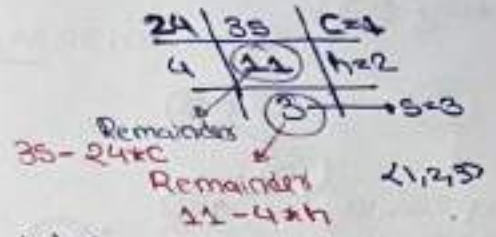
#sector per cylinder = n_c

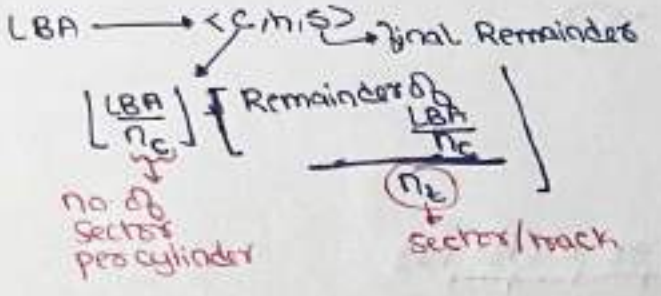
#sector per track = n_t

<C,h,S> = $C n_c + h n_t + S$

LBA ⟶ <C,h,S>?

35th sec





- usually (in Harddrive) head movement is slower than Rotation Speed.

Seek time ≫ Rotational latency

min. the overall Seek time

ex: when we have sector R/w Requests from many process, what matters most to improve performance?

(A) Total seek time

(B) Total Rotational latency

(C) Actual data transfer time

are almost same

Depend on speed of Rotation

1 Rotation ⟶ Transfer one track

ex. Sector R/w Request from many process. what matter most to improve performance?

(A) their cylinder no.

(B) their surface no.

(C) it is Read Request or it is write request.

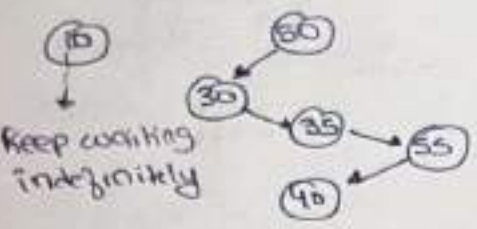
- Disk scheduling Algo. only want cylinder
- But process want to access sector.

• problem in SSTF : starvation possible

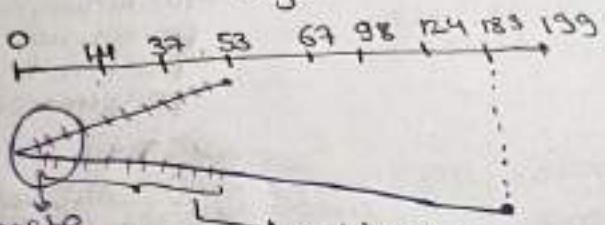
Disk queue: request will keep coming

Initially 50

Queue: 10, 30, 35, 55, 40, ... keep coming



• Disadvantage of Scan/C Algo.



Problem 1: waste of time

So for these prblm solution is that solve using look

problem 2: not uniform serving

Some are sense choice.

Solution is C-Scan.

ex: which Scheduling Algo. is best.

FCFS < (SSTF, SCAN, C-Scan, LOOK, C-LOOK)

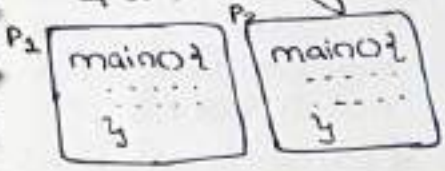
non-Comparable.

Implement as → look better than Scan

Implement as → C-look better than C-Scan

Memory Management

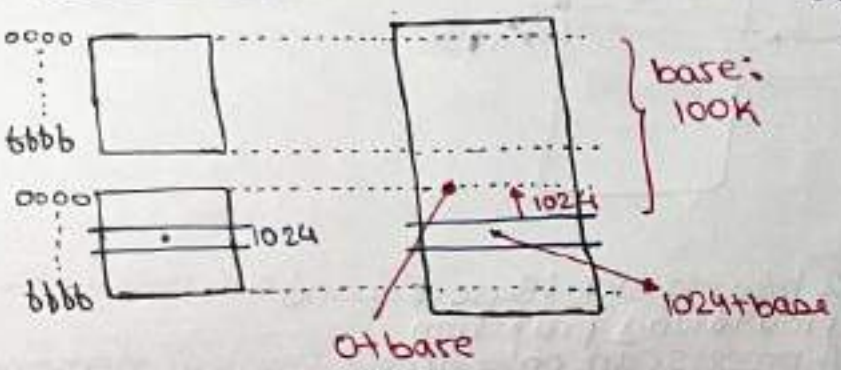
Prblm:
we have 2 program & one memory



if compiler assume that any program start address is '0' then we can't have 2 program at same time in M.M

note
- untill the word virtual memory come's we are assuming that whole process is in M.M
- In the current model the process size is fixed

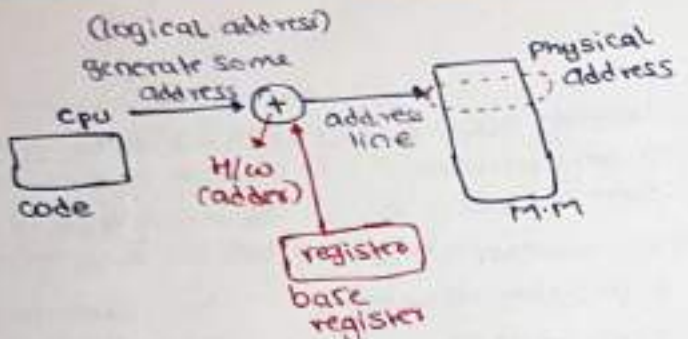
Save / store



- compiler Shouldn't generate the physical address; whatever it generate we will relocate it
  ↓
  one way to achieve this is to relocate program at different address.

Point to rem

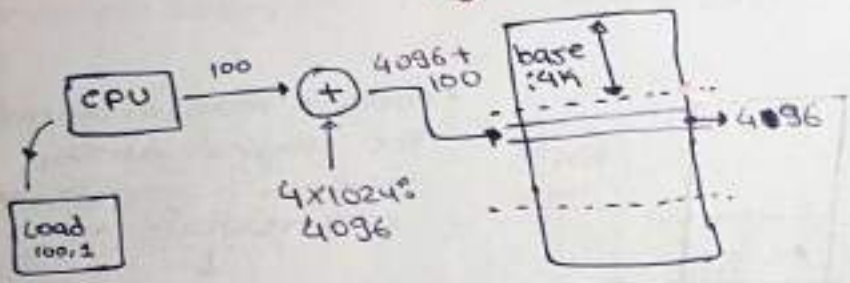
- process is contigiously stored in M.M
- process start from "base" address ..... decide by O.S (Allocation method)
- each process has it's own base address
- Base address is maintained in H/w register called Base register
- we can store process Base address into PCB
- at time of context switch we will (just like other register) load base register value corresponding to the particular process.



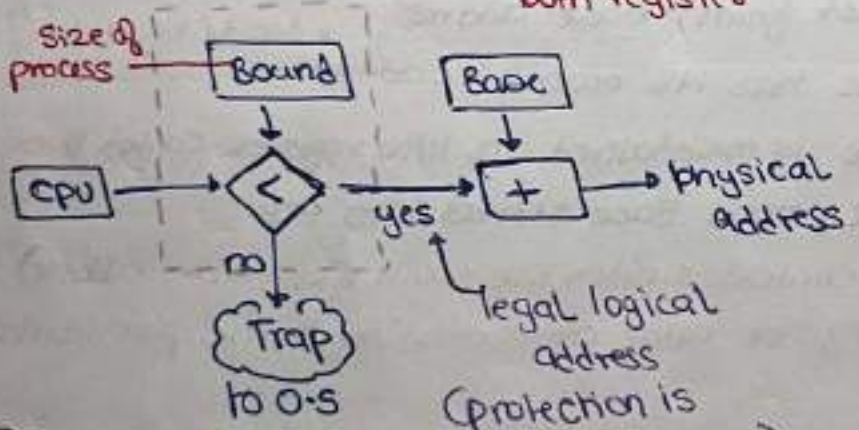
Prblm in current model

- in these model we are access the memory that we should not out of bound. [so we should bound these]

(process)



- Base & Bound (limit register)
  unable to provide protection So we need "Bound"
- 2 h/w register : Base & bound
  + providing protection
- A process can only access physical memory in [base, base+bound)
  process size — not included
- on Context switch we need to save/restore both register

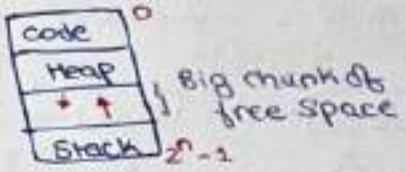


Advantages

- Provide protection
- Support dynamic relocation
- Simple / inexpensive / fast H/w implem.
- Simple context switch logic.

dis advantage

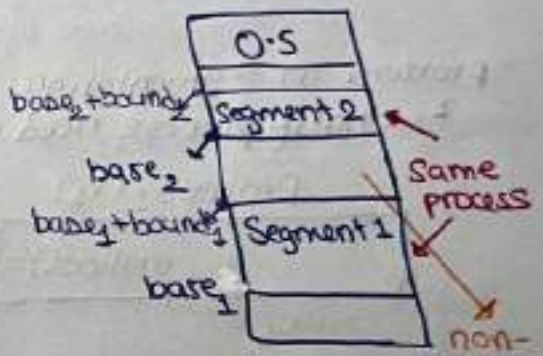
- each process must be allocated contiguously in physical memory.



Segmentation

idea
we will divide into some chunk & store chunks independently in M·M
(we are not saying hard disk) currently

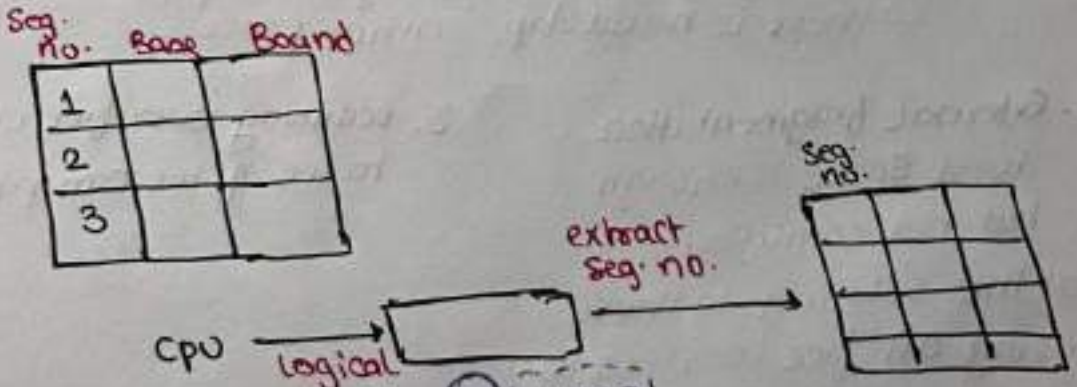
- point's
- Split each process logical address space into multiple segment & store in M·M at non contigious location
- variable sized area
- Base & Bound: 1 segment per process
  segmentation: many segment per process



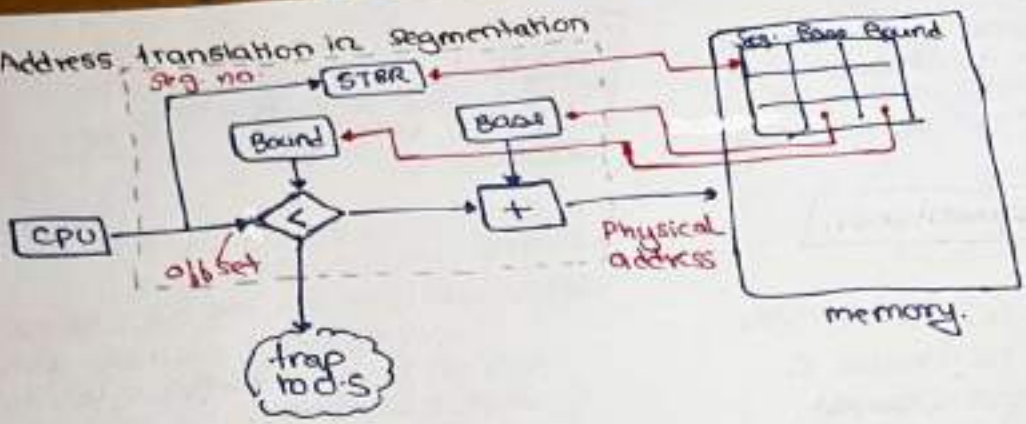
- Segment map (store in M·M)
  → one per process
  → hold base & bound registers permission for each segment

- Context switch
  → save / restore table / pointer to table in Kernal memory

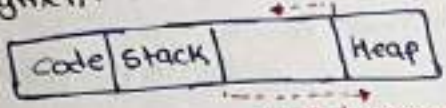
STBR (segment table Base register) : for every process value will be different



- Address translation in segmentation

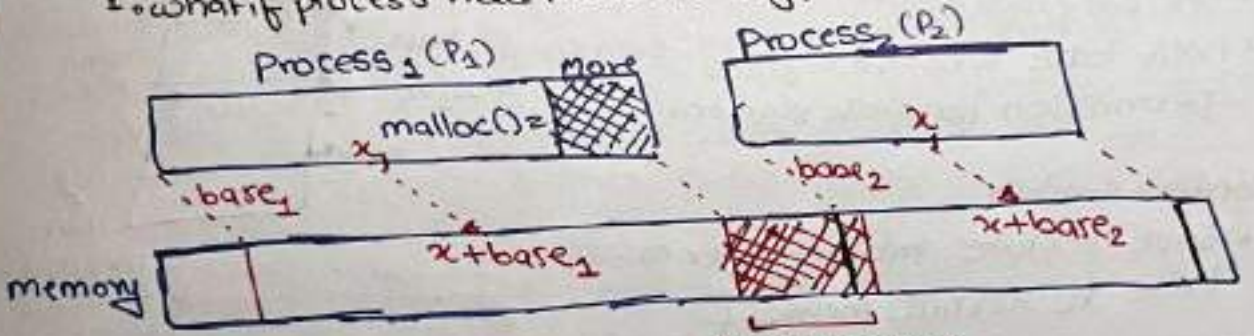


- Segment size are fixed



at runtime the size of segment may change than what we initially thought

- problem in segmentation:
  1. what if process need more memory?



due to malloc heap want more & due to which it may overlap to other process

Soluⁿ can be that you can either move P_2 forward or P_1 backward a bit so there is no overlap

2. External fragmentation.
   total space is enough but not contigious.

one of the solution to these is moving all free location to one side & allocate to one site but it's just overhead (wasting time) (Know as compaction)

3. scanning over free chunks to see if we can put segment here

Summary so far

→ only base reg.
- No protection
- Cont. memory needed
- external frag

→ Base + Bound
↳ protection
- Cont. memory needed
- external frag.

→ Segmentation (multiple base & bound)
- protection
- cont. mem. needed
- external frag.

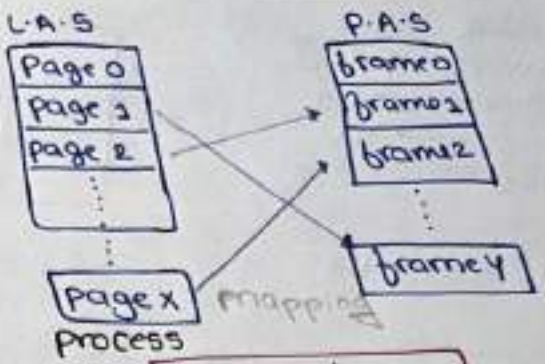
→ Seg. size is fix but heap/stack are not fixed.

- what if we keep all the segment to be equal size.

- paging doesn't have external fragmentation
(why so imp. tous ??)
↳ It is wastage of space or time
↓
Relocation of free partition will save space but will waste time

Paging

- divide content into fixed-sized pages.
  Solve prblm of external fragmentation
- Solve the internal fragmentn problem by making the unit's small

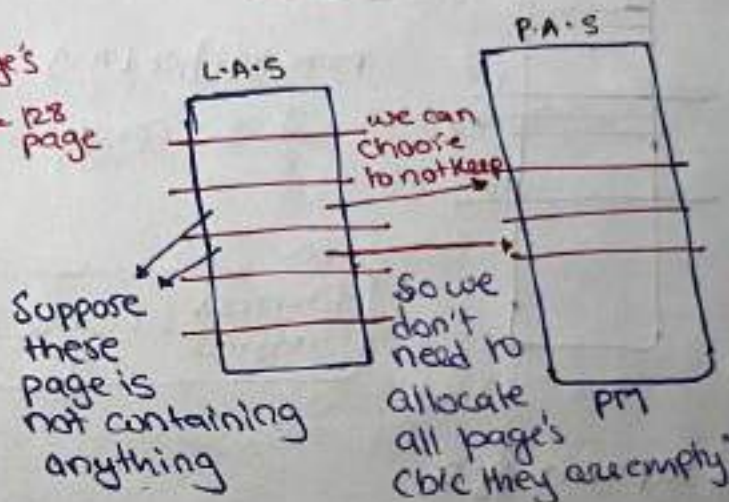


Page size = frame size

ex: 0 ..... 1023 Page size: 8

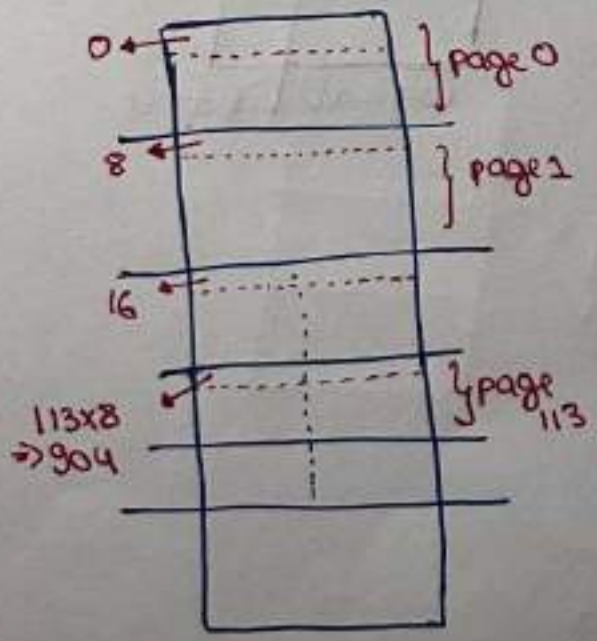
no. of page's
$\Rightarrow \frac{1024}{8} = 128$ page

- page table is created/maintained by O·S



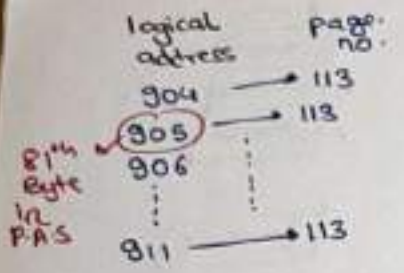
ex: Suppose
L·A·S : 1024B
P·S : 8B
Pagetable is given
Logical → physical
gos → ??

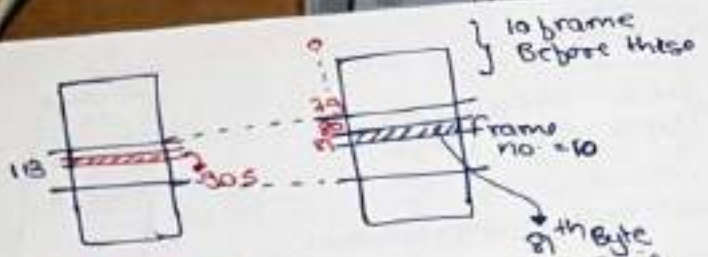
pageno.	frameno
0	3
1	6
⋮	
113	10
127	



CPU says gos
logical → $\frac{gos}{8} = 113 \cdot xx$
page size

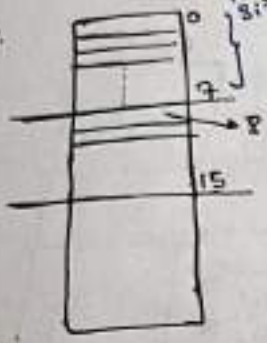
so. pageno. = 113





one frame consist of 8 Byte
So, 10 frame consist of 10×8
⇒ 80 Byte
[0 to 79]

ex:



page no. for L.A: 9

$\frac{9}{8} \Rightarrow 1$ (page no)

offset for LA = 9

9 % 8 ⇒ (1) offset

So,

Physical address : Frame no. × frame size + Offset

- size to power of 2

$1KB = 2^{10}B \rightarrow 10^3$
$1MB = 2^{20}B \rightarrow 10^6$
$1GB = 2^{30}B \rightarrow 10^9$
$1TB = 2^{40}B$

} when we measure data transfer Speed (in CN course)

Binary

807: (1100100)111

must be 100 (page no.)

offset

Decimal

Page no. × 2^3 + offset

append 3 zero and add offset

ex: page table

Virtual page no.	Frame no.
0	3
1	10
2	9
3	2
4	0

Page Size = 1024B

• convert virtual address into physical address

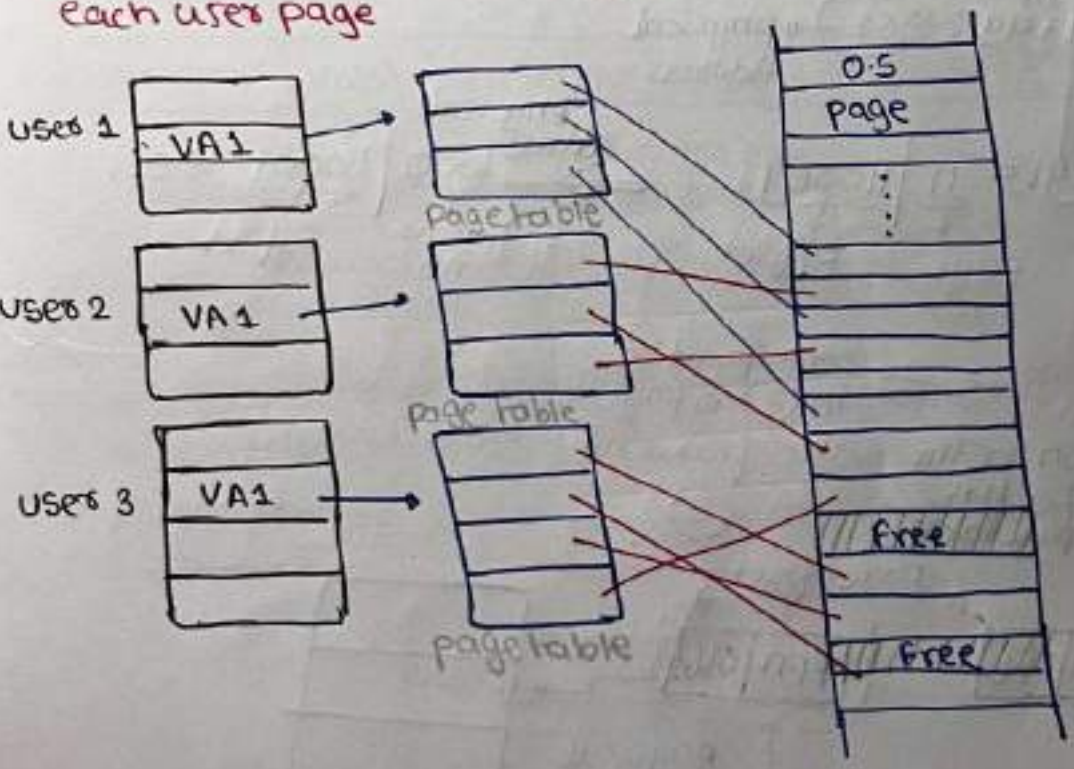
(a) 697 page no.
$\frac{697}{1024} = 0.xx$
real address:
3*1024+697 ⇒ 3769
(frame no.)

(b) 1054
$\frac{1054}{1024} = 1.xx$ real address
1*1024+30 = 10270
(frame no. corresponding to page no.)

(c) given physical address : 2075 convert to virtual address
So, $\frac{2075}{1024} = 2.xx$ So, frame no. 2 & offset 27
2075 % 1024
& frame 2 is mapped to virtual page 3
So, virtual address : 3*1024+27 = 3099

Point to remember

- Each user has a page table
- page table contain an entry for each user page



Page translation exercise

ex: V.A: 8bit, P.A: 10bit

P.S: 64B
↳ 2^6 Byte
⇒ 6bit

V.A: 8bit

2bit (page no)	6bit

P.A: 10bit

4bit (frame no)	6bit

(a) no. of virtual page? 2^2: 4 page (00, 01, 10, 11)

(b) no. of physical page? 2^4: 16

(c) no. of entries in page table: one entry for each page

So, 2^2 = 4 page

(d) given page table [2,5,1,8]
what's the physical address for virtual address 241?

- Decimal: $\frac{241}{64} = 3$

Offset: 241 % 64 = 49

So, 3 * 64 + 49

3 = page no., 49 = offset

replace by frame no.

8 * 64 + 49 = 561 → physical address

Page table

0	2
1	5
2	1
3	8

- Binary: 241 = 11 | 110001

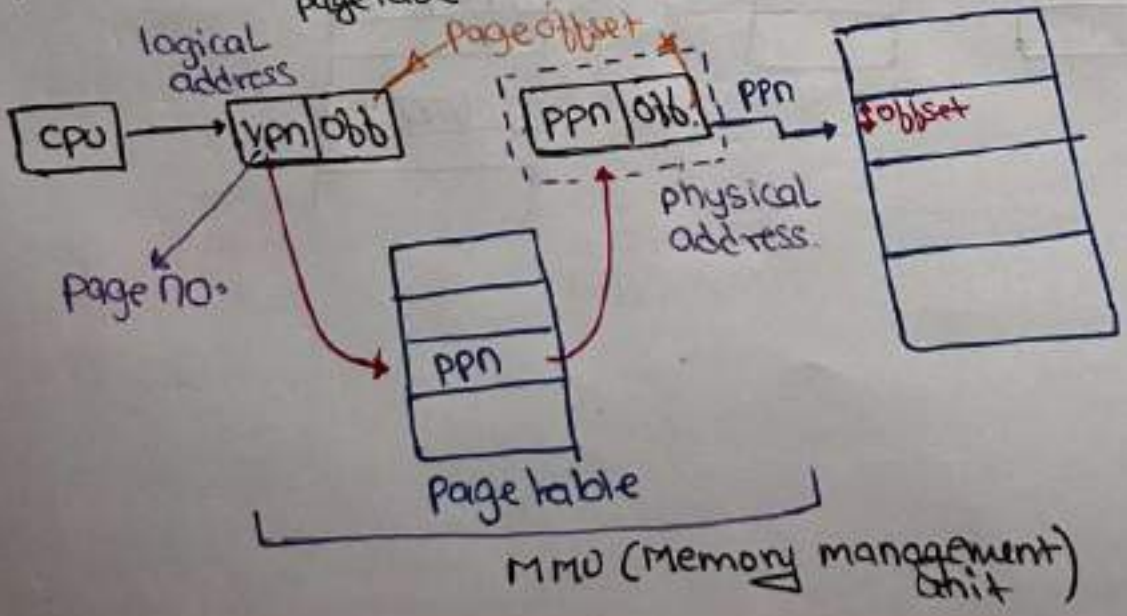
11 = page no., 110001 = offset

⇒ physical address: 1000 | 110001 ⇒ 561

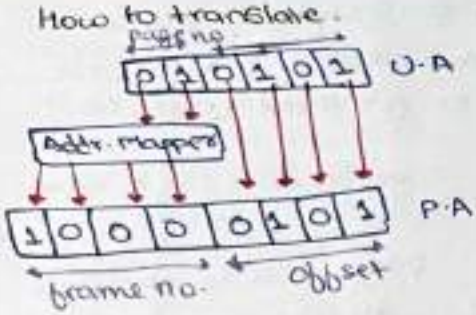
1000 = frame no., 110001 = offset

So, page no. = 3 ⇒ corresponding to (3) as page no. the frame no. is (8)

- Address translation in the hardware (MMU) for flat page table



How to translate.



- other PTE info
- valid bit
- protection bit
- present bit
- Referenced bit
- Dirty bit

Ex: PA: 4 6 LA 2 6

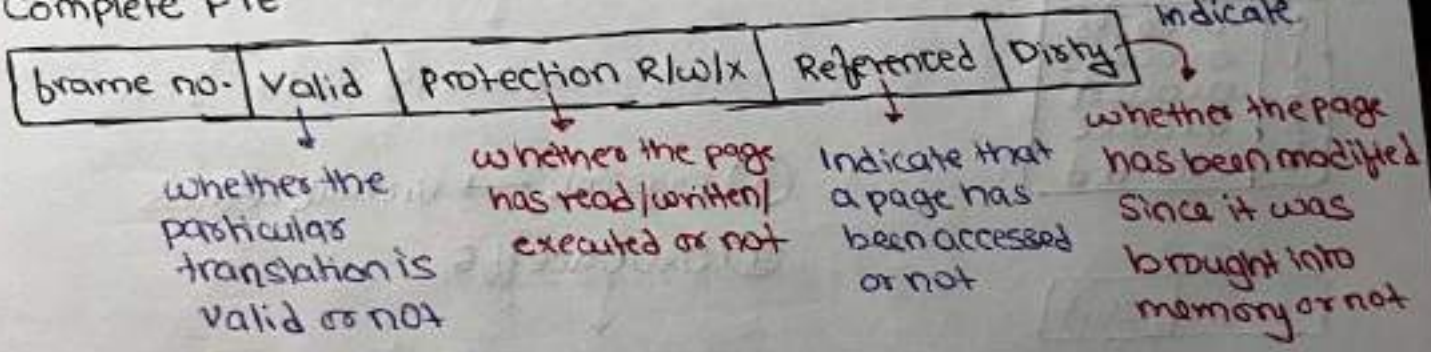
no. of entries in page table: $2^2 = 4$

Size of the page table: Size of one entry × no. of entry

(given) P.T.E Size: 2B ⇒ 4 × 4 = 16 bit = 2B

Page table Size: 2B × 4 = 8B

Complete PTE

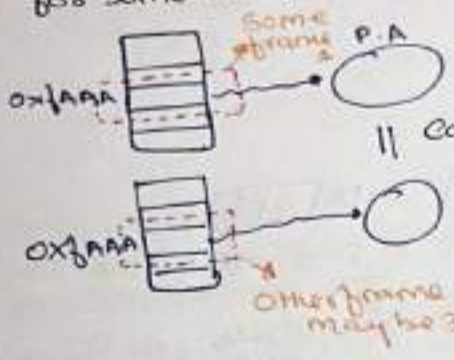


- present bit: whether this page is in physical memory or on disk or not (swapped out)

Synonyms:

- Valid bit == present bit
- Dirty bit == Modified bit
- Referenced bit == Accessed bit

- Suppose there are multiple process in paging system. Suppose no process is sharing any data with other process. Is it possible for some V.A of 2 different process we get same frame no.?



|| Can these 2 be same?

even though LA is same STILL PA is different b/c page will be mapped to different frames

ex: virtual memory with paging

L.A = P.A = 32bit

P.S: 4KB

Page table

Page no.	L.A	frame no.
	0	0x00788
	1	0x00249
	2	0x0023f
	3	0x00ace
	4	0x00bcd

32bit

20bit	12bit

(i) Virtual address → physical address.

(a) 0x00001a60

20bit 12bit

↓

0x00249 : a60 } P.A

(ii) physical → virtual address

(a) 0x00ace:bb6

↓

0x00003:bb6 } L.A

ex: V.A(L.A): 1GB

P.S: 1KB: 2^{10}B

PTE hold valid & page-frame no.

max of 2^{15} physical page

(32MB physical memory can be addressed at most)

30bit

VA: | 20 | 10 |

PA | 15 | 10 |

PTE | frame no. | valid |

Page table size: $2^{20} \times (15+1)$ bit

$\Rightarrow 2^{20} \times 2B = 2^{21}B$

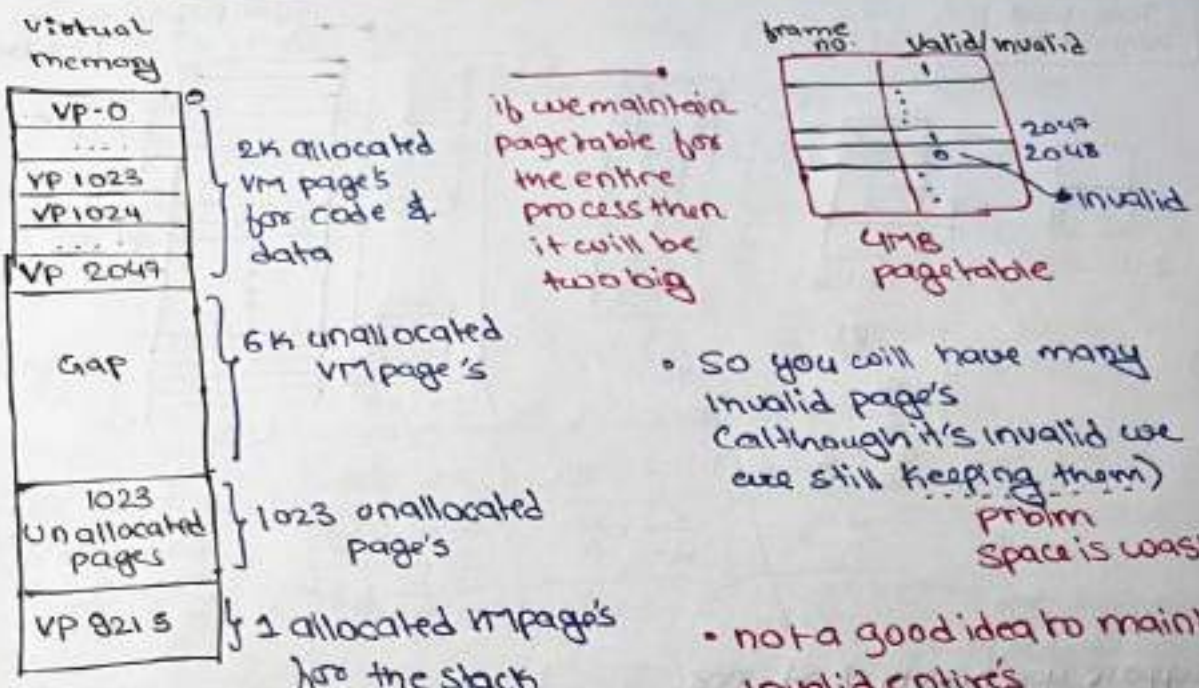
$\Rightarrow$ 2MB for 1 processes

How much memory is used for page table, when there are 100 processes running in the system

if 100 processes,

2MB × 100

$\Rightarrow$ 200MB



- So you will have many invalid page's (although it's invalid we are still keeping them)
  Prblm
  Space is wasted
- not a good idea to maintain invalid entires

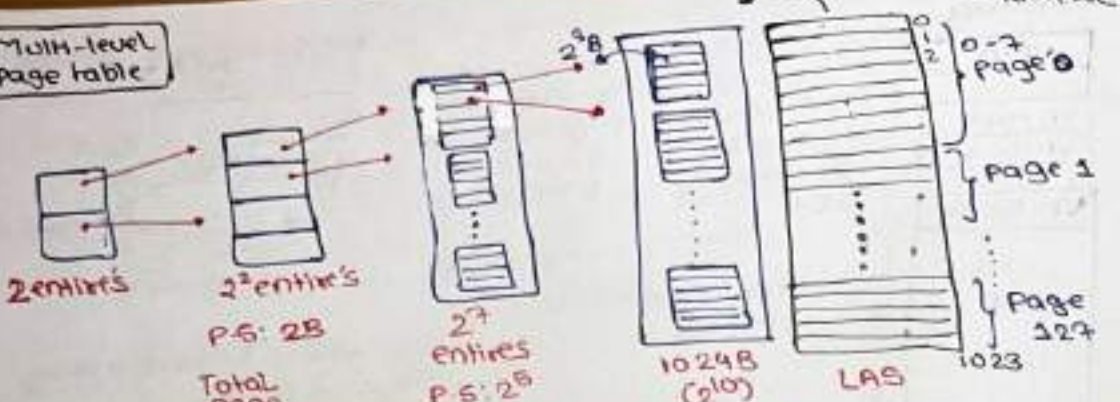
- why are page table so large?
  Ceven though the process is small (35B)

  But page table: 4MB
  Size
  B/c we have 2 many invalid page's to maintain

Prblm
How to reduce the size of page table?

Soln
Multi-level page tables

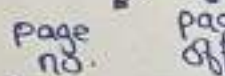
Multi-level page table



we are taking page size: 2^3B (Chunk size)

ex: suppose you need to find 803.

then:

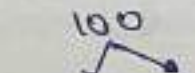


page offset: 803 % 8 : 3

$\frac{803}{2^3} = 100.xx$

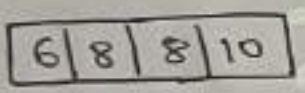
So, P.A: $100 * 2^3 + 3$

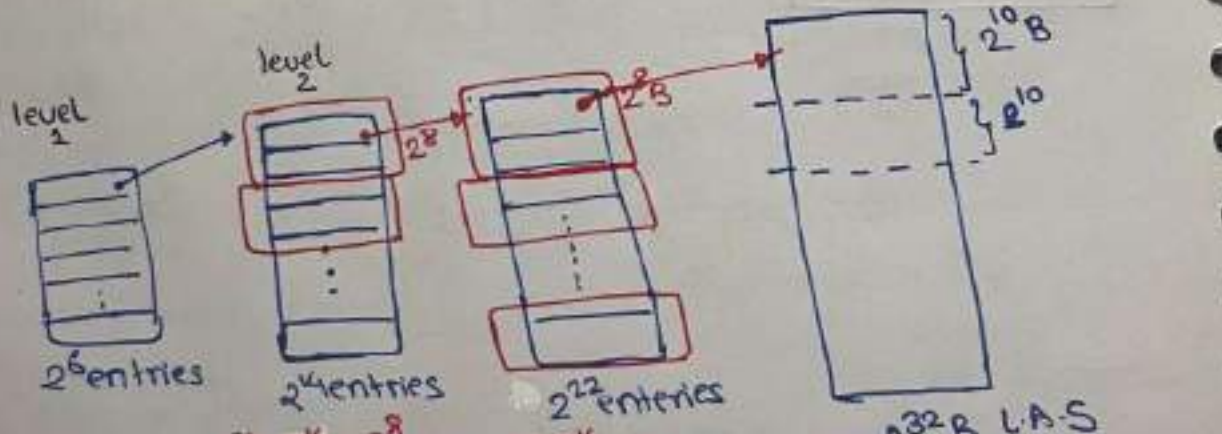
$$\begin{cases} 803 = 100 \times 2^3 + 3 \\ 803 = (3 \times 2^5 + 4) \times 2^3 + 3 \end{cases}$$



$\frac{100}{2^5} = 3.xx$ $\quad 100 \% 2^5 = 4$

ex: 32bit L.A





Chunk size: 2^8 entries

no. of entries: $\frac{2^{14}}{2^8} = 2^6$

Chunk size: 2^8

no. entries: $\frac{2^{22}}{2^8} = 2^{14}$

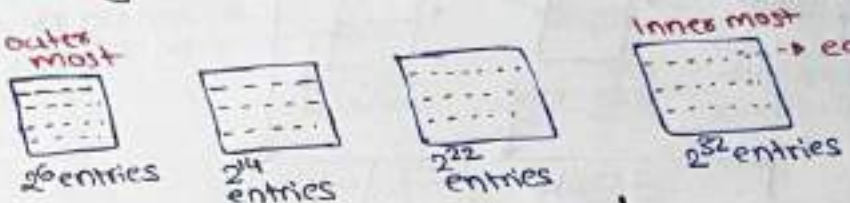
Chunk size: 2^{10}B

no. entries: $\frac{2^{32}}{2^{10}} = 2^{22}$ pages.

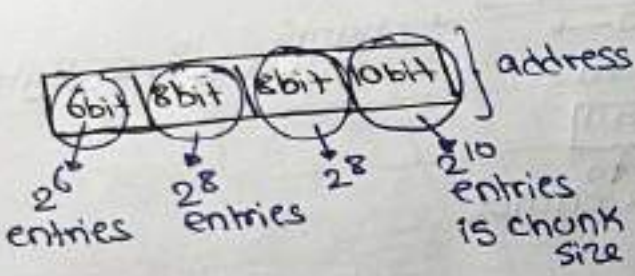
2^6 entries, every entry point to one chunk of size 2^8 chunk (pages)

how many no of chunks in level 2: $2^6 \times 1 = 2^6$

how many no of entries ..... : $2^6 * 2^8 = 2^{14}$ entries



# of Chunks	1	2^6	2^{14}	2^{22}
Chunk size	2^6	2^8	2^8	2^{10} → each entry just contain 1B
# of entries	2^6	$2^6 \times 2^6 = 2^{14}$	$2^{14} \times 2^8 = 2^{22}$	$2^{22} \times 2^{10} = 2^{32}$ entries. 2^{32} Byte



outer most: left most

Inner most: Right most

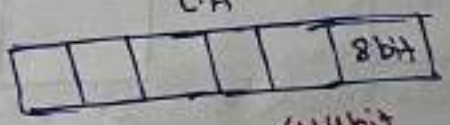
ex: Suppose we have

VA: 32bit

Value of address: 0x12345678

& we are using 5level page table at the Inner most page table the frame no. is 0x0134 then calculate P.A

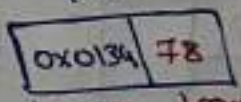
If size of page is 2^8B → 8bit

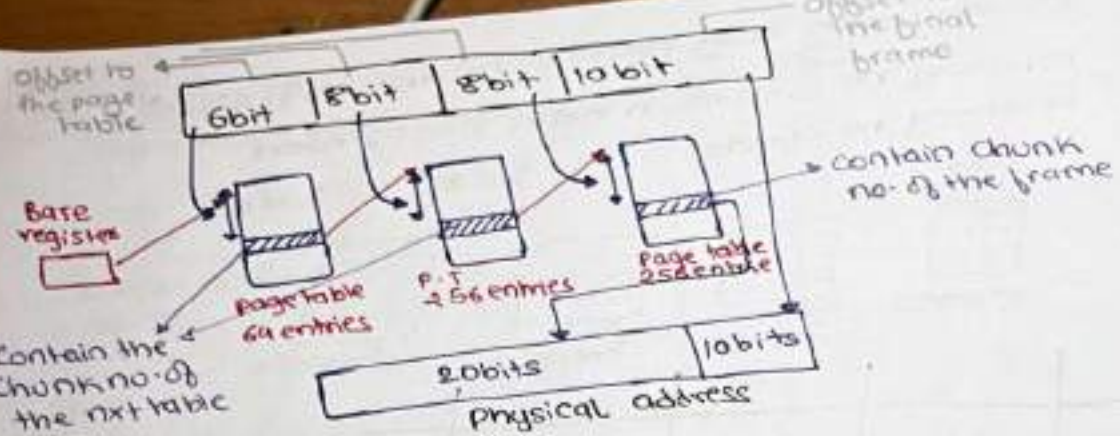


24+4bit

0x12345678

no. of page. | page size



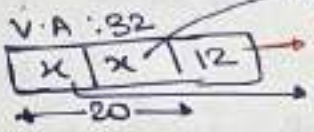


gate CSE 2002.

ex: V.A: 32bit
P.A: 32bit
P.S: 4KB
It use's 2 level paging
equal no. of bit is used in 1st & 2nd level
PTE: 4B.

P.S: 4KB ⇒ 2^{12}B
12bit

Chunk size = $2^{10} \times 2^2$ Byte



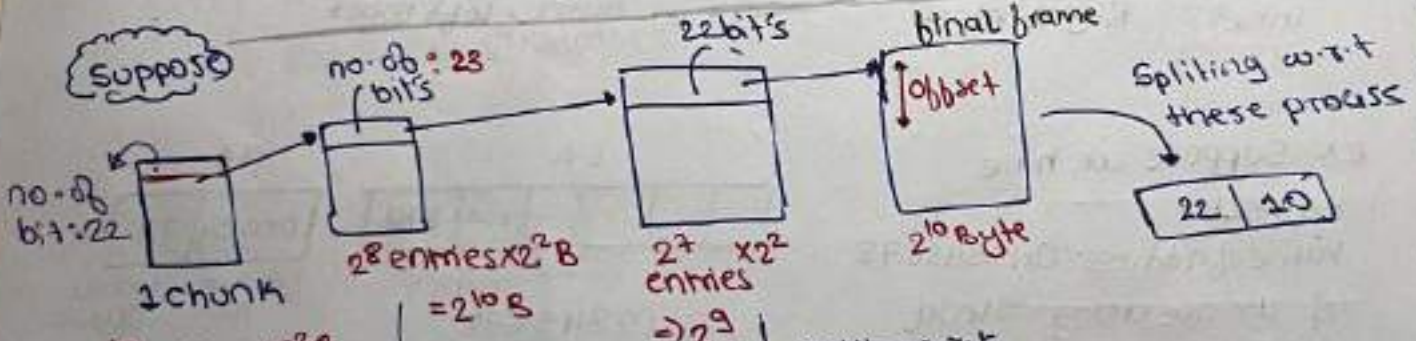
Chunk size : 2^{12} B

Chunk Size : $2^{10} \times 2^2$ Byte

$2^x = 20$

x210

to 2 entries

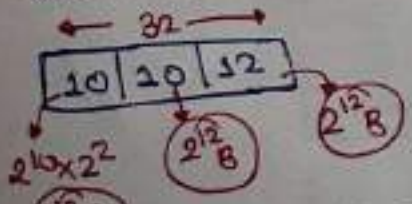


Split w.r.t these process

P.A 32

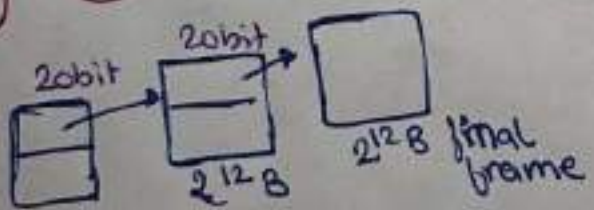
23 | 9

Back to question



Split w.r.t to these process

22 | 10



P.A | 20 | 12 |

gate 2008

* P.A: 36bit
  L.A: 32bit

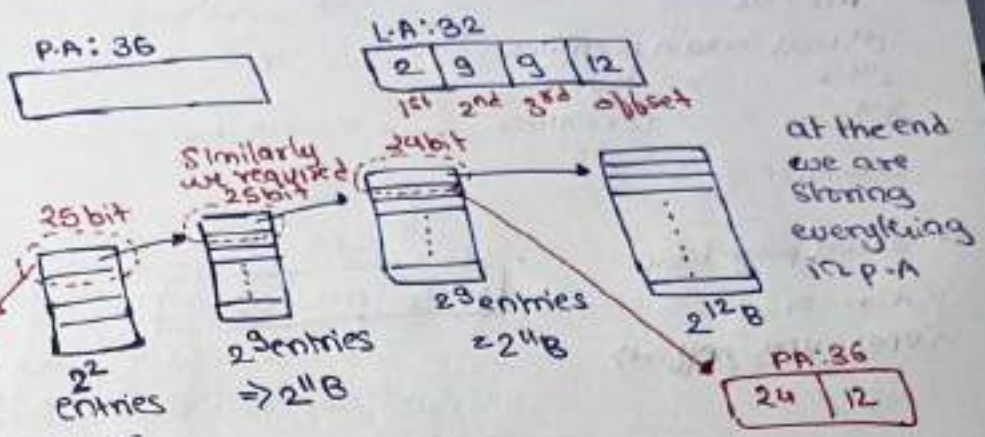
P-S: 4KB

PTE: 4B

3 level paging

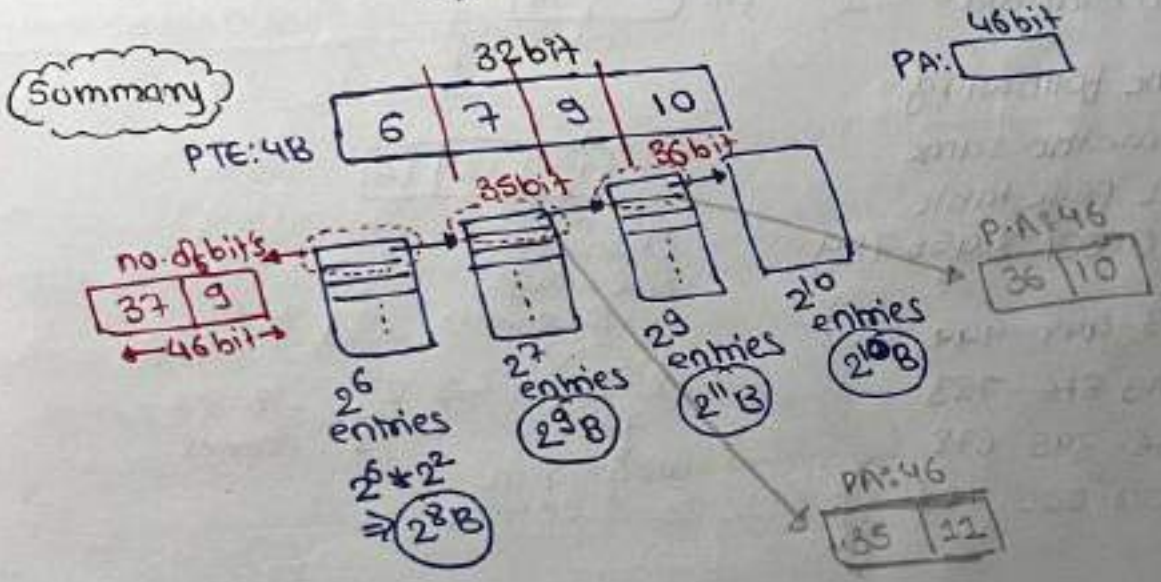
	bits
1st level	30-31
2nd level	21-29
3rd level	12-20
offset	0-11

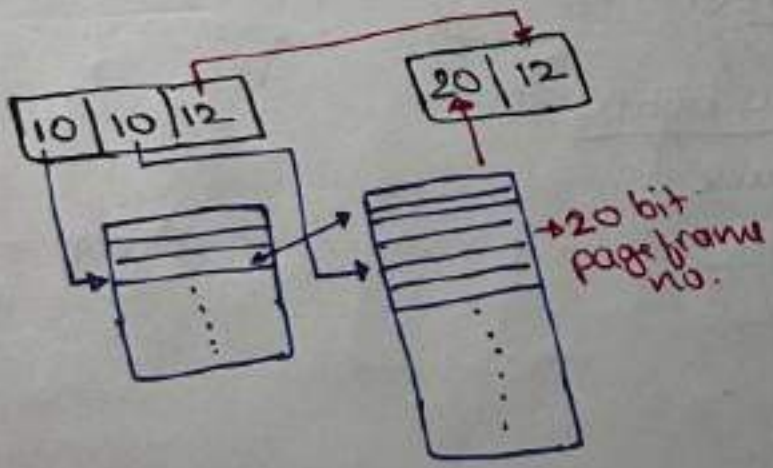
the no. of bits required for addressing the next level page table the PTE of 1st, 2nd & 3rd.



25, 25, 24

P.A: 36 | 32 | 4 → L.A





ex: 3level page table

P.S: 256B

PTE: 2B

1st level contain 16 entries

2nd " " " 128 "

3rd " " " 128 entries

Size of VAS?

L.A

4	7	7	8

⇒ 4+7+7+8

⇒ 26

- always deal in entries's

- if instead of 16 entries in 1st level it was given that it's 16B then we need to divide By PTE

⇒ $\frac{16B}{2B}$ = 8 entries

ex: 2level page table

V.A split in

<VPN_1, VPN_2, Offset>

1st 2nd

VPN_1	VPN_2	Offset

↳ is used to index the PTE in the 2nd level

ex: 2level page table

VA: 42bit & P.A: 40bit

PTE: 8B

P.S: 64KB pages at each level } chunk size

VA

		16

P.A

		16

40bit

Eg,

0x00 123 456 789

0000 0000 0001 0...

(i) which of the following address are the same 1st level page table entry: 0x00123456789

13	13	16

Chunk size = page size

2^{16}B ⇒ $\frac{2^{16}}{23} = 2^{13}$ entries

we want in entries

(a) 0x00 123 444 444

(b) 0x00 0A9 876 789

(c) 0x00 136 345 678

(d) 0x00 000 006 789

0x00 136...

0000 0000 0001 0...

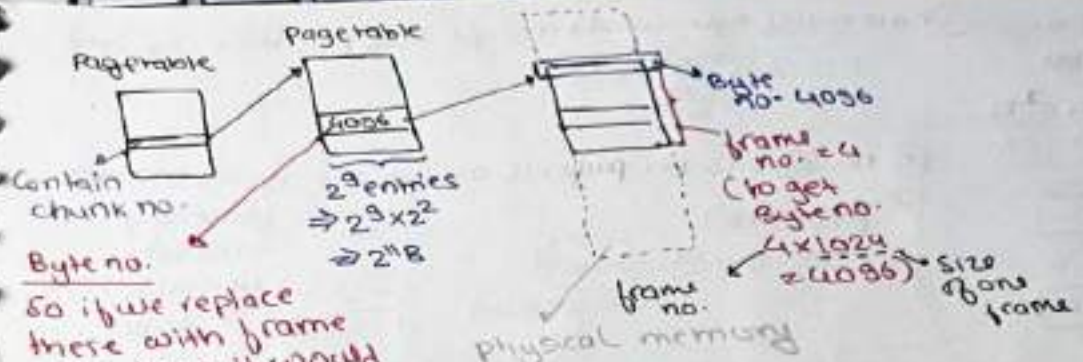
(ii) same as 2nd level page table 0x12345678

(a) 0x123F0000

(b) 0x000 46678

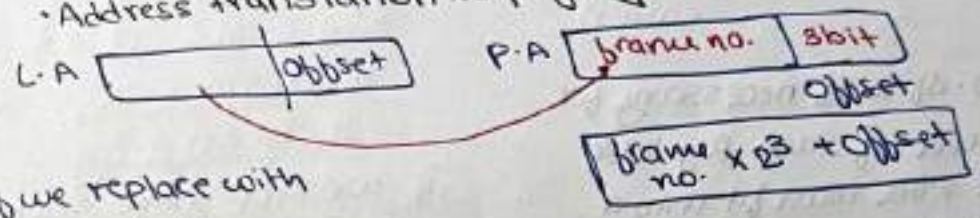
(c) 0x12345FFF

(d) 0x99345678



So if we replace these with frame no. then it would be convinient for Math & hardware

- Address translation in paging



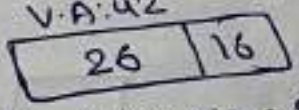
but if we replace with Byte no. then

Byte no. + offset

these is hard to compute.

ex. V.A: 42bit
P.A: 40bit
P.S: 64KB : 16bit
PTE: 8B

(9) Single level page table, no. of entries needed? How much memory is needed for storing the page table?



So, 2^{26} pages $\equiv$

Pagetable size: $2^{26} \times 8B = 2^{29}B = 512MB$

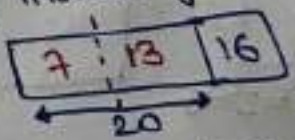
(6) V.A is mostly empty & multi-level is used to reduce the size

no. of level needed given that size of page table = size of physical page (chunk size)

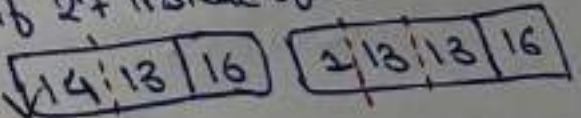
Page size: 2^{16}B

$\Rightarrow \frac{2^{16}}{2^3} = 2^{13}$ entries

if instead of 26 we have 20



& if 27 instead of 20



- we can partition anywhere in general
- but if chunk size of some level is given then we need to follow that

ex: P.A: 48bit
V.A: 32bit
P.S: 4KB (2^{12}B)
PTE: 64bit : 8B

P.A: 48bit

36	12

V.A: 32bit

20	12

(a) how many Byte would a single-level page table required
PTS ⇒ 2^{20} entries × 2^3B
⇒ 2^{23}B

(b) no of bit's per PTE are unused if PTE contain
PTE: 64bit
So, 64-39 = 25bit are unused

36 (frame no.)
+1 (valid bit)
+1 (writable bit)
+1 (Single bit)
39 bit

(c) how many PTE fit onto a single page
P.S: 2^{12}B ⟶ $\frac{2^{12}}{2^3} = 2^9$ entries (P.S in entries)
PTE: 2^3B

20	12

we can stop here itself if chunk size is allowed to be anything

(d) min no. of level necessary for multilevel if each page table at each level must fit into a single page

2	9	9	12

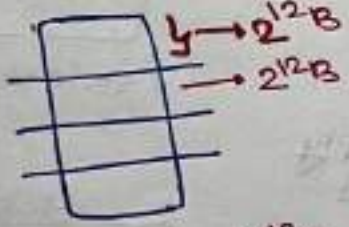
← 20 →

if we stop here, then no. of chunk = 1
chunksize: 2^{20} entries
⇒ 2^{23}B

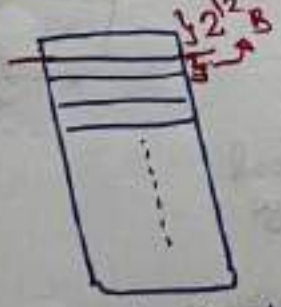
Multilevel paging is our choice (not forced) we can stop at any level

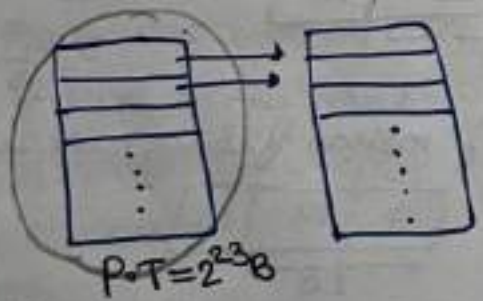
20	12

Chunk size: 2^{20} entries
= 2^{23}B



P.S: 2^9 entries
⇒ 2^{12}B





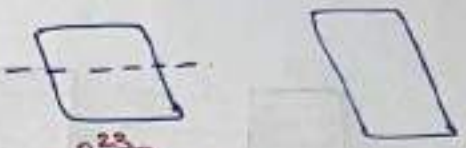
will also be Store in P.M

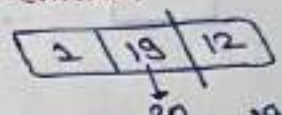
physical memory (P.M)

How to store PT = 2^{23}B into P.M if memory itself is divided into frame of lesser size in other word how we gonna file large PT into one frame?

to save P.T into M.M, we just need 2^{23}B contingous.
(why you even want to fit into one frame?)

- can we divide into 2 part & store
  yes, why not

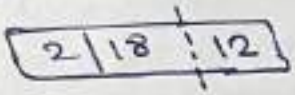




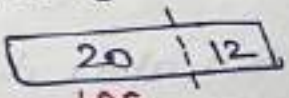
$\frac{2^{20}}{2} = 2^{19}$ entries.

- can we divide into 4 part & store?

$\frac{2^{20}}{2^2} = 2^{18}$ entries



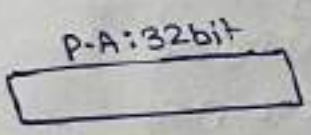
- can we choose not to divide at all? yes, why not

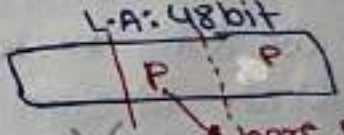


ex: VA: 48bit
PA: 52bit
4level pagetable
PTE: 8B: 2^3B
if P.S ↑ then no. of level ↓

how large must be the pagesize so that VA 48bit with only 2level pagetable

suppose given:
at every level page table must fit into a page

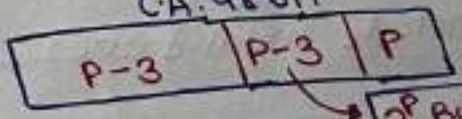




Suppose P.S: 2^P B

here P is 2^P entries

so writing P is wrong



2^P Byte
Chunk size
$\Rightarrow 2^{P-3}$ entries

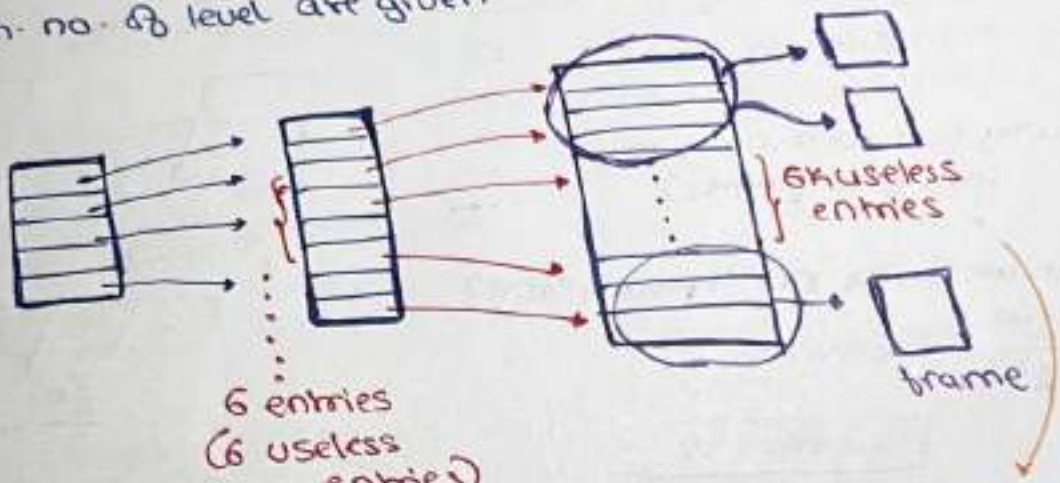
So, $P-3+P-3+P = 48$

$3P = 54$

$P = 18$

So, $PS = 2^{18}$ page's

→ the address translation in multilevel paging
→ what each entry of page table contain's
→ given that page table must fit into one page calculate the min no of levels
→ min. no. of level are given → calculate page size



Chunk size: 1k
6 useless chunk

Invalid chunks we will not store

- If are using single level paging or if we are not dividing then we don't have a choose of whether to store it or not we definitely have to store it. in continous manner
- In to store ALL entries (include invalid one)

- But after division we have the choose whether or not we will store it or not

[If we divide the page table into multiple page (chunk) then we have choice to save chunk independently hence we can avoid saving invalid entry page into physical mem.]

- Multi-level page table reduce the size of page table but increase the memory access time

ex: 32bit machine
V.A is divided in 4 segment

10bit	8bit	6bit	8bit

1st level

3level page table
P.T.E: 2B

$\#$ of useful page $= \frac{2^8 \times 2^{10} B}{2^8} = 2^{10}$ page's

10bit	8bit	6bit	8bit

1chunk 1chunk 16chunks } can't sum & say 18 chunks as it will not make any sense as chunksize is different

2^{10} entries 2^8 entries $16 * 2^6$ entries

now we can sum as entrie size is same

So, $(2^{10} + 2^8 + 2^{10})$ entries
$\Rightarrow (2^{11} + 2^8) * 2$Bytes
$\Rightarrow 2^{12} + 2^9$ B
$\Rightarrow 2^9(2^3 + 1)$ Byte
$\Rightarrow 9 * 2^9$ B

---

gate CSE 2024

ex 3level page table
V.A: 39bit

P.S: 4KB
PTE: 8B

A process P is currently using 2GB virtual memory mapped to 2GB physical memory 2^{31}B

min. amount of Memory required for the page table of P across all level is.

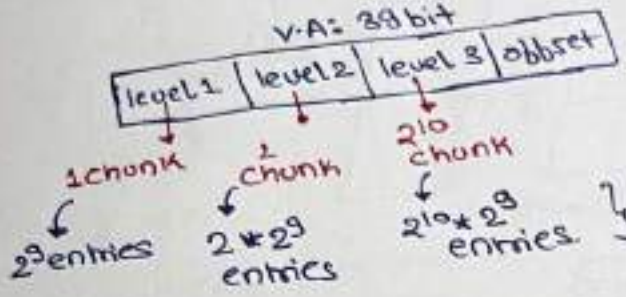
VA: 39bit

level 1	level 2	level 3	offset
9bit	9bit	9bit	12bit

Useful Byte $= 2^{31}$B

Useful page's: $\frac{2^{31}}{2^{12}} = 2^{19}$ page's

ⓐ P.S in such a system. 2^8: 256B
ⓑ Size of a page table for a process that has 256K of memory start with address 0?



$2^9(1+2+2^{10})$ entries
$\Rightarrow 2^3 * 2^9(3+2^{10})B$
$\Rightarrow 2^{12}(3+2^{10})B$
$\Rightarrow 2^2(3+2^{10})KB$
$\Rightarrow 4(3+2^{10})KB$
$\Rightarrow 1027 \times 4KB = \boxed{4108KB}$

(E) Size of the page table that has code segment of 48K starting at address 0x1000000 a data segment at address 0x80000000 & stack segment of 64K starting at address 0xf0000000 & growing upward.

code segment : 48KB → 0x1000000 → $\frac{48 \times 2^{10}}{2^8} = 48*4 = 192$ useful page's

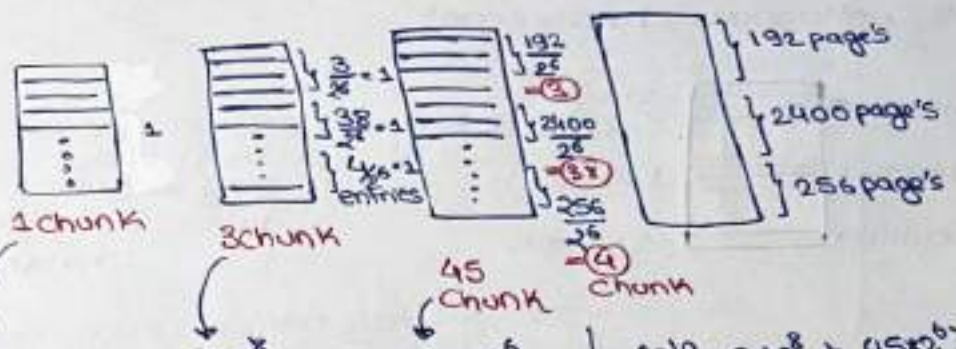
data segment : 600KB → 0x80000000 → $\frac{600 * 2^{10}}{2^8} = 600 \times 4 = 2400$ useful pages

stack segment : 64KB → 0xf0000000 → $64*4 = 256$ useful page's

Total useful page's : $48*4 + 600*4 + 64*4$ ⇒ 2848 page's

these direct may not be useful as there are scattered.

So we will calculate individual.



⇒ 9344B

~~Jan~~ ~~Feb~~ 2003

ex: 2 level page table.

for virtual to physical address translation

- page table for both level are store in M.M
- P.A = V.A = 32bit
- Byte addressable

for virtual to physical address, the 10 MSB of the V.A are used as index into 1st level page table while next 10 bit are used as index into the 2nd level page table. the 12 LSB of the V.A are used as offset

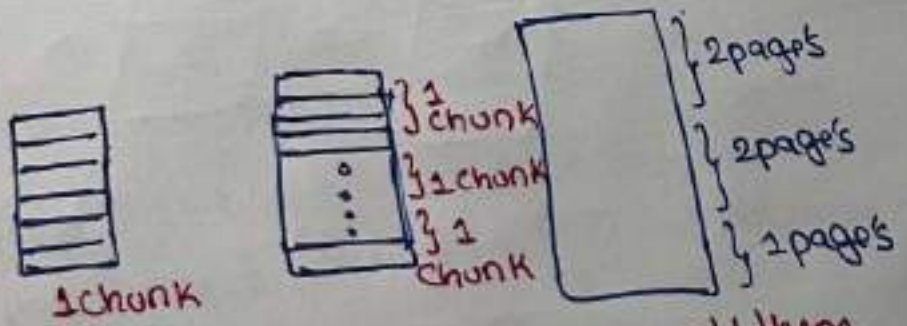
PTE in both level is 4B

- a process has only the following page in it's V.A space 2 contiguous code page start at 0x00000000 two continguous data page start at V.A 0x00400000 & stack page start V.A at 0xffff0000.. the amount of memory required for storing the page table of these process.

given | 10 | 10 | 12 |

P.T.E: 4B

Code : 2 page's
data : 2 page's
Stack: 1 page's



as, the chunk size is same we can add them

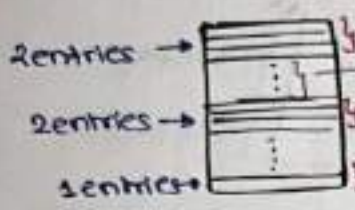
so total 4 chunk

⇒ $4 * 2^{10}$ ⇒ 2^{12} entries

⇒ $2^{12} * 2^2$ B ⇒ 16KB

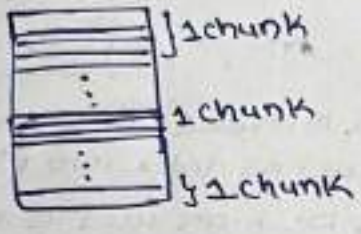
to 100% sure that ans is 16KB we need to check whether its contiguous entries or not

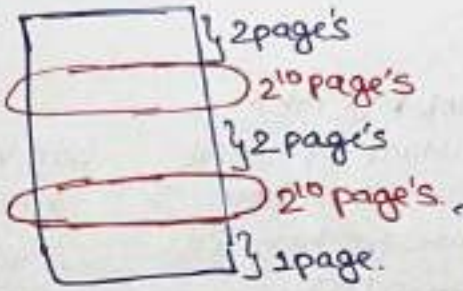
in 1 chunk = 2^{10} entries



we need to check whether there exist 1 chunk in b/w to say that they belong to 2 different chunk

- we need to check whether 2^{10} entries are there or not



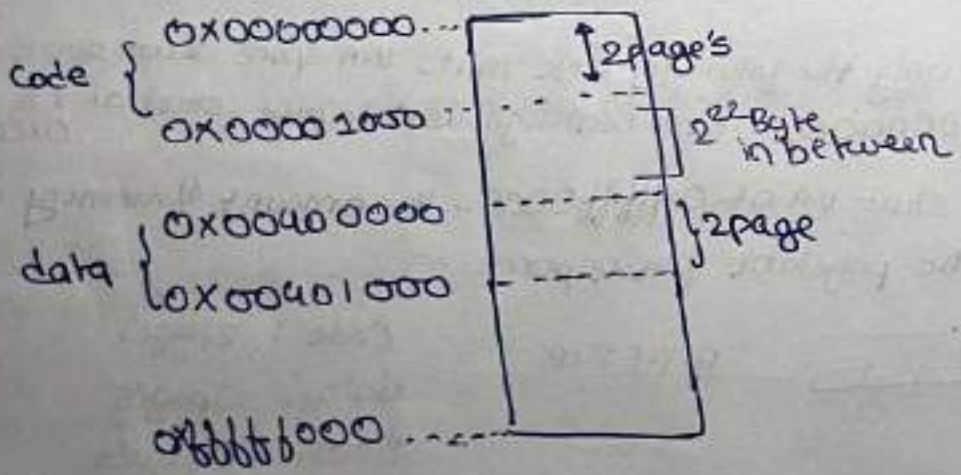


2^{10} page's means

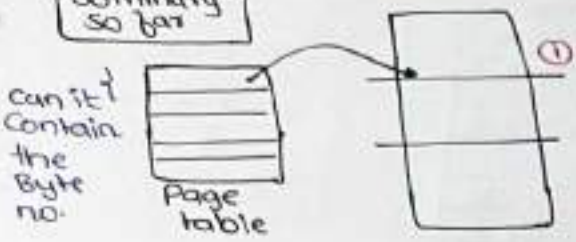
$2^{10} * 4KB$

$2^{10} * 2^{12}$

$\geq 2^{22}$ Byte



Summary so far



① Convert frame no. to Byte no.

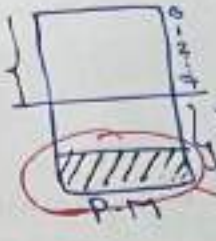
$$\text{Byte no.} = \text{Frame no.} \times \text{frame size}$$

② add offset to convert to P.A

Question

Suppose we have 8 page (each size 32B) to store in PM of size 2^{32}B.

bit's required to identify page no.?



if you are storing more than 7.

So, these page no. can't be converted into Byte no.

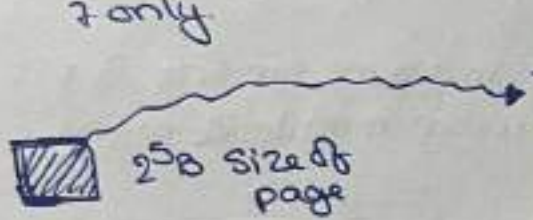
why 3 is wrong ans.

000 – 0
001 – 1
⋮
111 – 7

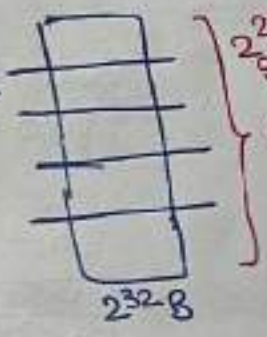
your page no. can range to 7 only

no. of page = 8 } doesn't matters to get the ans.

Size of the page matters



$$\text{no. of slot} = \frac{2^{32}}{2^5} = 2^{27}$$

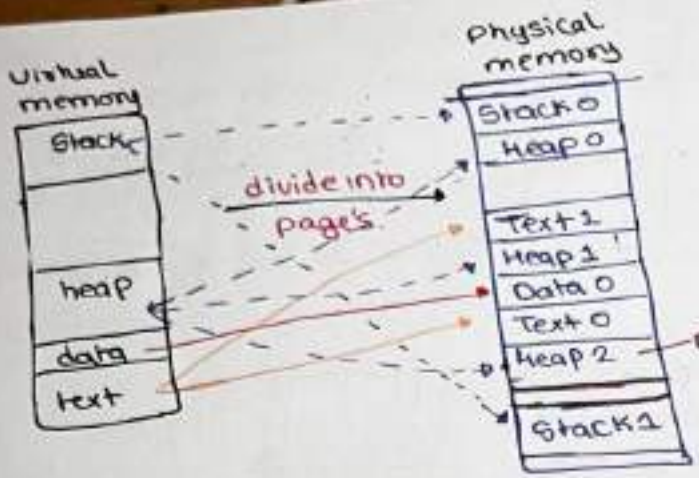


we need 27 bit (we need to identify in which slot we are storing)

- why multilevel page?

① [Page size : 4KB; page table size : 4MB (single)] → we can store the page table into M.M but we need lot of contigious space.

② even if we have contigious 4MB of space, is it always worth saving the page table? } no, what if page table contain too many invalid entries.



Paging

- Advantage
  1. no external fragmentation
  2. fast to allocate & free (no compacting)
- Disadvantage.
  1. Additional memory reference to page table
  2. Internal fragmentation
  3. Required space for P.T may be substantial.

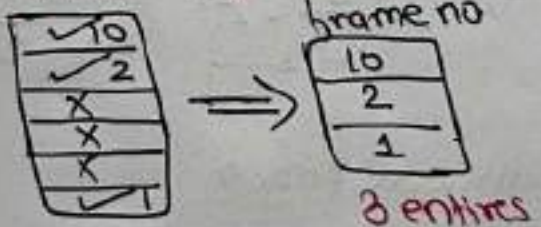
- one more idea to reduce page table size

Hashed page table

used as an alternative to multilevel paging (reducing the size of page table).

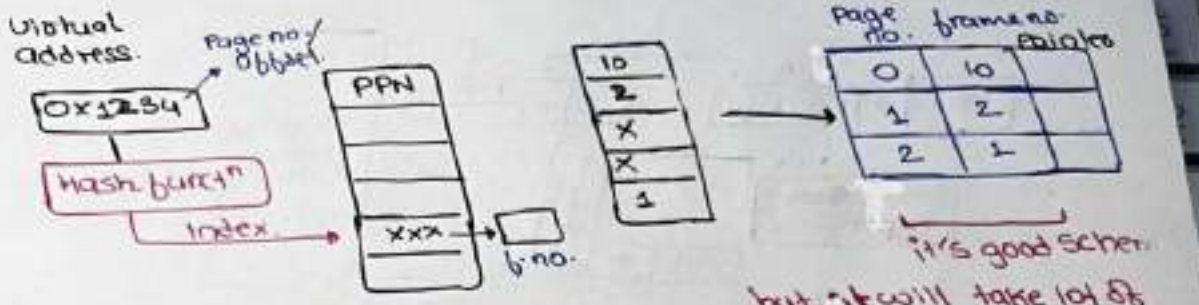
idea is to reduce the size of P.T but w/o using multilevel paging

Store only valid PTE.



P.T (suppose only 3 valid)

3 entries only



- hash value being the virtual page no.
- if collosion occurs then we use chaining method.

but it will take lot of time to search

So, solution is hashing.

- another way of reducing page table size

Inverted page table

one global page table for All process.

- for every frame there is a row.

hence size of inverted page table =

no. of frame x Inverted page table entry

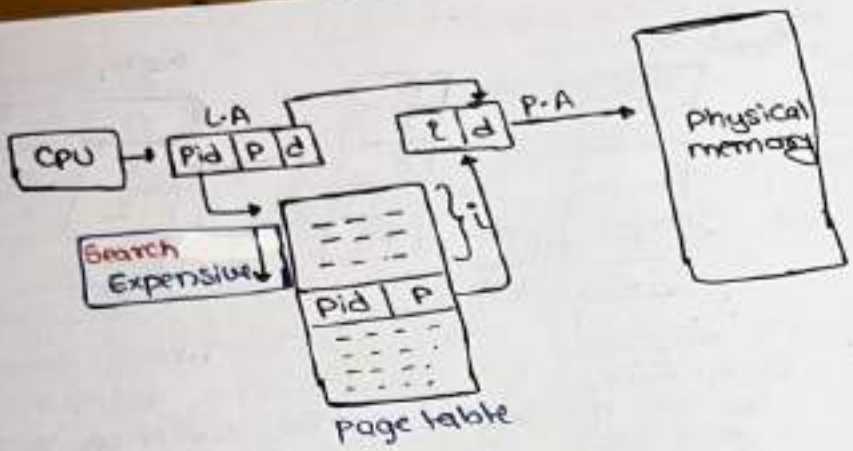
page table for P_1

page table for P_2

advantage & disadvantages

- Bounded amount of memory for page table
- Costly translation (using hashing can help)

Hashed inverted page table



ex: Assume we make a single memory access on a processor which has no Caches & no TLB. All else being equal.

(1) ~~(3)~~ inverted page table will on avg. be faster than single level page table need not be.
- doing memory access
- doing searching (doing multiple memory access)

(2) ~~(4)~~ multi-level page table on avg. faster than inverted page table depends on the no. of level's

(3) ~~(1)~~ Base & bound will on avg. be faster than a single level page table

Base & bound on single level required 2 memory access
- 1st need to access page table. (So, it will be slower)

in Base & Bound there is no meta data so we will directly access the frame. So that will be faster

(4) ~~(2)~~ Single level on avg. is faster than multi-level page table
- single level → one single memory access
- multi-level → required multiply memory access

- Problem with Segmentation
  - External fragmentation

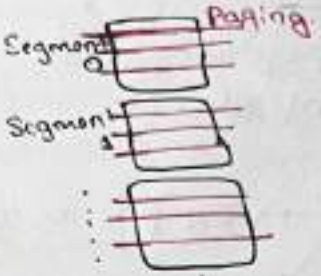
→ we required contigious space for one segment

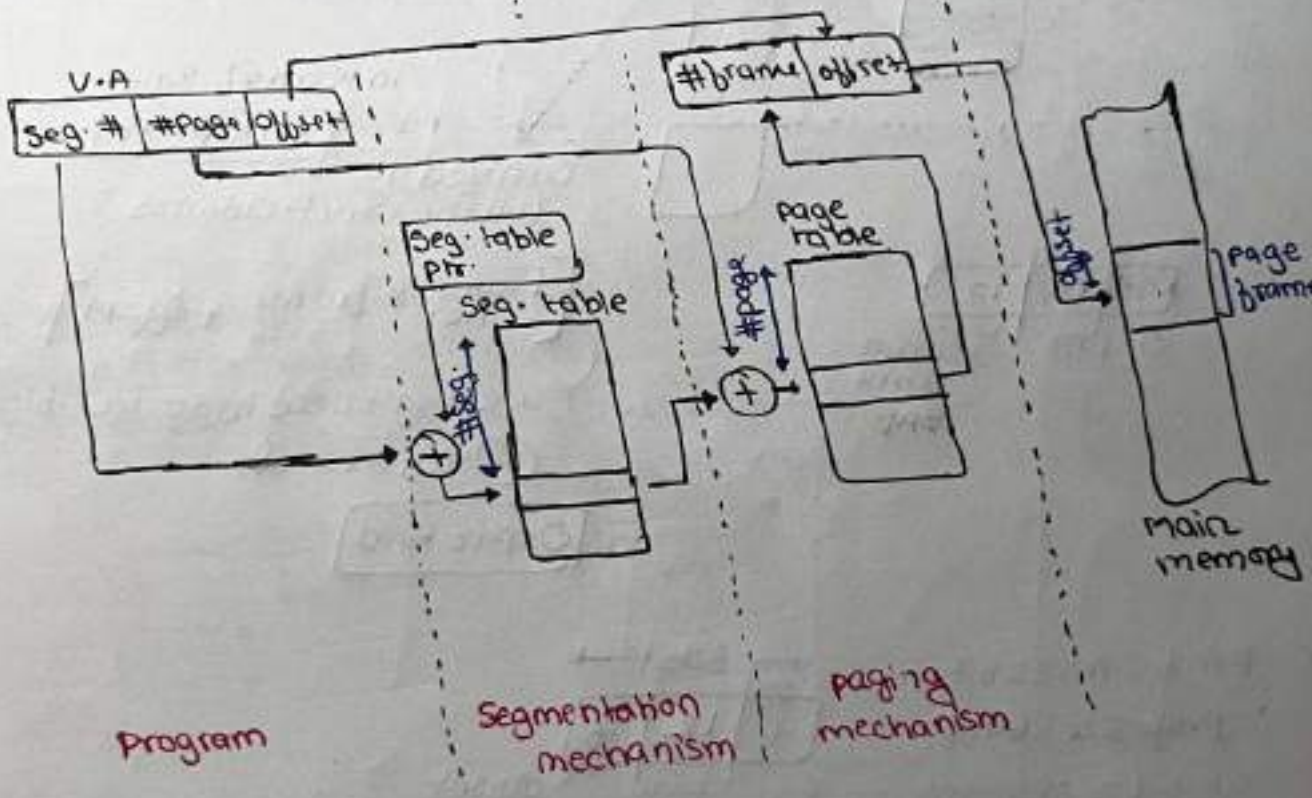
→ if the segment want to grow then we need to reshuffle.

- Problem with paging
  - Internal fragmentation (ptr's very less & managable)

Segmentation with paging

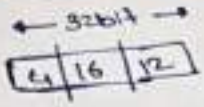
- we can page the Segment's itself





ex: A processor has 32bit address space & Combine both seg & paging

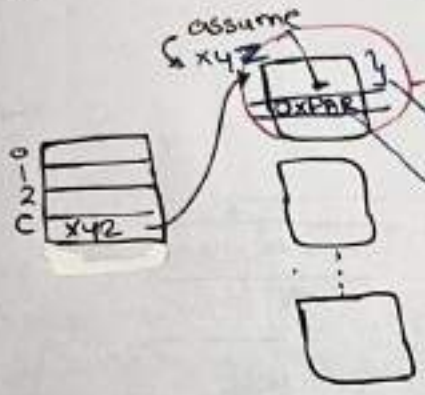
- 4bit for the segment
- 16 for page's &
- 12 for offset
- PTE is 4B



(a) a user process done a read of address 0x CODEDBAD. & it's in memory

seg no : C
page no : CODED → is page no (or) offset
offset : BAD

physical address ✗ no



there P.A we got from Seg no

how much down to go is told by page no.

in term of Byte

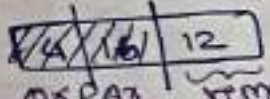
ODEDx4B

can you tell the exact address ?

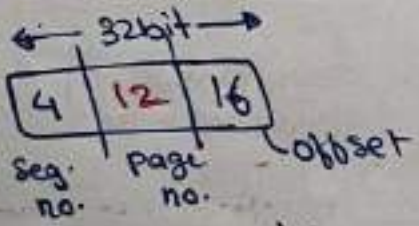
[frame no. x frame size + offset]

we don't need to do in binary (or) Hexdecimal.

0xPar BAD



ex. P.A & U.A : 32bit
Pagesize : 64KB
4bit for segment.



(a) max. possible size of a segment in this system in Byte.

each seg. can have : 2^{12} page's

$2^{12} * 2^{16}$ Bytes = 2^{28} Byte
⇒ 256MB

Gate 1999

ex: Segment page architecture for virtual memory

P.A & V.A : 2^{16} B

V.A.S is divided into & non-overlapping equal size Segment

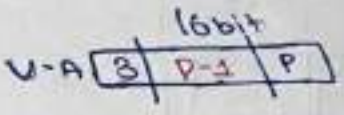
the MMU has a hardware Segment table

Each entry of which contains the P.A of the page table for the segment

& Page table are stored in MM

P.T.E = 2 Byte.

(a) min. Page Size in Byte so that the page table for a segment requires almost one page to store (P.S can only be in power of 2)

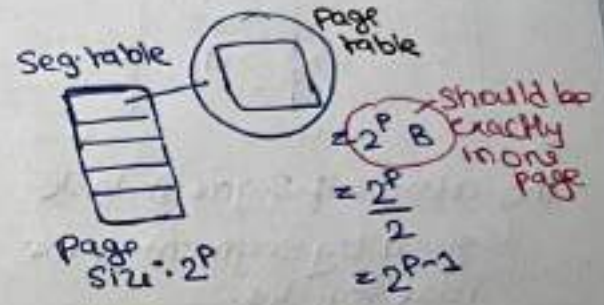


$\Rightarrow S+P-1+P=16$

$\Rightarrow 2(P+1)=16$

$P=7$

assume $P.S = 2^P$



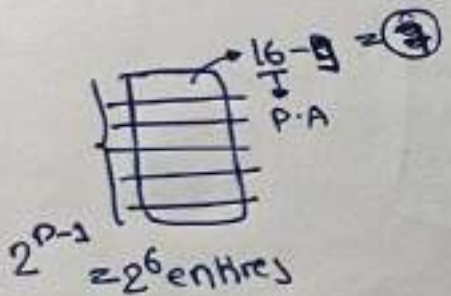
(b) Suppose P.T.E contain 1 valid bit, 3 bit for protection, & 1 dirty bit.

no. of bit available in P.T.E for storing information for the page.

(P.S : 512 B).

$7 + 1 + 3 + 1 = 12$ bit

4 bit for rest bit.



ex. 32 bit address divided into

4 bit	12 bit	16 bit
Seg id	page no.	offset

Seg. Id	P.T Ptr (P.A)	P.T.S
0	0x20000	0x4
1	0x30000	0x4
2	not valid	---
3	0x10000	0x4
4	not valid	----
....	not valid	-----
15	not valid	----

M·17

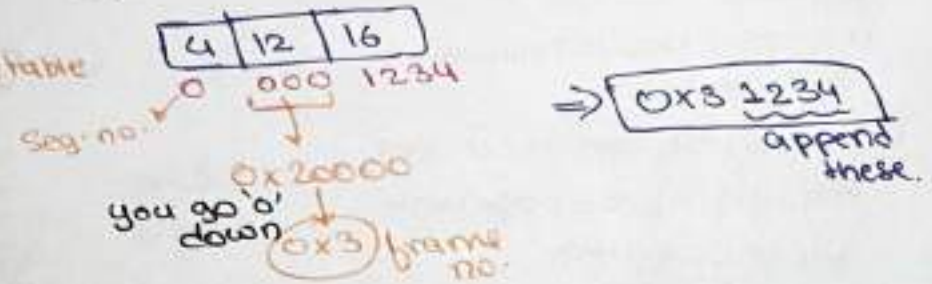
Address	value
0x10000	0x20
	0xA
	0xB
	0x5
....	....
0x20000	0x3
	0x7
	0x5
	0x9
....	....
0x30000	0x6
	0x4
	0x8
	0x11
....	....

page table

① content of physical memory

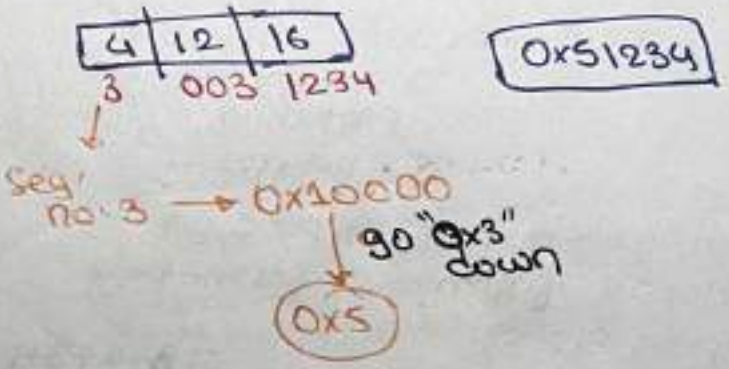
physical address to the following V·A map

ⓐ 0x00001234



ⓑ 0x30031234

PTE assume : 1B

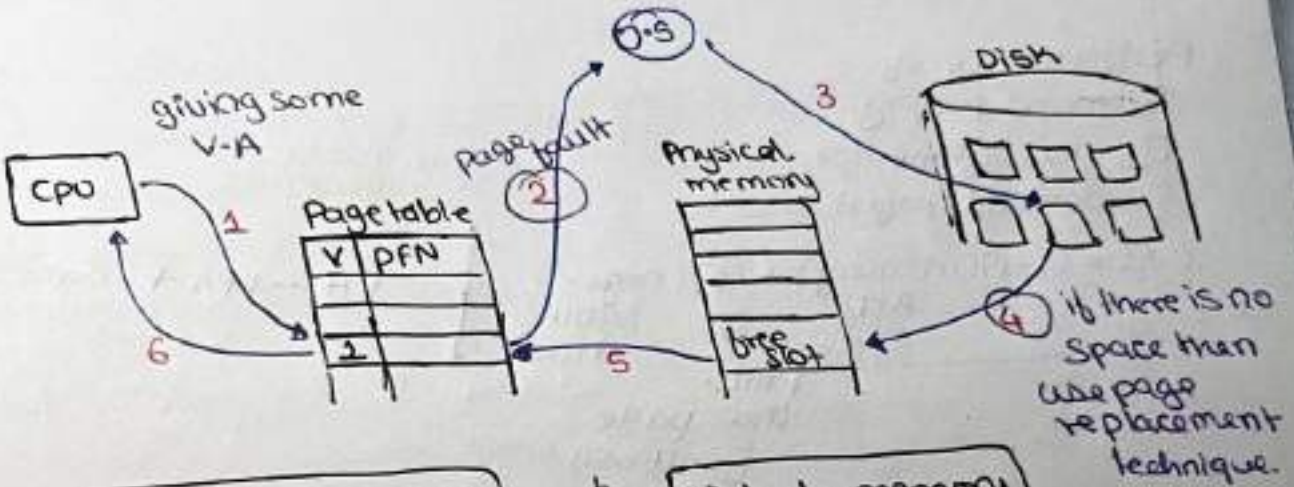


the address of Segment table is given by Segment table base register.

Demand Paging

Say's that even if we can load only one page in the m·m then also job can be executed.

- Since we are not pulling all page's in m·m hence some of the page's will also reside in 2nd Storage.
- "allow us to store page to disk"
(It means we can run processes larger than m·m)



Logical memory & Virtual memory

logical memory: we say to compiler that given memory is contiglous to you (use address from 0 to max)

Relocateable Code: any memory management technique we use in all case we are Converting L·A to P·A.

- whatever it's logically using can be even bigger than physical memory using Demand paging or Demand Segmentation

With virtual memory, we are not only able to run large process, also we are able to run many process.

- virtual memory is commonly implemented by demand paging

- Demand paging says.
  Keep some page's in physical mem. some page's in secondary memory
  Bring page from secondary memory to M.M whenever needed.

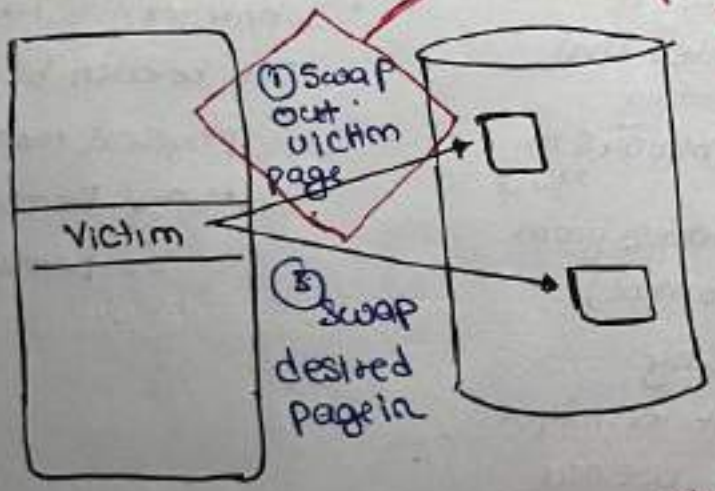
- Pure demand paging says you start with "0" page's in M.M keep all pages in secondary storage initially.

Performance of Demand paging.
Eff. access time for demand-paged.

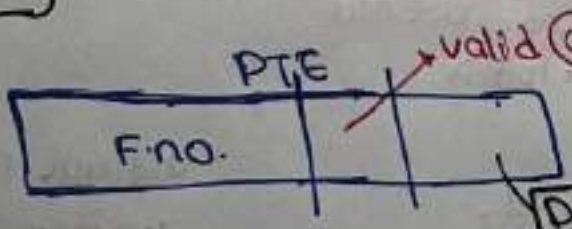
$$E.AT = (1-p)(\text{memory A.T}) + (p)(\text{page fault time})$$

p → Prob. that page fault will occur.

page fault time → can also be written as → VA → PA + data access.



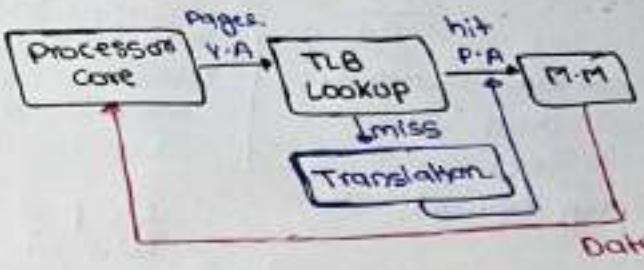
* we need to write the victim page into 2nd memory only if it has been modified.



the page is modified & should be written back to harddisk)

TLB
(Translation Lookaside Buffer)

is a memory which is way faster than M·M

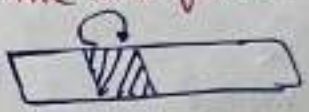


- TLB has few page table entry.

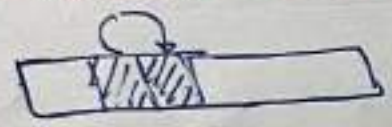
- why TLB works?

Locality programs tend to use data & Instruction's with address. near or equal to those they have used recently.

① Temporal locality:
- Recently referenced item's are likely to be referenced again in the near future



② Spatial locality
- Item with nearby addresses tend to be referenced close together in time.



ex: How many TLB Lookups assume 4KB page. → $4 \times 2^{10} = 2^{12}$B

```
int sum=0;
For(i=0; i<1024; i++){
    sum+=a[i];
}
```

a[] has 1024 item's each item is 4B.

Size of array = 4×1024
= 4096B
= 2^{12}B

- all of there can come in one page.
- So we are accessing one page no.
- initially TLB is empty.

- So only one miss will occur the initially one after that all will be hit

So, no. of TLB miss: $\underbrace{\frac{1}{1024}}_{\text{in \%age.}}$

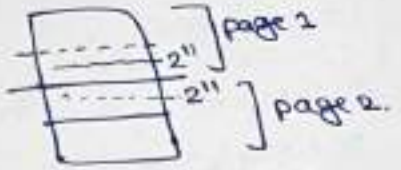
So
in the entire program only one P·T·E will be useful.

Size of array: $\frac{2^{12}B}{Data}$

we have seen that all data can be in one page.

① page required

It is also possible that half-half partition is done. (Some on one side & other on other side)



② page required atmost.

TLB

- Just like any other cache.
- Organized as fully, set, direct associative.

ex: machine
64bit address
P.S: 8KB : 2^{13}.B
TLB: 512 entires

what fraction of physical memory can be accessed w/o suffering a TLB miss?

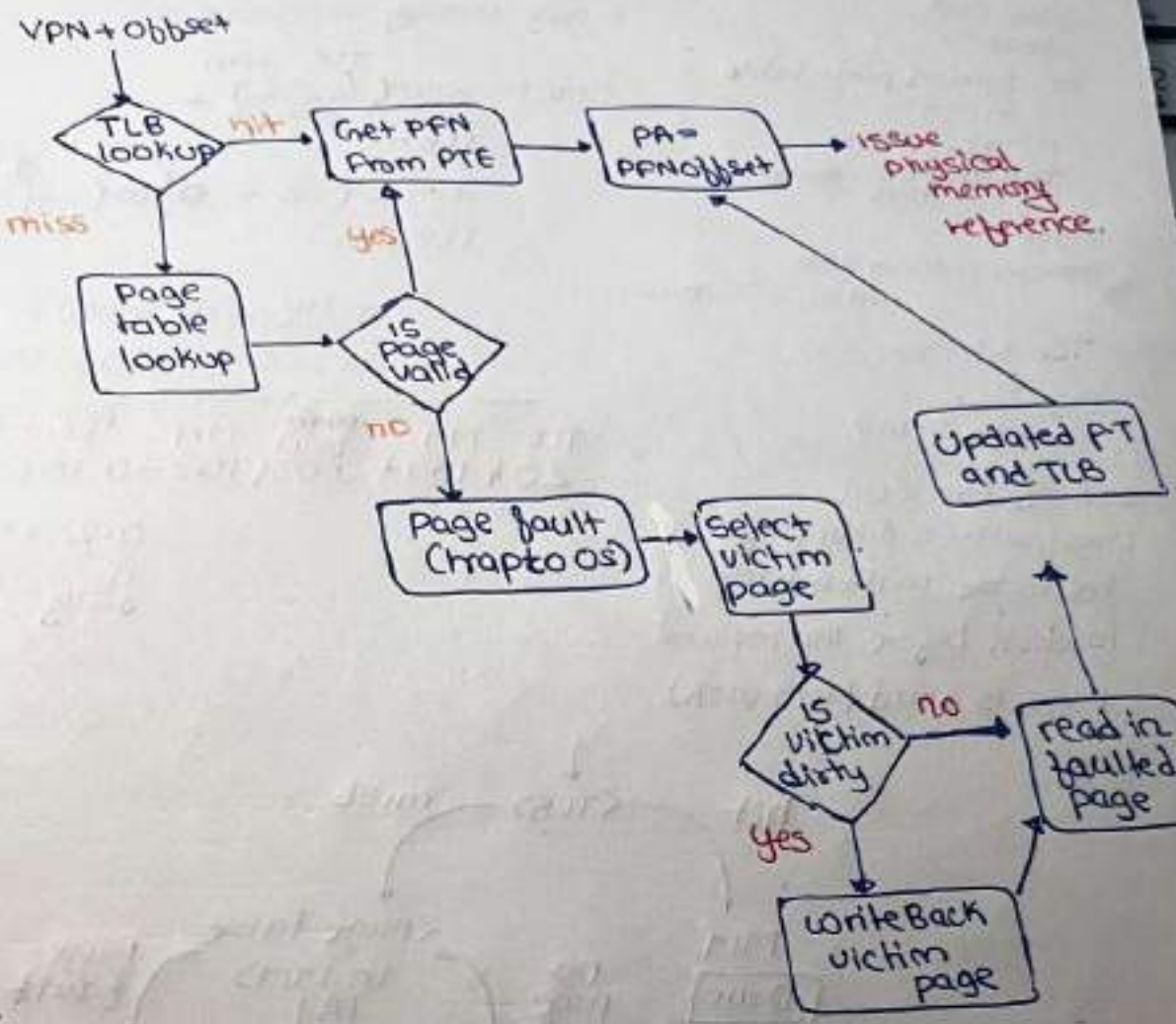
# of different page we can can access = 512 page. (2^9 page)

Our machine has 4GB RAM

$2^2 \times 2^{30} = 2^{32}$ PAS.

$\Rightarrow 2^9$ page $\times 2^{13} = 2^{22}B$
4MB

$\Rightarrow$ no. of fraction w/o TLB miss $= \frac{2^{22}}{2^{32}} = \frac{1}{2^{10}}$



gate 2019.

ex: V.A: 64bit
P.A: 48bit
word addressable
P.S: 8KB
word size: 48

the TLB in the address path has 128 valid entries.

Q. atmost how many distinct virtual addresses can be translated w/o any TLB miss?

VA: 64 bit : 2^{64} words.

PA: 48 bit.

P.S: $\frac{2^3 * 2^{10} B}{2^2} = 2^{11}$ word.

TLB = 128 valid entries

$\Rightarrow 2^7 * 2^{11}$ words

$\Rightarrow 2^8 * 2^{10} B$

256MB

gate CSE 2020

ex: 2-level page table in MM

MMAT: 100ns

TLB access time: 20ns

Page transfer to/from disk takes 5000ns.

TLB hit rate: 95%.

Page fault rate: 10%.

20% of the total pagefault (a dirty page has to be written back to disk before the required page is read from disk).

• avg. memory access time is.

EMAT = 0.95 [20 + 100] + (TLB hit; TLB, MM)

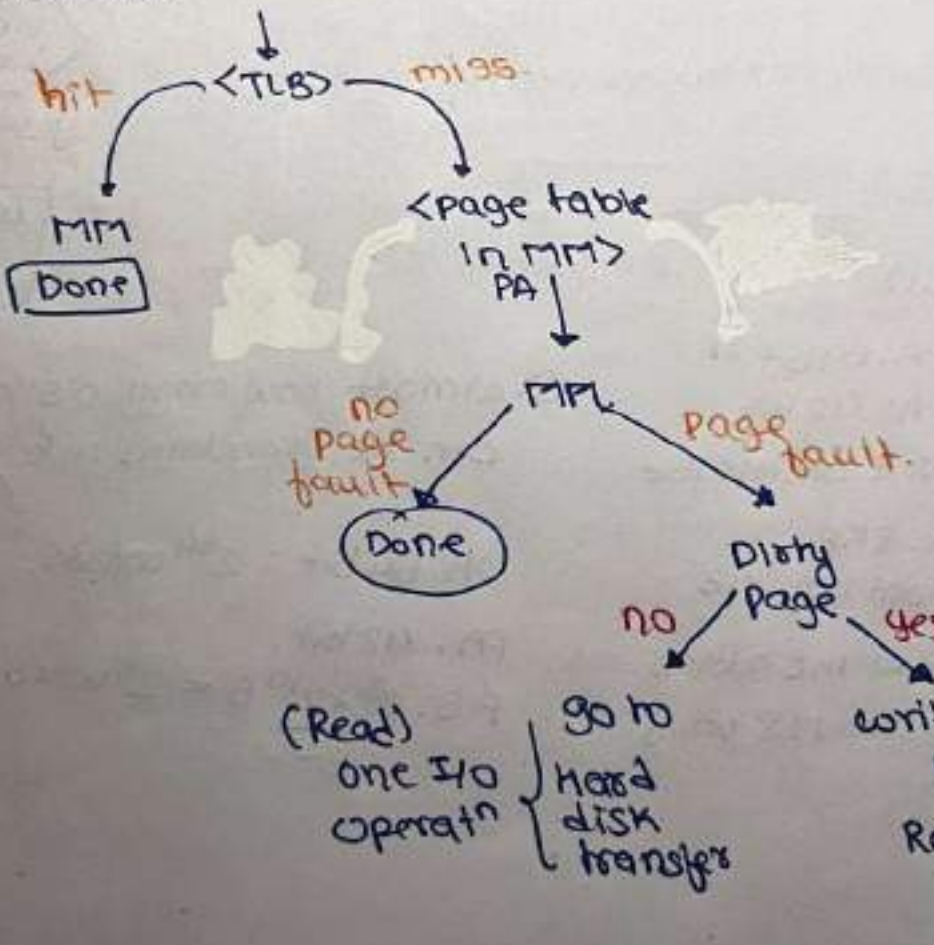
0.05 (20 + 100 + 100 + [0.90(0) + (no page fault)

0.10 (0.80 (5000) + 0.20 (5000 + 5000)) ⇒ 155.0 ns.

(TLB miss; TLB, MM, TLB fault, MM, page fault, one transfer)

20 + 100 + 0.05 (100 + 0.10 (5000 + 0.20 × 5000))

(if dirty; again transfer)



- M.E fail then Race condition occurs
- progress fails then Deadlock may occur
  - ↳ progress (if finite time fails) in that case Deadlock will occur
- Bounded wait fail's then starvation may come

- Bounded waiting & deadlock are 2 different concept. So Bounded waiting doesn't imply no Deadlock. A system could satisfy Bounded waiting & could be in deadlock as well.
- If a process is not using the critical section then it should not stop any other process from accessing it. (Independent of each other)