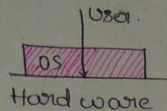


OPERATING SYSTEMS

1. PROCESS MANAGEMENT

1. INTRODUCTION TO OS

⇒ Operating system is an interface between user and Hardware.



⇒ Operating system acts as Resource Allocator. (Resource means anything like CPU, memory, printers), OS takes care of Resource Allocation (takes the responsibility to allocate CPU to some process, I/O to some process etc). Some Resources cannot be shared.

⇒ OS acts as manager ⇒ keeps track of the resources that are allocated to particular process (Memory, process, files, security are managed by OS).

GOALS:

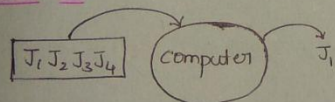
primary goal = Convenience

Secondary goal = Efficiency.

Types of operating system

Batch OS, Multiprogramming, Multitasking, multiprocess, Realtime.

Batch OS



⇒ processes need two types of times they are CPU time and I/O time.

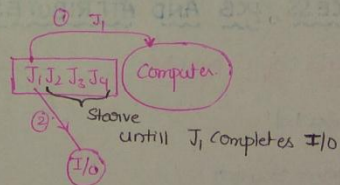
⇒ After completion of one job the other will start

⇒ Not useful for interactive Applications.

⇒ Starvation.

⇒ Less Throughput

⇒ No preemption.



Multiprogramming

⇒ CPU will not be kept idle

⇒ whenever a Job needs I/O then the CPU takes the next process and process it.

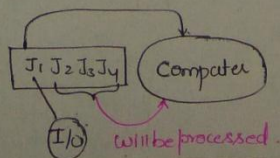
∴ CPU will be busy all the time.

⇒ No Starvation.

⇒ Efficiency is more

⇒ More Throughput.

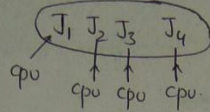
⇒ No Preemption.



Multi Tasking:

⇒ The cpu will be multiplexing among the Jobs without completing the Job completely and that way interactivenss can be improved.

⇒ preemption is present.



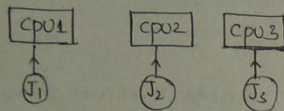
Multi processing:

⇒ Instead of single cpu we will have lots of cpus in a single computer.

⇒ Many Jobs can be processed simultaneously.

⇒ parallelism is improved.

⇒ Throughput is more. (Throughput is the no. of jobs that could execute in unit time)



⇒ Dual core (2 cpus) Quadcore (4 cpus)

⇒ Each cpu can be used as we wish we could use cpu1 as Batch processing, cpu2 can be used as Multi tasking etc.

Realtime:

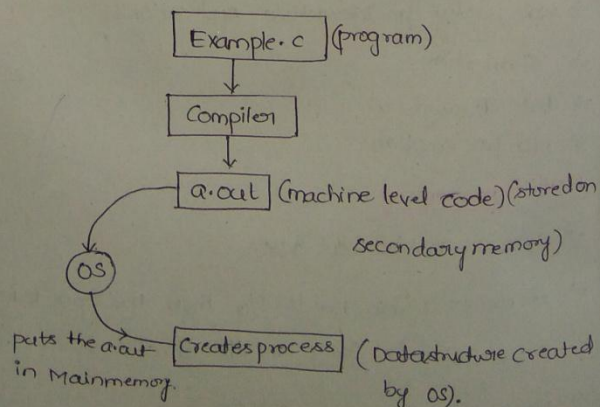
⇒ we will have jobs with Deadlines.

⇒ Result produced after the deadline is waste

2. PROCESS, PCB AND ATTRIBUTES

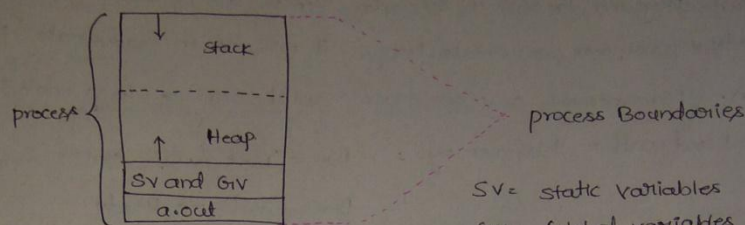
Attributes of a process

- 1> process id
- 2> program counter
- 3> process state
- 4> priority
- 5> General purpose Registers
- 6> list of open files
- 7> list of open devices
- 8> Protection.



completely

The operating system takes the a.out file and creates a structure like this in the main memory.

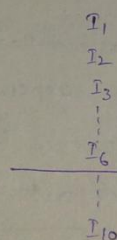


SV = static variables
GIV = Global variables
Heap = for dynamic allocation
malloc(), calloc() etc.

Unit time) $\Rightarrow$ The execution should not cross process boundaries, if it crosses we get "segmentation fault."

$\rightarrow$ process Id: Unique no. that is given to a process to identify it.

$\rightarrow$ program counter: Stores the starting point of next instruction.

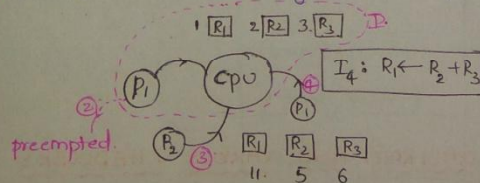


Executed till here and then we stopped this process and tried to execute the next one and again whenever we come to this process we start from I_7 and I_7 will be stored in program counter.

$\rightarrow$ process state: New, Ready, Running, Blocked

$\rightarrow$ priority: Number that is assigned to the process whenever it is created.

$\rightarrow$ GPR:



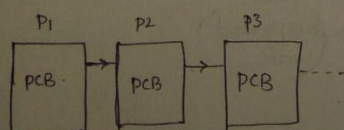
stored on
copy)

$\rightarrow$ list of open files: While execution some files are opened for reading and some for writing we should remember which files we have read/which files we have written.

created

$\Rightarrow$ All this information will be present in process control block

$\Rightarrow$ Every process gets its own PCB.



All the PCB are linked using linked lists.

3. PROCESS STATES AND MULTI PROGRAMMING

States of process:

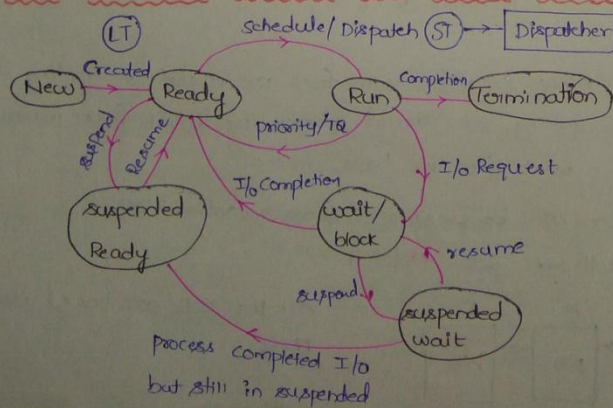
- 1> New = program present in Secondary Memory and is Ready to pick by the OS is called "New" state.
- 2> Ready = whenever we create process it will be in Ready state (process is in Main Memory).
- 3> Run = Main memory - only one process will be running in a single cpu.
- 4> Block and wait = Main memory - Run $\rightarrow$ Block / wait state
Block / wait $\rightarrow$ Ready
Ready $\rightarrow$ Run
- 5> Termination / completion = Context will be deleted, Every Trace of process will be deleted.
- 6> Suspend Ready = when the main memory is full and a new process having a highest priority has arrived then a process in the main memory should be suspended and room must be made for new process.
- 7> Suspended wait / suspend block = same as suspend Ready the diff is instead of suspending the process in Ready state, suspend the process that are in block state (secondary Memory). when a process is in suspend wait and it has completed its I/O then it makes transition to Suspend Ready.

Operations on processes

- 1> Creation
- 2> Scheduling
- 3> Execute it
- 4> Killing / Delete

Ready, Run, Block and wait - Main memory
New - Secondary memory.

4. PROCESS STATE TRANSITION DIAGRAM AND VARIOUS SCHEDULERS



TQ = Time Quantum.

LT = long Term scheduler

ST = short term scheduler

MT = medium Term scheduler

5. QUESTION

Consider a system

State
Ready
Running
Blocked

Degree of multiprogramming = No. of processes that can be present in the Ready state at maximum. (This factor is decided by long Term scheduler) (5)

⇒ Short Term scheduler decides which process must be executed next and after deciding it will call "Dispatcher". Now, Dispatcher is the algo that is responsible for moving the process from Ready-Run and Run-Ready state. It is called "Context-Switching".

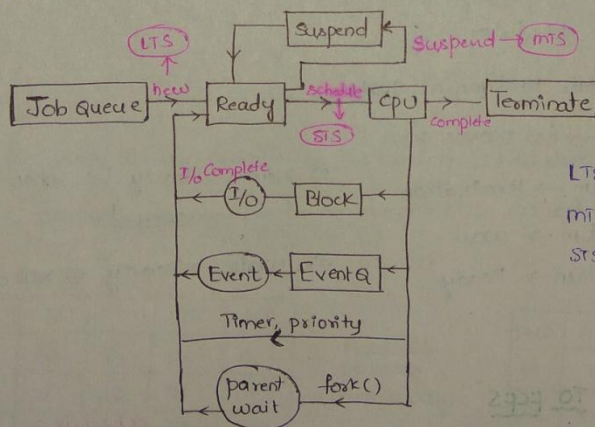
⇒ The medium/mid Term Dispatcher is responsible for swapping. Swapping is the process of moving the process from Main memory to secondary memory and from secondary memory to Main memory.

Long Term scheduler - Related to overall performance

Short Term scheduler - Related to context switching time

Medium Term scheduler - Impacts Swapping Time

5. PROCESS QUEUES



LTS = Long Term scheduler

MIS = Medium " " "

STS = Short Term scheduler

6. QUESTION ON PROCESS STATES

Consider a system with 'N' CPU processors and 'M' processes then.

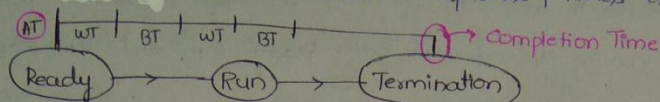
State	min	max
Ready	0	M
Running	0	N
Block	0	M

1. VARIOUS TIMES RELATED TO PROCESS

1) Arrival time: The time at which the process enters "Ready Queue".

2) Burst time: The amount of CPU time required by a process to finish.

3) Completion time: The time at which the process finishes execution.



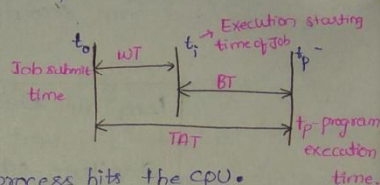
$$CT - AT = BT + WT \Rightarrow CT - AT = TAT$$

4) Turn Around Time: The difference between the Completion and Arrival Times is called TAT.

$$TAT = CT - AT$$

5) waiting Time:

$$WT = TAT - BT$$



6) Response time: What is the first time the process hits the CPU.

8. CPU SCHEDULING

Picking the process from Ready state and giving it to CPU is called CPU scheduling.

Who → Short term scheduler

Where → Ready state to Running state

When → when a process moves from

I. a) Run → Termination

b) Run → wait

c) Run → Ready

2) New → Ready i.e. when a process is just created

3) wait → Ready ⇒ Not all time → Based on priorities.

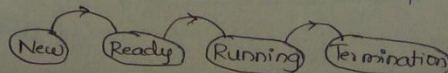
9. INTRODUCTION TO FCFS

Criteria: Arrival Time

Mode: Non-preemptive

PNO	AT	BT
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

⇒ The mode is Non-preemptive because when we give a job to CPU we will not forcibly pull it from CPU. we wait until the job is finished. so the mode is Non-preemptive.



Gantt chart

⇒ In case lower p

⇒ In the waiting ti

10. CONVOY

⇒ when a all the

PNO
1
2
3

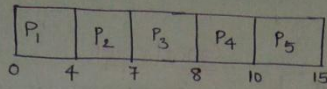
Gantt chart

Pi
0
20

Avg WT = $\frac{19}{3}$

Avg TAT = $\frac{80}{3}$

Gantt chart:



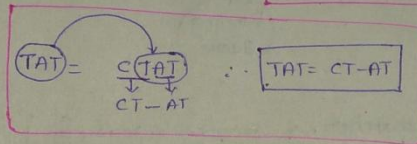
⇒ In case if 2 process have same arrival times then pick the process with lower process id/pno.

PNO	CT	TAT = CT - AT	WT = TAT - BT
1	4	4	0
2	7	6	3
3	8	6	5
4	10	7	5
5	15	11	6

$$\text{Avg. TAT} = \frac{4+6+6+7+11}{5}$$

$$\text{Avg. WT} = \frac{0+3+5+5+6}{5}$$

⇒ In the case of Non-preemptive version Both the Responsetime and waiting time of a process is same ⇒ Response time = waiting Time



10. CONVOY EFFECT

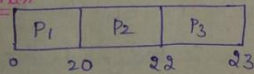
⇒ when a process with heavy Bursttime is being scheduled by the cpu then all the other process are going to starve, this is called "convoy Effect."

PNO	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	2	22	21	19
3	1	1	23	22	21

Reschedule
→
in this way

PNO	AT	BT	WT	TAT	CT
P1	1	20	2	22	23
P2	0	2	0	2	2
P3	0	1	2	3	3

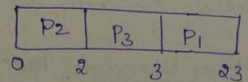
Gantt chart



$$\text{Avg WT} = \frac{19+21}{3} = \frac{40}{3}$$

$$\text{Avg TAT} = \frac{20+21+22}{3} = \frac{63}{3} = 21$$

Gantt chart



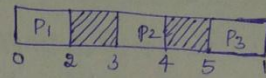
$$\text{Avg. WT} = \frac{0+2+2}{3} = \frac{4}{3}$$

∴ WT has drastically reduced

11. FCFS Example

PNO	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0

Gantt chart



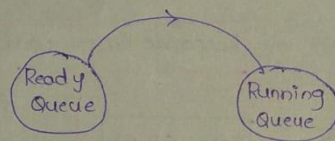
Here no process is waiting for CPU. In fact CPU is waiting for the processes.

⇒ Queue is the simplest data structure that is used to implement FCFS.

12. FCFS WITH OVERHEAD

⇒ The context switching time " δ " will be given.

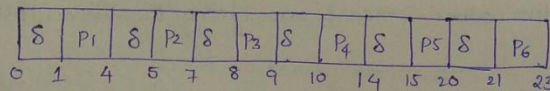
⇒ The context switching time (δ) is when a process is running, we are going to stop it and schedule the other process. In between many things will happen now,



{ scheduler → Dispatcher → process is ready for execution } Time taken for this entire process is called context-switching time

PNO	AT	BT
1	0	3
2	1	2
3	2	1
4	3	4
5	4	5
6	5	2

Gantt chart



⇒ Before the first process is called the scheduler will be called which in turn calls the dispatcher and the process is made available for execution. So context-switching occurs before every new process is brought to execution. So the total life cycle takes 23ms to complete out of which 6ms is wasted for context switches and hence the Algo's efficiency decreases.

$$\text{Efficiency} = \frac{6 \times 1}{23} \times 100$$

$$\text{Efficiency } \eta = (1 - \frac{6}{23}) \times 100$$

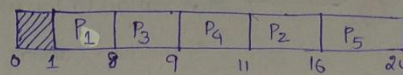
13. INTRODUCTION TO SJF

Criteria - Burst time

Mode = Non-preemptive

PNO	AT	BT	CT	TAT	WT
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11

Gantt chart



⇒ Shortest Job first Algorithm schedules the job having the least Burst time among the "AVAILABLE PROCESS."

⇒ victim of convoy effect.

⇒ The data structure used is "Minheap." (Root will be the process with least Burst time)

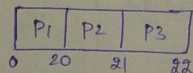
14. ANALYSIS OF SJF

⇒ SJF cannot be implemented practically.

PNO	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	1	21	20	19
3	2	1	23	21	20

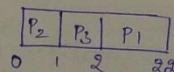
$$\text{Avg. WT} = \frac{39}{3} = 13 \text{ ms.}$$

$$\text{Through put} = \frac{3}{22}$$



PNO	AT	BT	CT	TAT	WT
1	2	20	22	20	0
2	0	1	1	1	0
3	1	1	2	1	0

Gantt chart:



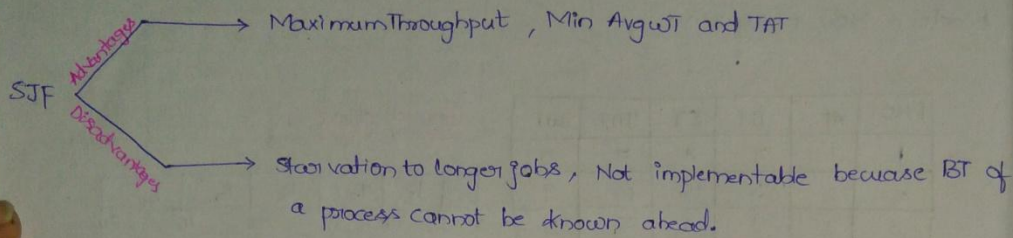
$$\text{Avg. WT} = 0$$

$$\text{Through put} = \frac{3}{22} \rightarrow \begin{matrix} \text{No. of process} \\ \text{Total schedule time} \end{matrix}$$

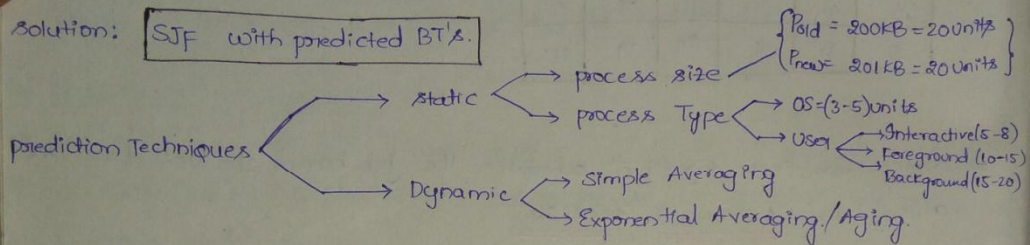
⇒ SJF gives the best through put than any other scheduling Algorithms.

15. SJF WITH PREDICTION of BT

(10)



Solution: SJF with predicted BT's.



One of the Best method available for Burst Time predicting is "Simple Averaging"

Simple Averaging

→ Given 'n' processes $(P_1, P_2, \dots, P_n)$

→ Let T_i be the actual BT.

→ Let τ_i denote the predicted BT

$$\tau_{n+1} = \frac{1}{n} \sum_{i=1}^n t_i$$

Exponential Average/Aging

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \quad (0 \leq \alpha \leq 1)$$

$\begin{cases} t_n = \text{Actual BT} \\ \tau_n = \text{predicted BT} \end{cases} \quad \alpha = \text{smoothing factor}$

Ex: $\alpha = 0.5$, $\tau_1 = 10$ actual BT $(t_1, t_2, t_3, t_4) = (4, 8, 6, 7)$ then $\tau_5 = ?$

$$\text{Now, } \tau_5 = (0.5)t_4 + (1-0.5)\tau_4$$

$$\tau_5 = (0.5)7 + (0.5)(6.75)$$

$$\tau_5 = 6.875$$

$$\text{Now, } \tau_4 = (0.5)t_3 + (0.5)\tau_3$$

$$\tau_3 = (0.5)t_2 + (0.5)\tau_2$$

$$\tau_2 = (0.5)t_1 + (0.5)\tau_1$$

$$\tau_1 = 10$$

$$\text{Now, } \tau_2 = (0.5)(4) + (0.5)10$$

$$= 2 + 5 = 7$$

$$\tau_3 = (0.5)8 + (0.5)7$$

$$= 4 + 3.5 = 7.5$$

$$\tau_4 = (0.5)6 + (0.5)7.5 = 6.75$$

16. INTRO

SRTF =

Criteria =

MODE : p

PNO
1
2
3
4
5
6

Gantt chart

⇒ cannot

⇒ Any s.c.
because

ExAmpl

PNO
1
2
3

18. Example

PNO
1
2
3
4

(10)

16. INTRODUCTION TO SRTF

SRTF = Shortest Remaining Time first.

Criteria = BT's.

MODE: pre-emptive

PNO	AT	BT	CT	TAT	WT
1	0	7	19	19	12
2	1	5	13	12	7
3	2	3	6	4	1
4	3	1	4	1	0
5	4	2	9	5	3
6	5	1	7	2	1

PNO	BT
1	7 0
2	5 0
3	3 0
4	1 0
5	2 0
6	1 0

Gantt chart :

P1	P2	P3	P4	P3	P3	P6	P5	P2	P1
0	1	2	3	4	5	6	7	9	13

⇒ cannot be practically implemented.

⇒ Any scheduling algorithm related to BT cannot be implemented practically because BT cannot be predicted accurately.

17. Example on SRTF GATE 2011

PNO	AT	BT	CT	TAT	WT
1	0	9	13	13	4
2	1	4	5	4	0
3	2	9	22	20	11

What is the Avg. WT using SRTF?

$$\text{Avg. WT} = 15/3 = 5 \text{ ms.}$$

P1	P2	P2	P1	P3	
0	1	2	5	13	22

18. Example on SRTF GATE 2007

PNO	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	15	25	55	40	15
3	30	10	40	10	0
4	45	15	70	25	10

What is the WT of P2?

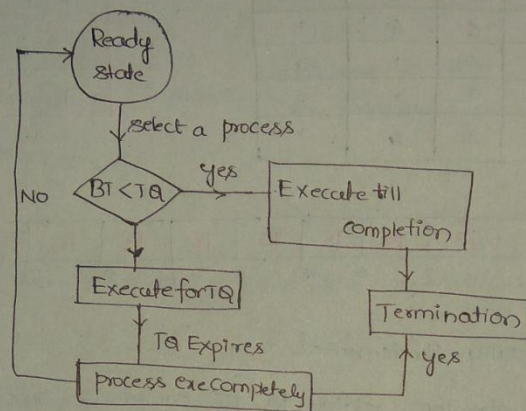
Gantt chart

P ₁	P ₁	P ₂	P ₃	P ₂	P ₂	P ₄	
0	15	20	30	40	45	55	70

$$\therefore \text{WT of } P_2 = 15$$

19. ROUND ROBIN ALGORITHM

- ⇒ Many of the operating systems in practice use Round Robin scheduling.
- ⇒ Not depending on BT
- ⇒ Simple Data structure: Queue
- ⇒ Each process is executed for TQ amount of time (TQ = Time Quantum)
- ⇒ Less starvation problem.



20. ROUND ROBIN EXAMPLE 1

CRITERIA: TQ + AT (First come first serve)

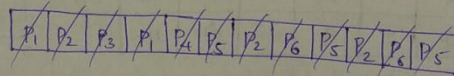
ROUND ROBIN: PRE EMPTIVE

TQ = Max amount of time for which a process is allowed to execute when it is scheduled.

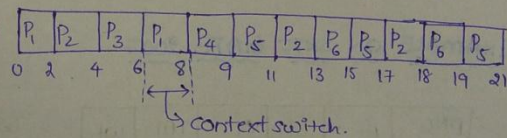
PNO	AT	BT
1	0	4
2	1	5
3	2	2
4	3	1
5	4	6
6	6	3

TQ = 2

Queue:



Gantt chart:

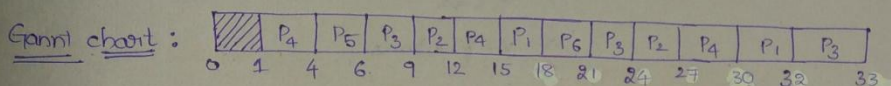
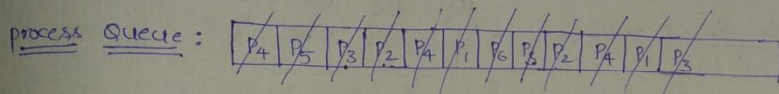


- ⇒ As the Time Quantum [↑] then no. of context switches [↓]
- ⇒ If the TQ is too large then starvation occurs.

21. ROUND ROBIN Example 2

TQ=3.

PNO	AT	BT	CT	TAT	WT	RT
1	5	5	32	27	22	10
2	4	6	27	23	17	5
3	3	7	33	30	23	3
4	1	9	30	29	20	0
5	2	2	6	4	2	2
6	6	3	21	15	12	12



PNO	BT
1	5 0
2	6 0
3	7 0
4	9 0
5	2 0
6	3 0

Response Time (RT): The diff b/w the time at which a job is submitted to cpu and the time which cpu starts scheduling it.

Ex: P_1 has been given to cpu at 5 (AT) but cpu started P_1 at 15 (look at Gantt chart)

$$\therefore (RT)_{P_1} = 15 - 5 = 10$$

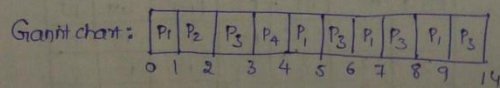
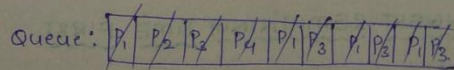
TQ	CS	RT
↑	↓	↑
↓	↑	↓

If $TQ = \infty$ then Round Robin will become FCFS.

22. ROUND ROBIN Example 3

Consider 4 Jobs P_1, P_2, P_3 and P_4 arriving in Ready Queue in the same order at time $t=0$. If BT requirements of these jobs are 4, 1, 8, 1 respectively, what is the completion time of P_1 assuming Round Robin with $TQ=1$?

PNO	AT	BT	CT
P_1	0	4	9
P_2	0	1	2
P_3	0	8	14
P_4	0	1	4

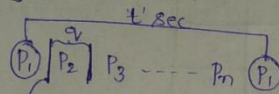


$$\therefore (CT)_1 = 9$$

23. ROUND ROBIN EXAMPLE 4

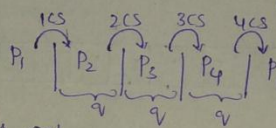
Consider 'n' processes sharing the CPU in RR fashion. If the context switching time is 's' units. what must be the time quantum 'Q' such that the no. of context switches are reduced, but at the same time each process is guaranteed to get the turn at the CPU for every 't' seconds?

Let the processes be



Context-switch time = Taken time to shift from one process to another process.

Now, let's analyze for 4 process



⇒ 4 CS/s.

⇒ 3 quantum.

∴ generalizing the above ex: $n(s) + (n-1)q \leq t$

$$\Rightarrow q \leq \frac{t - ns}{(n-1)}$$

24. LONGEST JOB FIRST

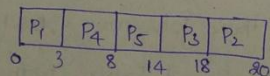
⇒ process having longest BT gets scheduled first

CRITERIA: BT

MODE: Non-preemptive

PNO	AT	BT	CT	TAT	WT	RT
1	0	3	3	3	0	0
2	1	2	20	19	17	19
3	2	4	18	16	12	12
4	3	5	8	5	0	0
5	4	6	14	10	4	4

The Gantt chart is:



25. LONGEST REMAINING TIME FIRST

PNO	AT	BT	CT	TAT	WT	RT
1	1	2	18	17	15	0
2	2	4	19	17	13	0
3	3	6	20	17	11	0
4	4	8	21	17	9	0

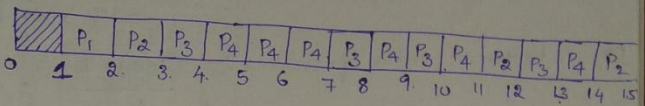
(14)atching time
context
to get the
one process

PNO	AT	BT	
1	1	2	1 0
2	2	4	3 2 1 0
3	3	8	5 4 3 2 1 0
4	4	8	7 6 5 4 3 2 1 0

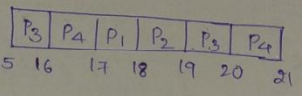
(15)

one process

The Gantt chart will be



Gantt chart continued



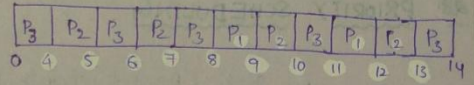
26. LONGEST REMAINING TIME FIRST GATE 2006 QUESTION

PNO	AT	BT	CT	TAT
1	0	2	12	12
2	0	4	13	13
3	0	8	14	14

what is the Average TAT using LRTF?

PNO	AT	BT	
1	0	2	1 0
2	0	4	3 2 1 0
3	0	8	7 6 5 4 3 2 1 0

Gantt chart:



$$\text{Avg. TAT} = \frac{12+13+14}{3} = \frac{39}{3} = 13 \text{ ms}$$

27. HIGHEST RESPONSE RATIO NEXT

CRITERIA: Response Ratio (RR) = $\frac{w+s}{s}$

w = waiting time for a process so far

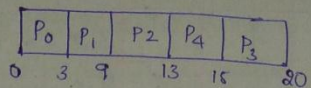
s = Service time of a process or BT.

→ HRRN not only favours shorter jobs but also limits the waiting time of longer jobs.

→ Mode: Non-preemptive.

PNO	AT	BT	CT	TAT	WT	RT
0	0	3	3	3	0	0
1	2	6	9	7	1	1
2	4	4	13	9	5	5
3	6	5	20	14	9	9
4	8	2	15	7	5	5

Gantt chart:



Now, at time $t=9\text{ms}$.

$\rightarrow P_2$ arrived at 4 and now the time is 9 so the waiting time till now = 5
 $RR_2 = \text{Response ratio of process 2} = \frac{(9-4)+4}{4} = 2.25$
 $\downarrow$
 more RR value so P_2 will be executed after 9ms of time.

$$RR_3 = \frac{3+5}{5} = 1.6$$

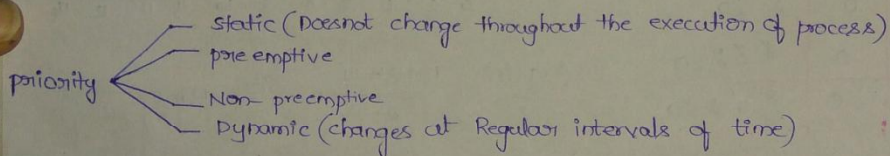
$$RR_4 = \frac{1+2}{2} = 1.5$$

At time $t=13\text{ms}$

$$RR_3 = \frac{7+5}{5} = 2.4$$

$$RR_4 = \frac{5+2}{2} = 3.5 \rightarrow P_4 \text{ will be executed after } t=13\text{ms}.$$

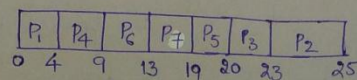
28. PRIORITY SCHEDULING



29. NON-PRE-EMPTIVE PRIORITY SCHEDULING

PNO	Priority	AT	BT	CT	TAT	WT	RT
✓ 1	2 (L)	0	4	4	4	0	0
✓ 2	4	1	2	25	24	22	22
✓ 3	6	2	3	23	21	18	18
✓ 4	10	3	5	9	6	1	1
✓ 5	8	4	1	20	16	15	15
✓ 6	12 (H)	5	4	13	8	4	4
✓ 7	9	6	6	19	13	7	7

Gantt chart:



30. PR

PNO
1
2
3
4
5
6
7

Gantt

$\Rightarrow$ Run

31. SR

PNO

1
2
3
4

Read

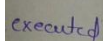
34. M

(७)

$$\begin{aligned} \text{bill now} &= 5 \\ &= (9-4) // \end{aligned}$$

executed

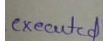
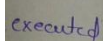
executed



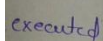
executed

executed

executed



executed



executed

executed

executed

®

Interprocess communication and synchronisation

NOU

⇒ No

 $\Rightarrow R_0$ ω

44

1

2. 1. 1.

2. INT

→ Synch

also

Entry section

Exit section &

ate using
there may

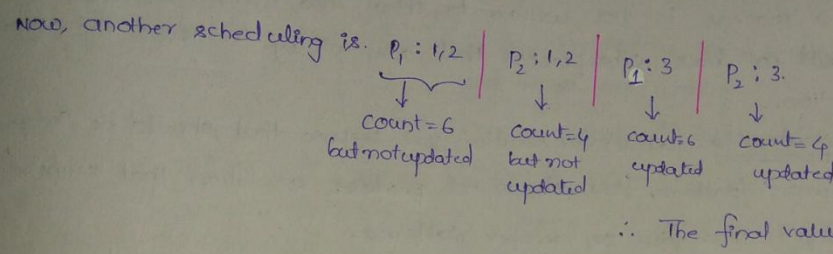
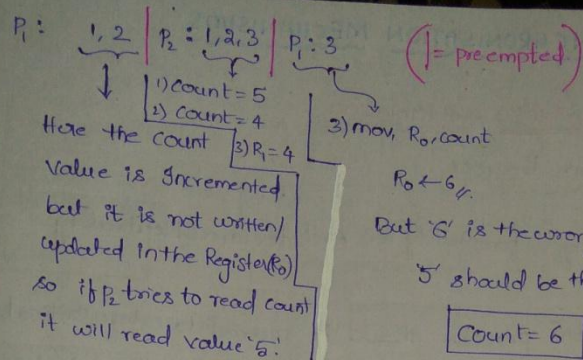
the
P₂ decrements

P₂
(1) mov R₁, count
(2) DEC R₁
(3) mov R₁, count

(mov, count, R₀)
"(INC, R₀)
(mov R₀, count)
w, let

any

because



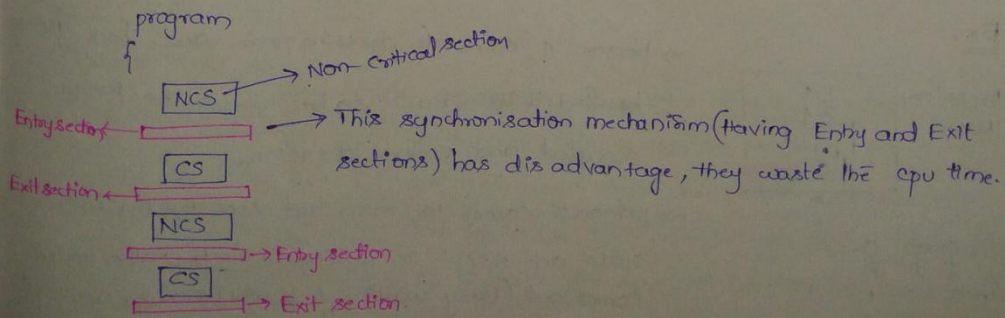
⇒ Now, all the above conditions are called "RACE CONDITIONS".

⇒ The part of the code inside the process that access the shared memory is called "CRITICAL SECTION".

⇒ Race condition means the o/p of the process depends on the order in which the process are executing. (The process which finishes first decides the final value). Therefore there must be synchronisation mechanism in such a way that if one process is accessing the critical section then other process is not allowed to access the critical section (only one process is allowed to access critical section at any instant of time).

INTRODUCTION TO SYNCHRONISATION MECHANISMS

⇒ Synchronisation mechanisms can also order the execution of the programs also.



3. CONDITIONS FOR SYNCHRONISATION MECHANISMS

Requirements for synchronization mechanisms

Primary: Mutual Exclusion, Progress

Secondary: Bounded waiting, postability (or) Architectural Neutrality.

Mutual Exclusion: Only one process should be allowed to enter critical section

Progress: If a process is not willing to enter into critical section then it should not block other process that are willing to access the critical section.

Architectural Neutrality: whatever solution you propose that should be independent of the hardware / platform. You should not propose a solution that runs only on one platform and fails on another platform.

Synchronization Mechanisms

- with Busy waiting (A process will not leave CPU until scheduler pulls it out as a result all the remaining process will be waiting).
- without Busy waiting

4. LOCK VARIABLES

→ software mechanism implemented in user mode

→ Busy waiting solution

→ can be used even for more than two processes

Entry section While (lock != 0);
Lock = 1

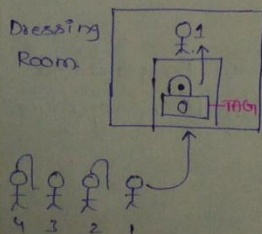
Critical section

Exit section Lock = 0

Initially Lock = 0 (which says that the CS is vacant i.e. no one is executing CS)

lock = 1 (says that the CS is occupied)

Ex:



⇒ person 1 comes to dressing room and check the tag
he find it its vacant so he changes the tag to occupied and enters into room.

⇒ person 2 comes to room and sees the tag in occupied state and keeps on waiting until the other person comes out (Busy waiting i.e. P₂ will be busy waiting for other person to come out)

5. TSL

- 1) Lo
- 2) st
- 3) cm
- 4) Jn

20

Now, say

P₁: 1 2 3

P₂: 1

P₁: 4

P₂: 1 2 3

Busy wait (Infinite loop)

Entry
1. while (lock != 0);
3. lock = 1

3. Critical Section

Exit
4. lock = 0;

→ P₁ comes initially with lock = 0.

→ while loop fails and it enters the CS and makes lock = 1

→ Now P₂ arrives and checks and finds lock = 1 (CS is occupied by some another process) and keeps on waiting.

→ Now, P₁ executes line 4 and makes lock = 0 (Room is vacant)

→ P₂ enters CS and make lock = 1

⇒ No "Mutual exclusion" is provided by this lock variable mechanism.

→ or lock = 1 (because lock can have only 2 values 0, 1)

1) While (lock != 0);

2) lock = 1

3) CS.

4) lock = 0

Entry {
1) load lock, R₀
2) cmp R₀, 0
3) Jump if Not Zero (JNZ) to step 1
4) Store #1, lock

Exit ← 5) Store #0, lock.

CS

Now, Initially lock = 0.

P₁: 1

P₂: 1, 2, 3, 4, CS

P₁: 2, 3, 4 CS

↓
P₁ will not look at the Tag

No Mutual Exclusion.

5. TSL (TEST - SET - LOCK)

1) Load lock, R₀

2) Store #1, lock

3) cmp R₀, #0

4) JNZ Step 1

If preemption occurs here then the mutual exclusion fails. The soln to this is steps 1, 2 should be executed as single unit (Atomic comparison) and b/w 1, 2 there should not be any preemption so to make this happen OS provides a new set of instruction called TSL (Test Set Lock)

Now, what TSL does is, if I write TSL Lock, R₀ then it will load the lock value at R₀ and will set the lock value to one so the two instructions above can be implemented using one single instruction.

⇒ TSL provides Mutual Exclusion, Progress, Bounded waiting depends.

⇒ TSL is dependent on Architecture.

⇒ There is a problem called "priority inversion" problem

Entry: Entry section

CS P₁

Exit section

Now, → P₁ is executing CS and priority of P₁ = 0

→ Now a process P₂ will be having high priority than P₁ will try to execute CS.

→ Now the scheduler preempts P₁ and schedule P₂.

→ Now P₂ is blocked by the Entry section and P₂ will never be executed until P₁ completes but P₁ is blocked by the scheduler and is in waiting state.

∴ There is a deadlock (Spinlock) In deadlock both the process are in blocked state but here one is in Ready state (P₂) and other is in running state (P₁).

GATE 2008 TSL QUESTION

The enter-CS() and leave-CS() functions to implement critical section of a process are realised using test-and-set instruction as follows:

```
void enter-CS()
{
    while (test-and-set(x));
}
```

```
void leave-CS()
{
    x = 0;
}
```

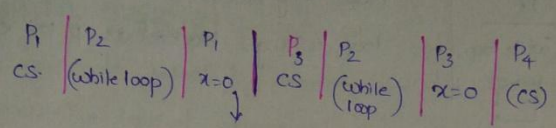
x is initialised to '0'. Now, which of the following is True?

- 1) The above solution to CS problem is deadlock free
- 2) The solution is starvation free
- 3) The process enter CS in FIFO order.
- 4) more than one process can enter CS at the same time

load the (2)
no instructions

- 1) TSL never causes Deadlock. (TRUE)
- 2) Starvation is possible

2.3



Here P_1 left the CS now instead of P_2 , P_3 has entered the CS (because it has the liberty to execute since $x=0$) and P_2 will be waiting.

P_2 will be starving all the time

- 3) FIFO Not possible (same explanation given above P_2 is never executed).
- 4) If FIFO is guaranteed there won't be any starvation
- 5) false, TSL guarantees Mutual exclusion.

7. GATE 2012 TSL QUESTION

Fetch-And-Add (x, i) is an atomic Read-modify, write instruction that reads the value of memory location x , increments the value by ' i ' and returns the old value of ' x '. It is used in the pseudo code shown below to implement Busy wait lock. ' L ' is an unsigned integer shared variable initialised to '0'. The value '0' corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

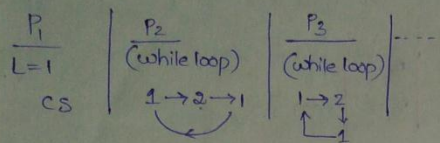
```
AcquireLock(L) {
    while (Fetch-And-Add(L, 1))
        L = 1;
}

ReleaseLock(L) {
    L = 0;
}
```

This implementation

- a) fails as ' L ' can overflow
- b) fails as ' L ' can take non-zero value when the lock is actually available
- c) works correctly but may starve some process
- d) works correctly without starvation.

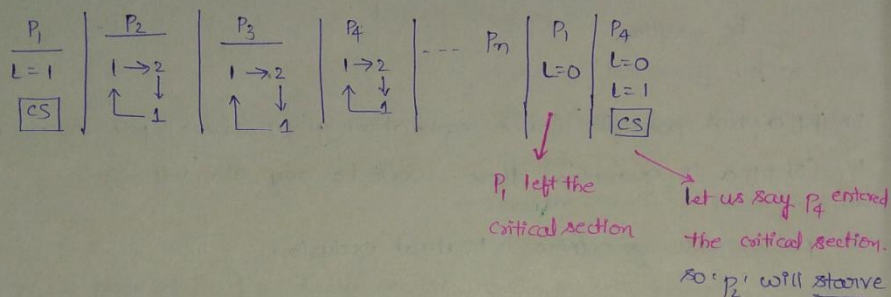
(24)



9. GATE

Consider
as given
randomly

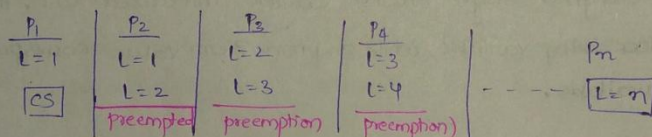
Now at some point of time 'P₁' leaves the critical section then any of P₂, P₃, P₄... P_n have the chance to enter the critical section, it need not be P₂ (FIFO order)



Now, $\frac{P_1}{L=1} \left| \begin{array}{c} P_2 \\ L=1 \\ L=2 \\ L=1 \end{array} \right| \frac{P_3}{L=1} \left| \begin{array}{c} L=2 \\ L=1 \end{array} \right| \dots \frac{P_n}{L=1}$

This occur if there is no preemption

Now, if there is preemption at this point



∴ 'L' can overflow beyond its bounds.

∴ This implementation fails 'L' can overflow.

Now,

Acquire lock(L) {

1. while(fetch and add(L,1)) ←

2. $L = 1$;

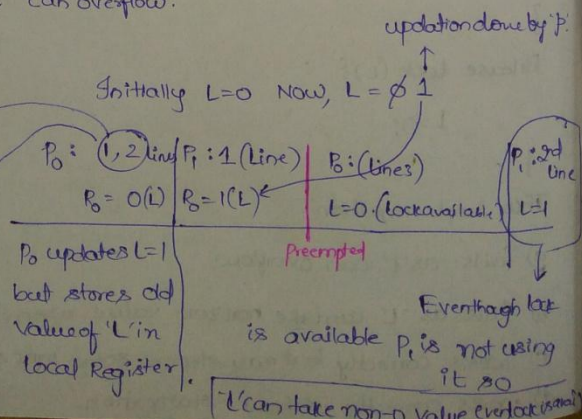
3

critical section

Release lock (L) {

3. $L=0$;

3



10. DISA

one of
Intercept

Disabl

CS

Enable

(24)

the while
ing CS.

9. GATE 2010 TSL QUESTION

Consider the methods used by process P_1 and P_2 to access ^{their} CS, whenever needed as given below. The initial values of shared boolean variables S_1 and S_2 are randomly assigned

Method used by P_1	Method used by P_2
while ($S_1 == S_2$); CS $S_1 = S_2$;	while ($S_1 != S_2$); CS $S_1 != S_2$

$\Rightarrow$ If ($S_1 == S_2$) ' P_1 ' will fall in Infinite loop (is imp)
 $\Rightarrow$ If ($S_1 != S_2$) ' P_2 ' will fall in Infinite loop.

Which of the following is true?

- a) Mutual exclusion but not progress (c) Neither ME nor progress
b) progress but not ME (d) Both ME and progress.

P_1 entered
section.
I share

S_1	S_2	
T	T	P_2 will be allowed to CS
T	F	P_1
F	T	P_1
F	F	P_2

At any point of time S_1 & S_2 can have these combinations and at any instant only one process is accessing the CS
 $\therefore$ ME is guaranteed

Now, assume both S_1 and S_2 are 'True' then ' P_2 ' has the chance to access the critical section, then assume ' P_2 ' is not interested to enter CS right now at this point of time ' P_1 ' tries to enter CS but ' P_1 ' is blocked because ' P_2 ' is turn. $\therefore$ progress is not guaranteed. (A process (P_2) which is not interested to enter CS is not allowing another process to access it.
which is interested to enter CS (P_1))

done by ' P_1 '

10. DISABLING INTERRUPTS

One of the way of handling the Synchronisation mechanism is "Disabling the Interrupts". A process should disable the interrupt before accessing the CS.

$\Rightarrow$ If we disable interrupt, preemption is not possible

$\Rightarrow$ Both ME and progress is Guaranteed.

$\Rightarrow$ Bounded waiting is not guaranteed unless you maintain the Queue.

$\Rightarrow$ Architecture Dependent

Disabling the Interrupts.

CS

Enable the interrupt

ugh lot
t using
O

11. TURN VARIABLE OR STRICT ALTERNATION METHOD

Strict Alteration Approach or Turn Variable

- Slow mechanism implemented at user mode
- Busy waiting solution.
- 2-process solution (P_0, P_1) or (P_1, P_0)

- Mutual Exclusion is guaranteed but progress is not guaranteed
- Bounded wait is present (4 units) Busy waiting is absent

For process P_0 : (int turn)

Non critical section

while (turn != 0);

CS

turn = 1;

Non CS.

For process P_1 :

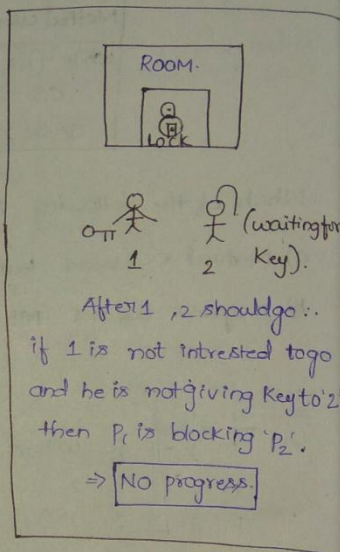
Non-CS.

while (turn != 1);

CS

turn = 0;

Non-CS.



⇒ cannot be implemented for more than 2 process.

Global Variable

P_0 (int turn)

P_1

Non CS

while (turn != 0);

CS

turn = 1;

Non CS

Non-CS

while (turn != 1);

CS

turn = 0;

Non-CS.

If turn = 0 ⇒ P_0 enters the CS.

If turn = 1 ⇒ P_1 enters the CS.

Now, Increase of Lock Variables

LV

$L=0$ - P_0, P_1 has chance to enter CS.

$L=1$ - No chance

Turn Variable

0 - P_0 can enter the CS.

1 - P_1 can enter the CS.

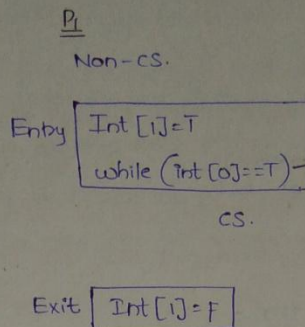
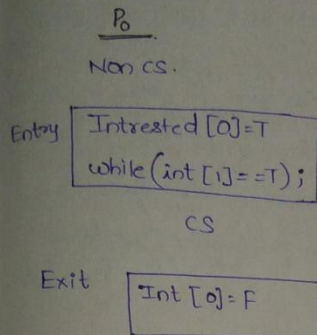
Now, let's try by adding another variable

Now, let us consider another variable. Interested[0] = T → I'm interested to enter CS
Interested[1] = F → I'm (P_1) not interested to enter CS.

(26)

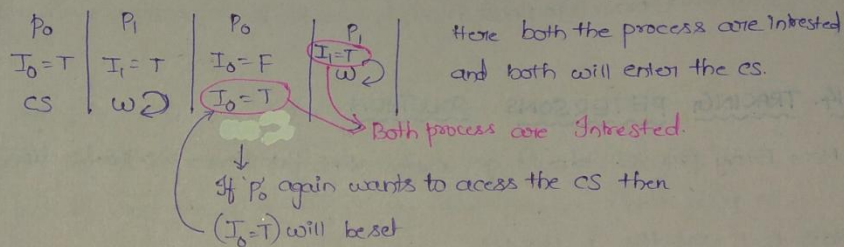
Now Before entering the critical section a process should say whether it is interested/not interested to enter the critical section.

(27)



We are checking whether the other process is also interested/not because it might already be present in CS / it might have shown interest previously.

⇒ By the above manipulations ME, progress is guaranteed, Bounded wait (not possible)
Deadlock is possible



2. INTERESTED VARIABLE

Same concept in video 11. (Repeated).

3. PETERSON SOLUTION

- Two mechanism at user mode
- Busy waiting solution
- 2-process solution (P₀, P₁)
- It uses (turn + interested) variable
- platform independent.
- No overhead to the operating system (Everything executes in user mode)

Entry

Critical section

Exit

```

#define N 2
#define TRUE 1
#define FALSE 0
int interested[N] = FALSE;
int turn;
void Entry_Section(int process)
{
    int other;
    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while (interested[other] == TRUE && TURN == process);
}
void Exit_Section(int process)
{
    interested[process] = FALSE;
}

```

14. TRACING PETERSONS SOLUTION

Now, Every process should go across Entry section → Critical section → Exit section

Now, P_0, P_1 are the 2 processes

Ex 1:

interested $\begin{matrix} 0 & 1 \\ \boxed{F} & \boxed{T} \end{matrix}$
 Turn $\boxed{0} \ 1$

$P_0: 1 \ 2 \ 3 \ 4 \ 5 \ CS \mid P_1: 1 \ 2 \ 3 \ 5$ Both ME, progress are guaranteed.
 $P_0: 6 \mid P_1: 5 \ CS$ loop.

Ex 2: interested $\begin{matrix} 0 & 1 \\ \boxed{T} & \boxed{T} \end{matrix}$
 Turn $\boxed{1} \ 0 \ 1$

$P_0: 1 \ 2 \ 3 \mid P_1: 1 \ 2 \ 3 \ 4 \ 5 \mid P_0: 4 \ 5 \mid P_1: 5 \ CS$
 preempted loop. loop.

One thing clear here is the process that executes line no. 4 has the chance to enter/will enter the CS first.

⇒ Generic solution → platform Independent.

```

#define N 2
#define TRUE 1
#define FALSE 0
int interested[N] = FALSE;
int turn;
void Entry_Section(int process)
{
    1) int other;
    2) other = 1 - process;
    3) interested[process] = TRUE;
    4) turn = process;
    5) while (interested[other] == TRUE && TURN == process);
}
void Exit_Section(int process)
{
    6) interested[process] = FALSE;
}

```

⇒ The solve
 solve #

→ Sleep me
 will go
 its use
 sleep on

16. SLEEP

Sleep() &

then it w
 wake up

⇒ The mos
 problem

→ produces

and th

Consum

has c

⇒ Both p

define

define

void proc

{

int

wh

it

if

in

{

}

}

(28)

15. SYNCHRONISATION MECHANISMS WITHOUT BUSY WAITING

(29)

⇒ The solutions that we have seen till now are subjected to Busy waiting, to solve this we use sleep and wakeup mechanism.

→ Sleep means if any process wants CS and if it is not available that process will go and sleep and as soon as the other process which is using CS finishes its usage it is going to go and wakeup which is sleeping. so using sleep and wakeup we can solve the problem of CPU wastage.

16. SLEEP AND WAKE

sleep() and wake() are two system calls. If any process calls sleep() then it will go and get blocked and if any process calls wake() it will wake up one of the blocked process.

⇒ The most common application of sleep and wake is producer and consumer problem.

→ producer means (if there are 2 process & one process is producing something and the other process is consuming something then it is called producer consumer problem, (one process writes & the other process reads what other has written)).

⇒ Both producer and Consumer will be using Buffers.

```
#define N 100 // slots in Buffer
#define count = 0 // items in Buffer
```

```
void producer(void)
```

```
{
    int temp;
    while (TRUE) {
        item = produce_item();
        if (count == N) sleep();
        insert_item(item);
        count = count + 1;
        if (count == 1) wakeup(consumer);
    }
}
```

```
void consumer(void)
```

```
{
    int item;
    while (TRUE) {
        if (count == 0) sleep();
        item = remove_item();
        count = count - 1;
        if (count == N - 1)
            wakeup(producer);
        consume_item(item);
    }
}
```

Some gate questions are when is each one going to wakeup other

The problem with producer consumer problem is if the consumer reads (count == 0) then it will go to sleep, assume when the consumer is about to sleep it got preempted, now the producer produces/adds item to the Buffer and it will increase count from the value 0 and try to wakeup the consumer thinking that the consumer is sleeping. Now whenever the consumer (after preemption) gets scheduled he will go to sleep(), and when the Buffer is full the producer thinks that the consumer will wake up and producer also sleeps. ultimately both end up in sleeping forever. so we need to have "Integer" that counts the no. of wakeups and if any process wakes up it will decrement the variable value and the variable is called Semaphore.

17. INTRODUCTION TO SEMAPHORES

- ⇒ Semaphore concept was proposed by DIJKSTRA in 1965
- Implementing Semaphore at the user level will be dangerous & inconsistent.
- ⇒ Variables on which Read, Modify, and update happens atomically in kernel mode (No preemption)
- They are two types
 1. counting semaphore
 2. Binary Semaphore (Mutexes)

18. COUNTING SEMAPHORES

The counting semaphore is a structure like this

struct Semaphore

```

{
    int value;  → Denotes the no. of process that can access the CS at the
    Queue type L; → same time some CS are/can be accessed by many
    process.
    down(semaphore s)
    {
        s.value = s.value - 1;
        if (s.value < 0)
        {
            put process (pcb) in L;
            sleep();
        }
        else
            return;
    }
    up(semaphore s)
    {
        s.value = s.value + 1;
        if (s.value > 0)
        {
            select a process from L; wakeup();
        }
    }
}
  
```

→ Represent the set of process that are blocked whenever they try to enter the critical section because they don't have permission.

Entry Se

Exit Se

→ when a pr
increments
and Inca
is less th
under star
in waiting
wake the

⇒ To get
one pr

19. PROBLE

A counting
operation

S=10,

20. BINAR

→ can ha

→ The Bin

ads (30)
to sleep
and

mon
(after
e full
leeps.
that
ement

istent.

nel

as at the
many

ed when
se they

Entry Section

Critical Section

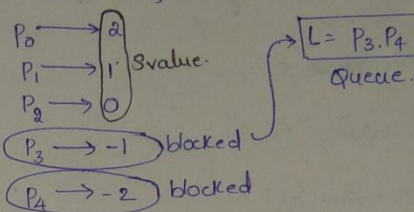
Exit Section.

Now say three processes can access the CS at a time say P_0, P_1, P_2 . (31)

Now the counting Semaphore (S) will be initialised to '3' whenever a process enters CS then it decrements the Semaphore value and access the CS.

when a process exists then it increments the Semaphore value and increase if semaphore value is less than Zero then it can understand that some processes are in waiting list and it is supposed to wake the process and come out.

S = 3 Initially



⇒ To get mutual Exclusion set the Initial value of Semaphore = 1 then only one process can access the CS at a time.

Decrement	Increment
Down	up
P	V
wait	signal

19. PROBLEMS ON COUNTING SEMAPHORE

A counting semaphore was initialised to 10. The 6P(wait) and 4V(signal) operations are computed on this semaphore. What is the result

$$S=10, \quad 6P \text{ and } 4V \\ \Rightarrow 10 - 6 + 4 \\ = 8 //$$

$$\textcircled{2} \quad S=7, \text{ then } 20P, 15V \\ \Rightarrow 7 - 20 + 15 \\ = 29 - 20 \\ = 9 //$$

20. BINARY SEMAPHORE OR MUTEXES

→ Can have only 2 values 0 and 1.

→ The Binary Semaphore is a structure

struct Bsemaphore

```
{
    enum value(0,1);
    Queue type L;
}
```

'L' contains all pcb's corresponding to process got blocked while performing down operation unsuccessfully.

down(Bsemaphore s)

```
{
    if(s.value == 1)
    {
        s.value = 0;
    }
    else
    {
        put the process (pcb) in s.L;
        sleep();
    }
}
```

up(Bsemaphore s)

```
{
    if(s.L is empty)
    {
        s.value = 1;
    }
    else
    {
        select a process from s.L;
        wakeup();
    }
}
```

21. GATE 92 QUESTION ON MUTEX

mutex = 1

P_i = 10

P_i = 1, 2, ..., 9

while(1)

{

down(mutex)

up(mutex)

<cs>

<cs>

up(mutex)

up(mutex)

How many process can be present in cs at max?

Now

mu

P_i =

whi

f

D

}

mate

mate

P₀

while

{

a = 0 → P

③ → P

① → P

② → P

④ → P

}

23. M

Mute

P₀

while(th

{

1. pca

2. pci

3. vca

4. vci

32

Now,

mutex = 1

$P_1 = 1, 2, \dots, 9$ $\downarrow$
 P_1

while(1)
{

down(mutex) $\Rightarrow$ mutex = 0;

<CS> (P_1) { $P_2, P_3, P_4, P_5, \dots, P_9$ }

up(mutex)

}

Now the other process

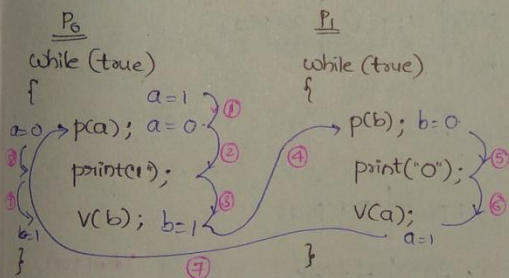
{ $P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9$ } are

in Queue.

$\therefore$ Now Initially P_1 enters the critical section Now, as P_{10} executes in reverse order (2 up's) it allows all the process one by one at to enter the CS. $\therefore$ { $P_1, \dots, P_9$ } can be present in CS at any time.

22. MUTEX Example

mutex a, b; a=1; b=0;



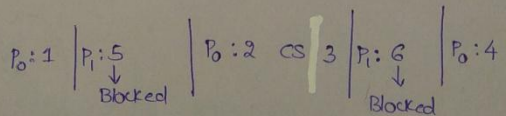
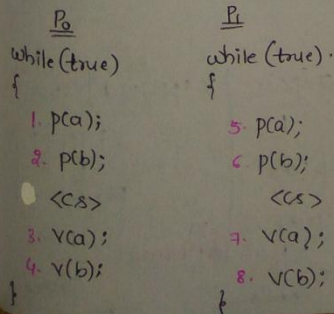
101010-...

Ans: 10101010-...

Here the mutexes are not just used for mutual exclusion but also for the order in which we schedule the process.

23. MUTEX Example 1

mutex a, b; a=1, b=1;



P_1 : CS

ME is guaranteed.

24. GATE 2013 QUESTION ON MUTEXES

(39)

Three concurrent process x, y, z execute 3 different code segments that access and update certain shared variables. process x execute the 'p' operation on semaphores a, b, c and process y executes 'p' operation on semaphores 'b' and 'c' and d; process z executes p operation on semaphores c, d, a before entering the respective code segments. After completing the execution of its code segments each process invoke 'V' operation on its 3 semaphores. All are Binary semaphores and are initialised to one. which of the following operations represents deadlock free order of invoking 'p' operation by the process.

a)

X: p(a) p(b) p(c)

Y: p(b) p(c) p(d)

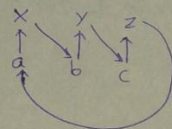
Z: p(c) p(d) p(a)

X: p(a) | Y: p(b) | Z: p(c)

X: p(b) | Y: p(c) | Z: p(c) p(a)

DEADLOCK

Blocked Blocked Blocked



X → Y → Z = deadlock

b) X: p(b) p(a) p(c)

Y: p(b) p(c) p(d)

Z: p(a) p(c) p(d)

X: p(b) | Y: p(b) | Z: p(a)

X: p(a) |

Wait state

c) X: p(b) p(a) p(c)

Y: p(c) p(b) p(d)

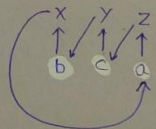
Z: p(a) p(c) p(d)

X: p(b) | Y: p(c) | Z: p(a)

X: p(a) | Y: p(b) | Z: p(a) p(c)

DEADLOCK

B B B



DEADLOCK

d) X: p(a) p(b) p(c)

Y: p(c) p(b) p(d)

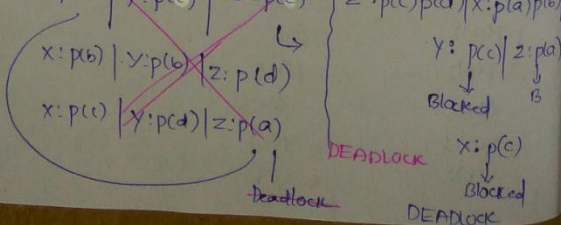
Z: p(c) p(d) p(a)

X: p(a) | Y: p(c) | Z: p(c)

X: p(b) | Y: p(b) | Z: p(d)

X: p(c) | Y: p(d) | Z: p(a)

DEADLOCK



25. GATE

Let m1
process

a) Thra

Now,

Mutual E

⇒ P0:

P2:

26. GATE

Pi: 0, 1, 2, n

M1: 0, 1, 2, n

Pi: p(m1)

<CS>

v(m1): v(m1)

(34)

that access and
on semaphores
dd; process z
ve code segments
'V' operation
sed to one.
voking 'P' operation

25. GATE 2000 QUESTION ON MUTEX

(35)

Let $m[0] \dots m[4]$ be mutexes (binary semaphores) and $p[0] \dots p[4]$ be process. Suppose each process $p[i]$ executes the following.

wait($m[i]$); wait($m[(i+1) \bmod 4]$); } This could cause
----- Critical Section
release($m[i]$); release($m[(i+1) \bmod 4]$); }

a) Thrashing (b) Deadlock (c) Starvation (d) None of the above
but not DL

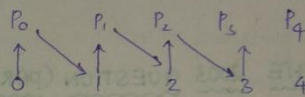
Now, $P_0: p(m[0]) \ p(m[1]) \Rightarrow P_0$ executes down operation on $m[0]$ and $m[1]$.

$P_1: p(m[1]) \ p(m[2])$

$P_2: p(m[2]) \ p(m[3])$

$P_3: p(m[3]) \ p(m[0])$

$P_4: p(m[4]) \ p(m[1])$



$P_0 - P_1 - P_2 - P_3 - P_0$

$\therefore$ Deadlock is possible

Mutual Exclusion is not possible P_0 and P_2 can enter CS at same time

$\Rightarrow P_0: p(m[0]) \ p(m[1]) \Rightarrow CS$
 $P_2: p(m[2]) \ p(m[3]) \Rightarrow CS$ } P_0, P_2 are operating on 2 different mutexes
so they can enter critical section at same time

(P_0, P_2)
 (P_1, P_3)
 (P_4, P_2)
 (P_4, P_3) } These processes can enter the CS at the same time.
so at any time max 2 process can be in CS.

26. GATE 96 QUESTION ON DINING PHILOSOPHER PROBLEM

$P_i: 0, 1, 2, 3$

$M_i: 0, 1, 2, 3$

$P_i: p(m_i), p(m_{(i+1) \bmod 4});$

<CS>

$v(m_i); v(m_{(i+1) \bmod 4});$

$P_0: p(m_0) \ p(m_1)$

$P_1: p(m_1) \ p(m_0)$

$P_2: p(m_2) \ p(m_3)$

$P_3: p(m_3) \ p(m_2)$

This sequence
results in deadlock
we have seen before
Now to avoid DL
let us modify the

d) $x: p(a) p(b)$
 $y: p(c) z: p(a)$
↓
Blocked
 $x: p(c)$

Now, after modifying the sequences will be

$P_0: p(m_0) \quad p(m_1)$

$P_1: p(m_1) \quad p(m_2)$

$P_2: p(m_2) \quad p(m_3)$

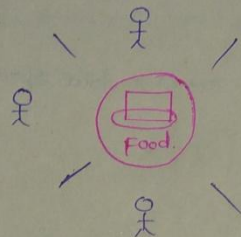
$P_3: p(m_0) \quad p(m_3)$

$P_0: p(m_0) \mid P_1: p(m_1) \mid P_2: p(m_2) \mid P_3: p(m_0)$

↓ Blocked

$P_0: p(m_1) \mid P_1: p(m_2) \mid P_2: p(m_3)$ CS release mutex 2,3.

↓ Blocked ↓ Blocked



⇒ Make one person to take right spoon first and then left spoon
 ⇒ Make all others to take left spoon and then right spoon first
 ⇒ To break the Deadlock (GATE)

27. GATE 2003 QUESTION (PART 1)

Suppose we want to synchronise 2 concurrent processes 'P' and 'Q' using Binary semaphores S and T. The code for the process 'P' and 'Q' is shown below.

process P:

```
while (1) {
    W: print '0'
    print '0';
    x:
}
```

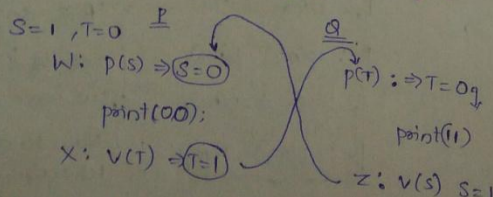
process Q:

```
while (1) {
    Y: print '1';
    print '1';
    z:
}
```

Synchronization stmnts
 can be inserted only at
 points w, x, y, z

a) which of the following will always lead to an output starting with 001100110011?

- A) (P(S) at W), (V(S) at X), (P(T) at Y), (V(T) at Z) (S and T) initially 1
- B) (P(S) at W), (V(T) at X), (P(T) at Y), (V(S) at Z) (S=1, T=0) initially.
- C) (P(S) at W), (V(T) at X), (P(T) at Y), (V(S) at Z) (S=T=1) initially
- D) (P(S) at W), (V(S) at X), (P(T) at Y), (V(T) at Z) (S initially 1, T initially 0)



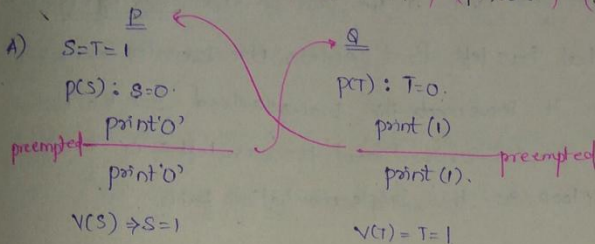
∴ Alternate 001100110011... will be printed.

(36)

Q8: GATE 2003 QUESTION (PART 2)

b) which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where 'n' is odd.

- A) $(p(s) \text{ at } W) (v(s) \text{ at } x) (p(t) \text{ at } y) (v(t) \text{ at } z) (S=T=1)$
 B) $(p(s) \text{ at } W) (v(t) \text{ at } x) (p(t) \text{ at } y) (v(s) \text{ at } z) (S=T=1)$
 C) $(p(s) \text{ at } W) (v(s) \text{ at } x) (p(s) \text{ at } y) (v(s) \text{ at } z) (S=1)$
 D) $(v(s) \text{ at } W) (v(t) \text{ at } x) (p(s) \text{ at } y) (p(t) \text{ at } z) (S=T=1)$



$$010 = 01^n0 \Rightarrow n=1$$

$$= n=\text{odd}$$

$\therefore 01^n0$ substring is possible

Similarly B, D also produce the sequence $01^n0/10^n1$. $\therefore$ 'C' is the Ans.

Q9. GATE 06 QUESTION ON BARRIER (PART 1)

$\Rightarrow$ Barrier is also one kind of synchronization mechanism.

Barrier is a synchronization construct where a set of processes synchronises globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all process leave the barrier. Let the no. of process in the set be three and S be a Binary semaphore with the usual 'P' and 'V' functions. Consider the following 'C' implementation of barrier with line nos shown on left:

```
Void barrier (void){
1. p(s);
2. process-arrived++;
3. v(s);
4. while (process-arrived != 3);
5. p(s);
6. process-left++;
7. if (process-left == 3){
8. process-arrived = 0;
9. process-left = 0;
```

10. }

11. v(s);

}

The variables process-arrived, process-left are shared among all process and are initialised to '0'. In a concurrent program all the three process call barrier function when they need to synchronise Globally.

A) The above implementation of barrier is incorrect. which one of the following is true? (38)

- a) The barrier implementation is wrong due to the use of Binary Semaphore S.
- b) The barrier implementation may lead to deadlock if 2 barrier invocations are used in immediate succession. $\rightarrow$ NOT always
- c) Lines 6 to 10 need not be inside CS.
- d) The barrier implementation is correct if there are only 2 process instead of three

$\Rightarrow$ consider all the process arrived and all the process are leaving out now while leaving all the process that has left first enters the barrier again (because 2 successions are given) then it increments the process arrived but the last process leaving the CS makes process-arrived and process-left $\neq 0$ even though a process is executing while loop so, this implementation fails.

30. GATE 06 QUESTION ON BARRIER (PART 2)

How do you fix the above problem?

The main problem is before making process-arrived and process-left $= 0$ the value is incremented because of the process that has left critical section has arrived again

The solution is the first process which is reentering should wait until the process arrived $= 0$

31. GATE 2008 QUESTION ON IMPLEMENTING COUNTING SEMAPHORE USING MUTEX

The 'p' and 'v' operations on counting semaphores, where S is counting semaphore are defined as follows.

P(S): $S = S - 1$;

if ($S < 0$) then wait;

V(S): $S = S + 1$;

if $S < 0$ then wake up process waiting on S.

Assume that P_b and V_b the wait and signal operations on Binary semaphores are provided. Two Binary semaphores X_b and Y_b are used to implement the semaphore operations P(S) and V(S) as follows:

P(S): $P_b(X_b)$;

$S = S - 1$;

if ($S < 0$)

These cannot be interchanged (deadlock) $\left\{ \begin{array}{l} V_b(Y_b) \\ P_b(Y_b) \end{array} \right.$

else V_b

$\Rightarrow$ The initial process w of 'A' B

$\Rightarrow$ Now, every $\therefore$ The v

OR

The counting S

$S = 4$

L

$\therefore \left\{ \begin{array}{l} \text{Let } X_b = \\ L_1 \end{array} \right.$

This contains process that to get access

$\Rightarrow$ Now to g

$\Rightarrow$ Now, Let 3 will be in L_1

following is

(38)

aphore S.

ations are

instead of three

gocet now

gain (because

he last

eventhough

o the

tion has

still the

(NB) MUTEX

naphore

semaphores

ment

$P(s): P_b(x_b);$
 $S = S - 1;$
 if ($S < 0$) {
 $V_b(x_b);$
 $P_b(y_b);$
 else $V_b(x_b);$
 }

These cannot be interchanged (deadlock)

$V(s): P_b(x_b);$
 $S = S + 1;$
 if ($S = 0$) $V_b(y_b);$
 $V_b(x_b);$

(39)

The initial values of x_b and y_b are respectively?

- Ⓐ 0 and 0 Ⓑ 0 and 1 Ⓒ 1 and 0 Ⓓ 1 and 1

⇒ The initial value of x_b cannot be Zero because if it is Zero all the process will get blocked. No process can access the critical section. of $\therefore$ A/B are eliminated.

⇒ Now, every process that performs down on y_b gets blocked (should be blocked)

$\therefore$ The value of y_b should be '0'

$\therefore$ Ans = 1, 0

OR

The counting Semaphore has 2 parts $\rightarrow$ value of the semaphore itself

$\rightarrow$ list (queue having blocked process).

$S = 4$
 L []

Now, we should prevent two process accessing the Semaphore 'S' at the same time $\therefore$ we use 2 semaphore (mutex semaphore to protect the counting Semaphore 'S' and the list (queue)

$\therefore \left\{ \begin{array}{l} \text{let } x_b = \{0, 1\} \\ L_1 \text{ []} \end{array} \right\} \left\{ \begin{array}{l} y_b = \{0, 1\} \\ L_2 \text{ []} \end{array} \right\} \rightarrow$ This contains set of process that are blocked while accessing 'S'

This contains set of process that are waiting to get access to Semaphore 'S'

⇒ Now to get access to the i th semaphore 'S' it should down the x_b and

$\therefore$ x_b Initially should be 1

⇒ Now, Let 4 process executed the CS then 1 process will be in CS and remain 3 will be in L_1 and P_5 (5th process) try to access the Semaphore it will get blocked and will be in L_2 $\therefore$ Blocking occurs when we perform down on '0' $\therefore$ $y_b = 0$

32. GATE 2010 QUESTION ON MUTEXES

The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialised as $S_0=1, S_1=0$. (40)

<u>process P_0</u>	<u>process P_1</u>	<u>process P_2</u>
while (true)	wait(S_1)	wait(S_2)
{	Release(S_0);	Release(S_0);
1) wait(S_0)		
2) print '0';		
3) Release(S_1);		
4) Release(S_2);		
}		

How many times P_0 will print 0?

- (a) Atleast twice (b) Exactly twice (c) Exactly thrice (d) Exactly once

Given, $S_0=1, S_1=0, S_2=0$ $S_i < 1 \Rightarrow S_0=1, S_1=0, S_2=0$ } Try with both combinations
 $S_0=1, S_1=0, S_2=0$

$S_0=1, S_1=0, S_2=0$

$P_0: 1 \ 2 \ 3 \ 4$	$P_1: 1 \ 2$	$P_2: 1 \ 2$	$P_0: 1 \ 2 \ 3 \ 4$
$P_1: 1 \ 2$	$P_0: 1 \ 2 \ 3 \ 4$		

0 (only printed)

$P_0: 1 \ 2 \ 3 \ 4$ | $P_2: 1 \ 2$

$P_0: 1 \ 2 \ 3 \ 4$ | $P_0: 1$

(00) 2 0's printed.

(000) $\Rightarrow$ 3 times

$\therefore$ Almost 3 times

33. GATE 2013 QUESTION ON COUNTING SEMAPHORE

W: int x=0 x: p(s)
 p(s)
 $x = x+1$
 v(s)
 Y: p(s) Z: p(s)
 $x = x-2$
 v(s)

's' is a semaphore whose value is initialised to two. What is the max value of 'x'.

Here the Semaphore is counting semaphore

(41)

Now, there are only two process that are incrementing the value of 'x' (i+1) and (j+1) by 'w' and 'x'.

least value = -4 // (x-2) (x-2) by y, z.

34. GATE 1995 QUESTION ON CONCURRENT PROCESS AND SEMAPHORES

Draw the precedence Graph for statements S_1 to S_9 .

var: a, b, c, d, e, f, g, h, i, j, k : Semaphore; All are mutexes & initialised to '0'

begin

cobegin Enable a, b $\Rightarrow S_2, S_3$ can be executed.

begin: S_1 ; $v(a)$; $v(b)$ end;

begin: $p(a)$; S_2 ; $v(c)$; $v(d)$; end;

begin: $p(c)$; S_4 ; $v(e)$; end;

begin: $p(d)$; S_5 ; $v(f)$; end;

begin: $p(e)$; $p(f)$; S_7 ; $v(k)$; end;

begin: $p(b)$; S_3 ; $v(g)$; $v(h)$; end;

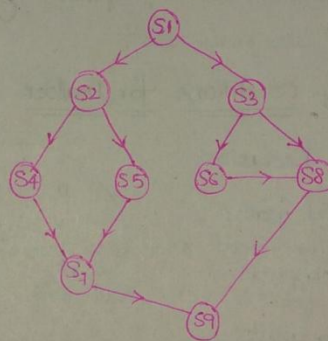
begin: $p(g)$; S_6 ; $v(i)$; end;

begin: $p(h)$; $p(i)$; S_8 ; $v(j)$; end;

begin: $p(j)$; $p(k)$; S_9 end;

coend;

end.



3. DEADLOCKS

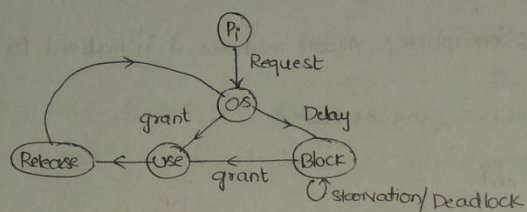
(42)

1. INTRODUCTION TO DEADLOCKS

→ A set of processes are said to be in deadlock, if they wait for happening of an event caused by others in the same set.

→ Starvation is long waiting

→ Deadlock is Infinite waiting



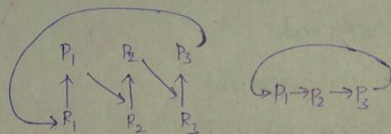
Necessary Conditions for Deadlock

→ Mutual Exclusion

→ Hold and wait:

→ No preemption

→ Circular wait



2. GATE 1997 QUESTION AND MORE EXAMPLES

A system is having 3 user processes each requiring 2 units of resource 'R'. The minimum number of units of 'R' such that no deadlock will occur.

a) 3 b) 5 c) 4 d) 6

⇒ There are 3 processes and each require 2 units $\Rightarrow 3 \times 2 = 6$ units, so, if there are 6 units of Resource type 'R' then there won't be any deadlock. (But in question they have asked for minimum no. of units.

⇒ Now we might think that 2 units is the minimum. $\Rightarrow P_1$ uses 2 resources and releases them then P_2 takes resource and releases it and P_3 takes the resource and releases it. But what if P_1 takes one unit of Resource and P_2 takes for one unit of Resource then P_1 waits for P_2 and P_2 waits for P_1 so deadlock occurs.

⇒ Now if P_1 has
 P_1 waits for

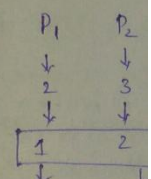
⇒ Now, if

⇒ The max

⇒ The min

Find the

⇒ Now consid



P_1 will be
waiting
for one
more source

Let us say

(7)

=

=

Min

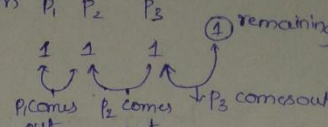
⇒ Now th

And

42

Now if I have 3 process say P_1 - 1 unit, P_2 - 1 unit, P_3 - 1 unit and here P_1 waits for P_2 , P_2 waits for P_3 and P_3 waits for $P_1 \Rightarrow$ DEADLOCK (43)

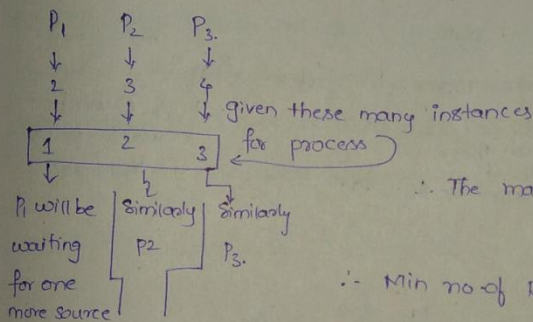
opening

Now, if I have 4 units it will solve my problem P_1 P_2 P_3


The max. no. of Resource that could cause deadlock = 3

The min no. of units reqd. that No deadlock will occur = 4 $\Rightarrow$ Now to find this find the max no. of units that could cause deadlock and add 1 to it.

Now consider process P_1 P_2 P_3 Requires } then how many resources are
 2 3 4 Resources } req'd in minimum so that
 there won't be DL?



The max no. of Resources that could cause deadlock = $1+2+3 = 6$

Min no. of Resources that will not cause DL = $6+1 = 7$

ace 'P' The

Let us say P_1 P_2 P_3 ... P_n
 x_1 x_2 x_3 ... x_n

$(x_1-1) + (x_2-1) + (x_3-1) + \dots + (x_n-1)$ These are the max no. of resources that when allocated can lead to deadlock

$$= (x_1 + x_2 + x_3 + \dots + x_n) - n$$

$= \left(\sum_{i=1}^n x_i \right) - n \Rightarrow$ It is the value of max no. of resources that could cause deadlock.

resources
 the resource
 and
 arts for P1

Min. no. of Resources that cannot/will not cause deadlock

$$= \left(\sum_{i=1}^n x_i - n \right) + 1$$

Now they might ask the Question in Reverse way a Resource has '6' instances and Each process requires P1: 2 instances what is the max no. of

Given $R=6$ instances $P_i = 2$ instances.

(4)

Min. no. of process which result in deadlock = $P_1 P_2 P_3 P_4 P_5 P_6$

Instances: 1 1 1 1 1 1

↓
wait for
one instance

∴ The max no. of process that can operate without deadlock = $5 \times (6-1)$

→ Total instances.

$R=6$, $P_i=3$. (Each require 3 instances so give 2 instances for them and make them wait for
→ NO. of instances needed by the process (each process). one more instance)

$P_1 P_2 P_3$ ∴ Then three process results in DL

2 2 2. ∴ No. of process that doesn't cause DL = 2.

$R=100$, $P_i=2$ instances. then

Min no. of process reqd for Deadlock = 100. $P_1 P_2 P_3 \dots P_{100}$

Max. no. of process reqd such that there

won't be DL = $100-1 = 99$

1 1 1 ... 1 ⇒ All will be waiting for one instance

$R=100$, $P_i=3$ instances

Min no. of process reqd for Deadlock = $\frac{P_1}{2} \frac{P_2}{2} \dots \frac{P_{100}}{2} = 50$.

Max no. of process reqd such that there won't be DL = 49

$R=100$, $P_i=4$ instances.

$n \times (3) = 100$

⇒ $n = \left\lceil \frac{100}{3} \right\rceil$

$n = 34$ process

Min no. of process lead to DL = 34

Max no. of process don't lead to DL = 33.

(44)

3. GATE 1992 QUESTION

A computer system has 6 tape drives, with n process computing for them. Each process needs 3 tape drives. The max value of n for which the system is guaranteed to be deadlock free

- a) 2 b) 3 c) 4 d) 1

$$= 5 \times (6-1)$$

$\Rightarrow R=6, P_i = \text{No. of instances needed by each process} = 3.$

Now Give 2 instances to each process.

$\Rightarrow$	$\frac{P_1}{2}$	$\frac{P_2}{2}$	$\frac{P_4}{2}$	$\therefore \text{Min no of process that can cause deadlock} = 3$ $\text{Max no of process that results in DL free} = 2.$
	$\downarrow$	$\downarrow$	$\downarrow$	
	wait for 1 instance	=	=	

them and
em wait for
instance)

= 2.

4. GATE 1993 QUESTION ON MINIMUM RESOURCES REQUIRED

Consider a system having m resources of the same type. These resources are shared by 3 process A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur?

$m = ?$	<u>A</u>	<u>B</u>	<u>C</u>
	3	4	6

- a) 7 b) 9 c) 10 d) 13.

Max m where there will be dead lock = $2+3+5 = 10$.

Min no. of Resources where there will not be dead lock = 11. (anything > 11)

5. GATE 2005 QUESTION

Suppose n processes $P_1, \dots, P_n$ share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process P_i is S_i where $S_i > 0$. Which one of the following is a sufficient for ensuring that deadlock does not occur.

- a) $\forall i, S_i < m$ (b) $\forall i, S_i < n$ (c) $\sum_{i=1}^n S_i < (m+n)$ (d) $\sum_{i=1}^n S_i < (m \times n)$

There are m Resource units, n -process $P_1, P_2, \dots, P_n$

$P_1 - S_1$ units of Resource

$P_2 - S_2$

$P_n - S_n$ units of Resource

$$\begin{array}{c} P_1 \quad P_2 \quad P_3 \quad \dots \quad P_n \\ S_1-1 \quad S_2-1 \quad S_3-1 \quad \dots \quad S_n-1 \\ \hline = \left(\sum_{i=1}^n S_i - n \right) \end{array} \quad \therefore m \leq \sum_{i=1}^n S_i - n$$

For deadlock to occur

For DL not to occur.

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

6. GATE 06 QUESTION ABOUT NECESSARY CONDITION FOR DEADLOCK

(16)

Consider the following snapshot of a system running 'n' process. process 'i' is holding ' x_i ' instances of a Resource 'R' for $1 \leq i \leq n$. currently all instances of 'R' are occupied. Further, for all 'i' process 'i' has placed a request for an additional y_i instances it already has. There are exactly two process 'p' and 'q' such that $y_p = y_q = 0$. which of the following can serve as necessary condition to guarantee that the system is not approaching a deadlock?

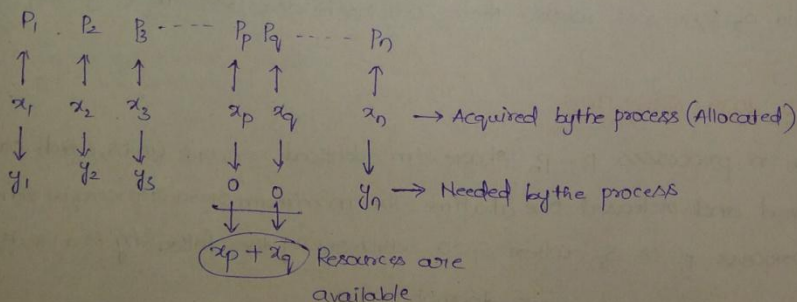
- a) $\min(x_p, x_q) < \max y_k \quad k \neq p, q$
- b) $x_p + x_q \geq \min_{k \neq p, q} y_k$
- c) $\max(x_p, x_q) > 1$
- d) $\min(x_p, x_q) > 1$

R=9

<u>P₁</u>	<u>P₂</u>	<u>P₃</u>	
5	4	6	⇒ Requested
3	4	2	⇒ allocated
2	0	4	= Needed

4 Resources are released now the necessary condition to break the Dead lock is the freed Resource should serve atleast one of the process that is in need of Resource then the Deadlock will be Broken.

Now, Generalising the above concept



$(x_p + x_q)$ should satisfy atleast one of the Remaining process need (atleast it should satisfy the min need of the among the Remaining process.

$$\therefore x_p + x_q \geq \min_{k \neq p, q} y_k$$

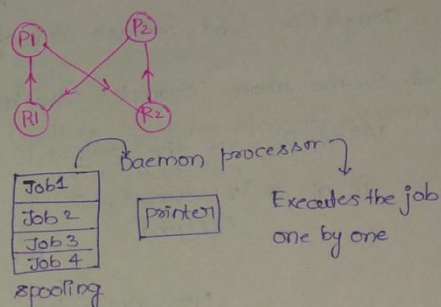
7. DEADLOCK HANDLING MECHANISMS

Strategies for Handling Deadlock

- 1) Deadlock Ignorance - Both windows and linux uses Deadlock Ignorance
- 2) Deadlock prevention - Disable one of the four necessary conditions.
- 3) Deadlock Avoidance
- 4) Deadlock Detection and Recovery - Most practical method.

8. DEADLOCK PREVENTION

Condition	Approach
Mutual Exclusion	Spool Everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically



- ⇒ ME and Hold and wait disabling are not possible practically.
- ⇒ Hold and wait disabling is not possible because a process cannot declare its need well advanced of the execution.
- ⇒ preemption causes inconsistency.
- ⇒ Disable circular wait is possible (Number all the Resources Initially and now a process is supposed to ask for resources in the increasing order) (P_1 asks R_2 then it should ask for resource $R_3, R_4, \dots, R_n (n \geq 2)$ but not R_1) (Can be implemented practically).

9. SAFE, UNSAFE, DEADLOCK AVOIDANCE AND BANKERS ALGORITHM

At any instant of time the snapshot of the system is

	Tape drives	plotters	scanners	CD Rom's
A	5	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources Assigned

	Tape drives	plotters	scanners	CD Rom's
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

$E = (6, 3, 4, 2)$

$A = (5, 3, 2, 2)$

$C = (1, 1, 1, 0)$

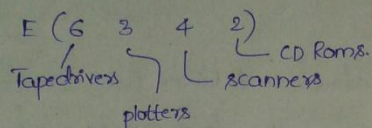
$D = (1, 1, 0, 1)$

$E = (0, 2, 0)$

is need remaining

E = It is a vector that represents the total no. of resources available in system

(48)



band (A) → How many resource instances are still available (E - p)

→ How many resources are assigned currently

→ safe state is a state in which you could satisfy the need in some order

→ when ever the Available satisfies the need of a process it will run till completion and release all of its "Resources assigned."

⇒ In the above example Available = (10 20) it satisfied the need of process 'D' so the process 'D' completes the execution and it releases all of its "resources. Assigned"

$$\begin{array}{r}
 A = \begin{pmatrix} 1 & 0 & 2 & 0 \end{pmatrix} \\
 D = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix} \\
 \hline
 \text{New Available} = \begin{pmatrix} 2 & 1 & 2 & 1 \end{pmatrix}
 \end{array}$$

'D' satisfied

⇒ with the New Available A can be satisfied

$$\begin{array}{r}
 \Rightarrow \text{New available} = \begin{pmatrix} 2 & 1 & 2 & 1 \end{pmatrix} \\
 \begin{pmatrix} 3 & 0 & 1 & 1 \end{pmatrix} \\
 \hline
 \begin{pmatrix} 5 & 1 & 3 & 2 \end{pmatrix}
 \end{array}$$

'A' satisfied

⇒ with New Available B can be satisfied

$$\begin{array}{r}
 A = \begin{pmatrix} 5 & 1 & 3 & 2 \end{pmatrix} \\
 B = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \\
 \hline
 \begin{pmatrix} 5 & 2 & 3 & 2 \end{pmatrix}
 \end{array}$$

'B' satisfied

⇒ with New Available A = 5 2 3 2

$$\begin{array}{r}
 \text{'C' can be satisfied} \\
 A = \begin{pmatrix} 5 & 2 & 3 & 2 \end{pmatrix} \\
 C = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \\
 \hline
 \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix}
 \end{array}$$

'C' satisfied

⇒ If you could satisfy the needs of all the process in some sequence then it is called safe state.

∴ DABCE = safe state

(48)

⇒ The above phase is called Deadlock Avoidance and the Algo is called Banker's Algorithm.

(49)

10. GATE 2007 QUESTION ON SAFE STATE

5) Three resource types X, Y, Z

There are 5 units of each resource type

consider the following table and say which process finishes last.

	X	Y	Z	X	Y	Z
P ₀	1	2	1	1	0	3
P ₁	2	0	1	0	1	2
P ₂	2	2	1	1	2	0

Alloc
Request

(A) P₀

(B) P₁

(C) P₂

(D) None of above since the system is in Dead lock.

$$\text{Total} = \begin{pmatrix} X & Y & Z \\ 5 & 5 & 5 \end{pmatrix}$$

$$\text{Allocated} = \begin{pmatrix} 5 & 4 & 3 \end{pmatrix}$$

$$\text{Available} = \begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$$

Now, the Available satisfies the need of P₁ ⇒ after execution P₁ releases the resources

$$\text{present available} = \begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$$

$$P_1 \text{ Resource} = \begin{pmatrix} 2 & 0 & 1 \end{pmatrix}$$

$$\text{Released} \quad \begin{pmatrix} 2 & 1 & 3 \end{pmatrix} = \text{New Available}$$

Now, $\begin{pmatrix} 2 & 1 & 3 \end{pmatrix}$ will satisfy the need of P₀ $\begin{pmatrix} 1 & 0 & 3 \end{pmatrix}$

$$\therefore \begin{pmatrix} 2 & 1 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 3 & 4 \end{pmatrix} = \text{New available}$$

Now, New available satisfies the need of P₂ ⇒ $\begin{pmatrix} 3 & 3 & 4 \end{pmatrix} \bullet \begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$

$$\therefore \text{New available} = \begin{pmatrix} 3 & 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 & 1 \end{pmatrix}$$

$$\text{Total} = \begin{pmatrix} 5 & 5 & 5 \end{pmatrix}$$

P₁ P₀ P₂ is the safe sequence

safe state

II. QUESTION ON SAFE STATE

	Allocated	Maximum	future need
P _A	1 0 2 1 1	1 1 2 1 3	0 1 0 0 2
P _B	2 0 1 1 0	2 2 2 1 0	0 2 1 0 0
P _C	1 1 0 1 1	2 1 3 1 1	1 0 3 0 0
P _D	1 1 1 1 0	1 1 2 2 0	0 0 1 1 0

Available = (0 0 x 1 1) (Q) what is the smallest value of 'x' for which this is a safe state?

Need = (max - Allocation) $\Rightarrow$ The future need matrix will be

$\Rightarrow$ using the available we should satisfy any one of the process future need

Now the available (0 0 x 1 1) satisfies need of P_D if x=1

$$= (0 0 1 1 1)$$

$$\therefore \text{Available} = (0 0 1 1 1)$$

$$P_D \text{ Resources} = (1 1 1 1 0)$$

$$(1 1 2 2 1) = \text{New Available}$$

Now, New Available (1 1 2 2 1) doesn't satisfy any of the process need.

If we observe we have put x=1 but if we put x=2 then the new available will be Avail (0 0 2 1 1) $\therefore$ Deadlock for x=1

new available will be Avail (0 0 2 1 1)

$$P_D (1 1 1 1 0)$$

$$\text{New Avail} = (1 1 3 2 1) \rightarrow \text{This solves/saves the need of}$$

all other processes $\therefore$ for x=2 safe state exists

$$\therefore \boxed{\text{Min value of } x=2} \therefore \text{No deadlock}$$

50

12. GATE 2014 QUESTION ON BANKERS ALGORITHM

51

	X	Y	Z	X	Y	Z	Future Need
P ₀	0	0	1	8	4	3	8 4 2
P ₁	3	2	0	6	2	0	3 0 0
P ₂	2	1	1	3	3	3	1 2 2

Allocation

Max

Need

this

Available (3, 2, 2)

Now given the system is in safe state then will the Req₁,

Req₁: P₀(0, 0, 2)

Req₂ be granted?

Req₂: P₁(2, 0, 0).

Req₁: P₀(0, 0, 2)

Available: (3, 2, 2)

need

Now, P₀ needs (0, 0, 2) Now assume it allocated it then, Allocation, Need matrices look as, when it allocate it the allocated Resources gets increased and Need of the process gets decreased

Allocation			Need		
X	Y	Z	X	Y	Z
P ₀	0	0	3	P ₀	8 4 0
P ₁	3	2	0	P ₁	3 0 0
P ₂	2	1	1	P ₂	1 2 2

Available will become (3, 2, 0).

↓
satisfy by P₁

⇒ 3, 2, 0 → New Available

3, 2, 0 → P₁ Release resources.

6, 4, 0 → New Available

need.
Deadlock
for x=1

Here the need of P₁ is satisfied, P₀, P₂ doesnot get satisfied so the Req₁ will not be granted.

Similarly perform with P₂ then you will get a safe sequence

sof

13. GATE 1996 ON BANKERS ALGORITHM

Allocation			Max Need			Future Need		
X	Y	Z	R ₀	R ₁	R ₂	R ₀	R ₁	R ₂
P ₀	1	0	2	4	1	2	3	1 0
P ₁	0	3	1	1	5	1	1	2 0
P ₂	1	0	2	1	2	3	0	2 1

Available (2, 2, 0)

Request P₀: (0, 1, 0)

which will the Req be granted

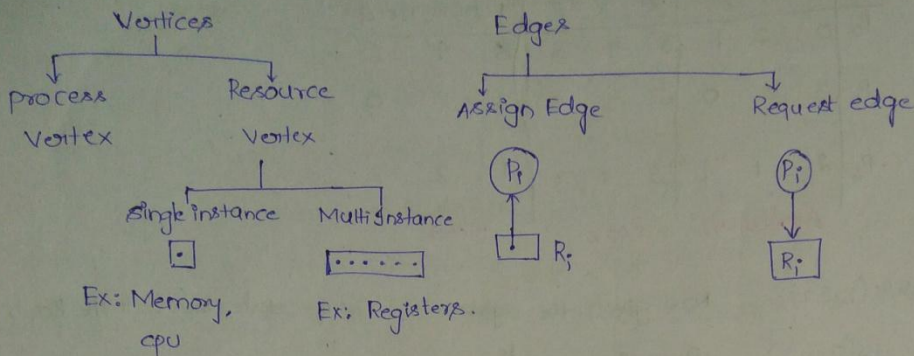
Now, assume it allocated the resource then

Allocated, Future Need, Available are going to change.

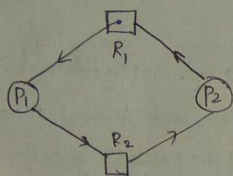
Allocated	Need (Future)
P ₀ : (1, 1, 2)	P ₀ : (3, 0, 0)

Available: (2, 1, 0) ⇒ cannot satisfy ⇒ Deadlock

14. RESOURCE ALLOCATION GRAPH

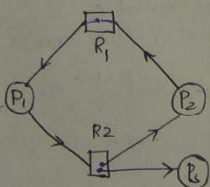
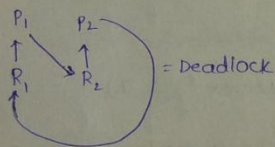


15. RESOURCE ALLOCATION GRAPH EXAMPLES



⇒ In case of single instance types if there is a cycle then there will be deadlock.

⇒ A presence of cycle in single instance Resource type is sufficient for Deadlock.



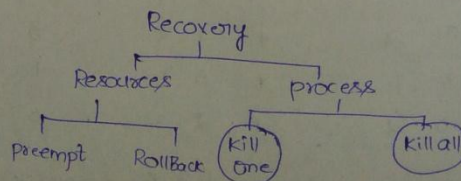
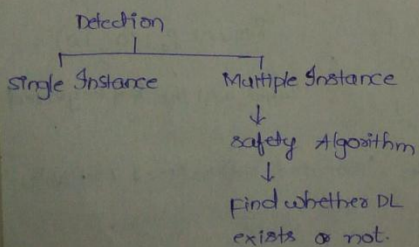
	Alloc		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
B	0	1	0	0

Available = (0 0)

No deadlock, safe sequence exists.

⇒ In multi instance of Graph the presence of cycle is not the sufficient condition but Necessary condition.

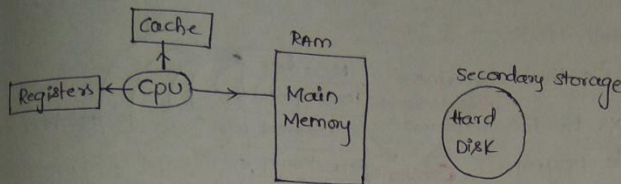
16. DEADLOCK DETECTION AND RECOVERY



4. MEMORY MANAGEMENT

NEED FOR MULTIPROGRAMMING AND MEMORY MANAGEMENT

→ CPU can directly access, Registers, Main memory, cache (if available)



Now assume the main memory is 4MB and I want to run process of size 4MB but the process spends fraction of 'p' time in doing I/O then the CPU utilisation = ?

CPU utilisation = The amount of time that you are using CPU running the program.

$$\text{CPU utilisation} = (1 - p) = 20\% \quad \left(\begin{array}{l} \text{MM} \\ 4\text{MB} \end{array} \quad \begin{array}{l} \text{Process size} \\ 4\text{MB} \end{array} \right) \quad (80\% \text{ time it spends in I/O assuming})$$

Now, MM = 16MB process size = 4MB. Each process spends fraction of 'p' time doing I/O what is the prob that all the process do I/O at same time?

$$\Rightarrow \begin{array}{l} \text{MM} = 16\text{MB} \\ \text{ps} = 4\text{MB} \end{array} \Rightarrow 4 \text{ processes} \quad \therefore \text{Now prob that all process do I/O at same time} = p \times p \times p \times p = p^4$$

$$\therefore \text{CPU utilisation} = 1 - p^4 \approx 60\% \quad (\text{assuming } 80\% \text{ of time spends in I/O})$$

$$\Rightarrow \text{MM} = 32\text{MB} \quad \text{ps} = 4\text{MB} \Rightarrow 8 \text{ processes}$$

$$\text{CPU utilisation} = 1 - p^8 \approx 83\%$$

$$\therefore \text{CPU utilisation} = 1 - p^n$$

$$= 1 - \left(\text{fraction of time spent in I/O} \right)^n$$

Degree of multiprogramming = The no of process that can be present in main memory at any given instant of time.

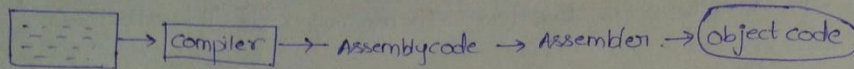
$$\therefore \text{CPU utilisation} = 1 - p^n \quad (n = \text{no. of process in MM}).$$

2. OBJECT CODE, RELOCATION AND LINKER

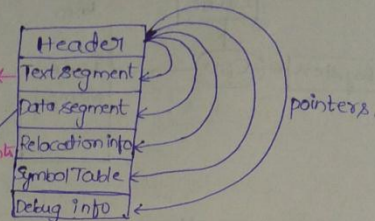
(54)

We write the program in the Text editor and the further process is

Text editor

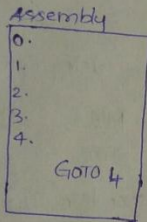


Now, the object code is a file that contains Instructions. The Header contains pointers to the various segments of the object code program.



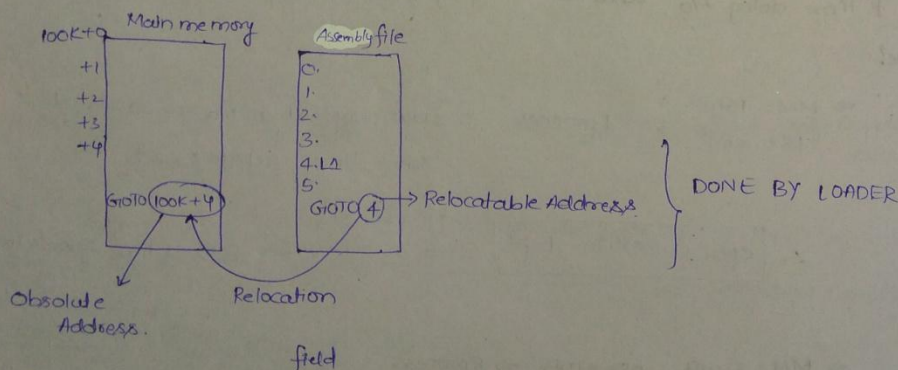
Now this object code file contains "Relocation Info". Now the compiler/assembler assume that the line no 0 starts with Address 0.

Now,



This works fine when the address starts from 0, but there is no guarantee that the file (object file) will be loaded from Main memory from address 0. (All process will not start from address 0).

Now, in the main memory say the object file loaded starting from one Address as 100K then,



Now, the "Relocation Info" in the object file contains what are the lines that are changed to (Relocatable Address -> absolute Address).

Symbol table contains every symbol that is present in the program.

main()	100	← Address
printf()	-	} unresolved Symbols (their address cannot be found)
scanf	-	

Libraries
C
D
E
!

This contains Relocatable Addresses because linker doesn't know where the program is loaded

phase (like contain the so we can => The main

3. LOADER

Object Linked

- 1) program load
- 2) Relocation
- 3) Symbol Res

Linker:

Symbols Resolution

Loader

Program loading

=> Loading is not

something and Main memory.

=> Loading time

process is

code

pointers

compiler/assembler

starts from
the file (object
code) from address
address 0).

from one Address

LOADER

lines that

not be found)

54

Libraries	
C	
D	
E	
!	

Object code: A	
Call: B, C, D	
call: printf	

Object code: B	
Calling: D, E	

Linker

Go through 2 pass

Object code

A
B
C
D
printf()

Stub/Gilucode

- 1st pass: Segments in the program
- 2nd pass: figure out the external References and resolve them
- Uses Segment table
- Symbol table

This contain
Relocatable
Addresses because
linker doesn't
know where the
program is loaded in MM.

some of the symbols are not resolved in this
phase (like printf()) if we resolve here then every process is going to
contain the copy of "printf" and the Memory will be unnecessarily wasted.
so we write (Stub or Gilucode) there.

⇒ The main purpose of linker is Relocation and symbol Resolution.

3. LOADER

Object linked (Relocatable)

- 1) program loading
- 2) Relocation
- 3) Symbol Resolution

Linker:

Symbols Resolution + Relocation

loader

Program loading + Relocation

⇒ Loading is nothing but Copying
something and loading it into
Main memory.

⇒ Loading time $\propto$ process size

The various times that are present while
the program and process creation are

1. compile time: what ever you do during
the compilation is called.
Symbolic names $\rightarrow$ Relocatable
Address
2. Link time: If linker knows the address
where it is loaded then linker converts
one form of relocatable code - Relocatable
code
3. load time: Relocatable Address $\rightarrow$ Absolute
Address
4. Runtime: Dynamic linking happens
 $\rightarrow$ Dynamic link libraries are linked
(Need support of operating system)

5. GATE 98 QUESTION ON LOADER

In a Resident OS computer, which of the following system software must reside in the main memory under all situations.

- a) Assembler (b) Linker (c) Loader (d) Compiler

Assembler can be loaded whenever we require

Linker " " " " " "

⇒ There is No program that can load the process except loader.

4. GATE 95 QUESTION ON LINKER

A linker is given object modules for a set of programs that were compiled separately. what information need not be included in an object module?

- a) object code (b) Relocation bits (c) Names and locations of all external symbols defined in the object module (d) Absolute Address of internal symbols.

The object module contains Header, Text Segment, Data Segment, Relocation info, symbol table, Debug Info.

- a) object code must be present definitely.
b) ~~Relocation~~ Relocation bits are needed to find the location.
c) stored in symbol Tables (Defines the names of External symbols and their address)
d) Absolute Address is not present in the object module.

6. 2001 QUESTION ON RELOCATION

The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned address

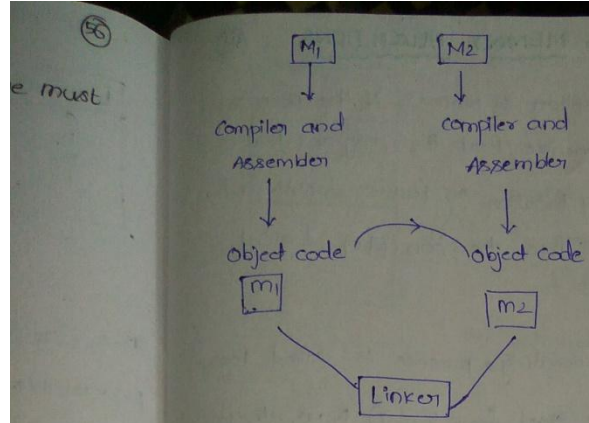
- a) Assembly (b) passing (c) Relocation (d) Symbol Resolution is called —

Basic definition of Relocation.

7. GATE 2004 QUESTION ON LINKER

Consider a program 'p' that consists two source modules m_1 and m_2 contained in two different files. If m_1 contains a reference to a function defined in m_2 , the reference will be resolved at

- a) Edit time (b) compile time (c) link time (d) load time



9. GATE 2003 QUESTION ON LINKING

Which of the following is NOT an advantage of using shared dynamically linked libraries as opposed to using statically linked libraries?

- a) smaller sizes of executable file
- b) Lesser overall page-fault rate in the system.
- c) faster program startup.
- d) Existing programs need not be re-linked to take advantage of newer versions of libraries.

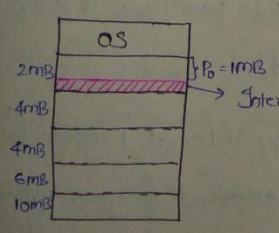
a) → Yes, when we have dynamic sharing we are not going to have object code of the library function in our executable code but we just pass the reference to where the code is present.

b) → FALSE = Because the library that you are referring might not be present and must be loaded in case of dynamic linking.

c, d) → TRUE

10. FIXED PARTITIONING

Contiguous Memory Allocation



⇒ Memory is divided into fixed size partition

⇒ $P_0 = 1\text{MB}$ ⇒ smallest partition that I could assign = 2MB

⇒ Internal fragmentation (cannot be assigned to any of the process)

⇒ If $P_2 = 11\text{MB}$ (No partition could be assigned)

= External fragmentation

⇒ Degree of Multiprogramming is fixed/limited

11. RELOCATION AND PROTECTION IN CONTIGUOUS MEMORY ALLOCATIONS

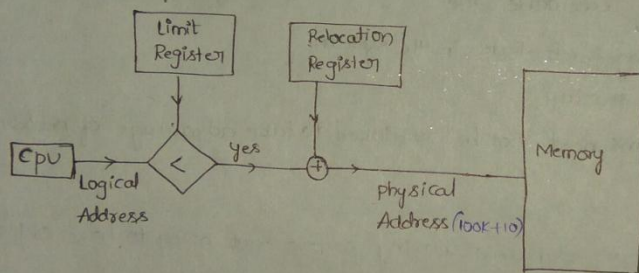
(58)

⇒ Loader works efficiently if there is no preemption of process in the memory because if there is preemption there is no guarantee that the program that is preempted will not load from same memory location. So loader doesn't work. So operating system used Run time binding. Thus the concept of logical and physical Address come into picture.

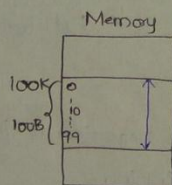
⇒ The Addresses generated by the cpu while generating a process is called Logical Addresses. (Relocatable Address (assume address start from Zero)). Now, the logical Address should be converted to physical address.

⇒ Involving loader to convert Relocatable Address to absolute Address is not a good idea (preemption is possible).

Relocatable Address = Logical Address
- Generated by cpu



⇒ A process cannot span over two partitions.



Relocation Register = 100k

Limit Register = 100.

⇒ The 10th Byte in the process is equal to 100k+10 in the Memory

12. DYNAMIC PARTITIONING

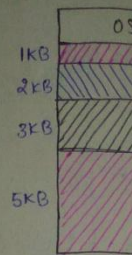
⇒ Dynamic partitioning / variable partitioning.

⇒ The user space / Memory is not divided into fixed size partitions it is left as it is. (The partition will be equal to process Need)

⇒ If i go for Dynamic partitioning there won't be any internal fragmentation.

⇒ If there is internal fragmentation then there is external fragmentation.

⇒ If there is no internal fragmentation then there may or may not be external fragmentation.



⇒ The adva

a) It is

b) size

⇒ The Alloca

13. BITMAP

⇒ The data st

⇒ The data stru

Bitmap:

⇒ The memory is

⇒ for every A

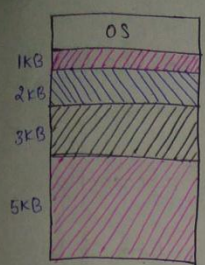
the allocation occupied.

58

memory that is not work. logical

logical the logical

process is not logical Address generated by cpu



⇒ Suppose that all the memory fragments are filled completely ⇒ There is no internal fragmentation, Now if the cells 1KB and 3KB got released but we cannot allocate the space to program of 4KB because the memory that is free is not contiguous locations. ∴ External Fragmentation is present.

⇒ Compaction/Defragmentation ⇒ costly (Bring all empty spaces together)

⇒ The advantages of Dynamic partitions is

- a) It is flexible (Degree of Multiprogramming is not fixed)
- b) Size of the process is not limited by the size of partition.

⇒ The Allocation and deallocation of memory is complex

3. BITMAP FOR DYNAMIC PROGRAMMING

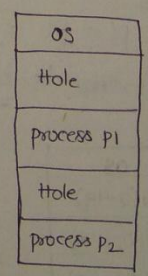
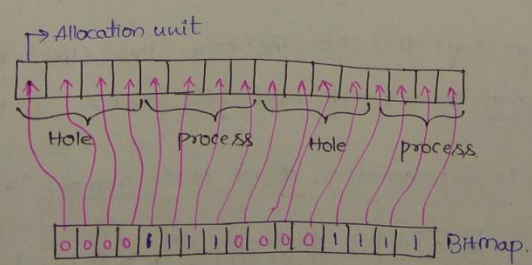
⇒ The Data structures that are used to keep track of free and occupied spaces

⇒ The Data structure is Bitmap and Linked Lists.

Bitmap:

⇒ The memory is divided into small parts called "Allocation units".

⇒ For every Allocation unit the single bit 0 or 1 is assigned (0 represents the allocation unit is open/free and 1 represents the allocation Bit Unit is occupied).



it is left

fragmentation.

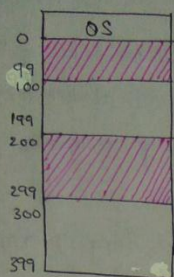
fragmentation.

external fragmentation

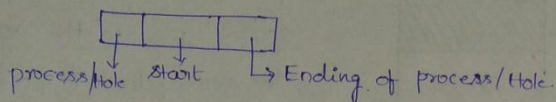
⇒ Now suppose the size of allocation unit is 4 Bytes = 32 Bits. Now 1 bit in the Bitmap is used to represent whether these 4B (Allocation unit) is occupied or not. so the Bitmap is going to take $\frac{1}{(32+1)} = \frac{1}{33}$ rd part of memory is taken by Bitmap.

⇒ Bitmap is not widely used / Not used nowadays.

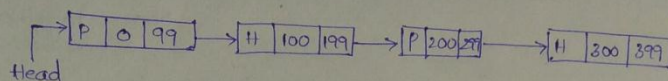
14. LINKED LIST FOR DYNAMIC PARTITIONING



⇒ The structure of the linked list will be like this



⇒ For the above diagram the structure of Linked list will be



⇒ Searching will not take lot of time

⇒ Main the nodes in the increasing order of starting Address

⇒ Mainting as double linked list has advantage, Now whenever you free a process then check if the previous node is hole and then merge them and check the node after hole then you should also merge it. precisely after freeing the process if you get run of holes then merge them into single node

15. FIRST FIT, NEXT FIT, BEST FIT, WORST FIT

The various algorithms that are used on the linked list are

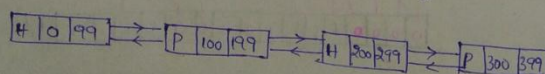
- 1) First fit
- 2) Next fit
- 3) Best fit
- 4) worst fit

First Fit

⇒ The first fit algorithm says that scan the entire linked list and find the hole that is big enough to hold the need of the process.

OS
H(0-99)
P(100-199)
H(200-299)
P(300-399)

The LL Representation of the above memory is.



⇒ Now say the process 'P' requires 50 bytes, Now the first fit algo says that start scanning the LL and while scanning if you find a hole that is large enough to hold the process then allocate memory to the required process.

(60)

Now, after a

Next Fit

The Next fit will start a

⇒ Next fit is

Best Fit

⇒ Best fit Algo which can a

⇒ The disadvan it will crea

Worst fit

⇒ This says assign the

Quick fit

⇒ This says th and maintain

⇒ We might hav

⇒ The malloc()

16. GATE 2004 Qu

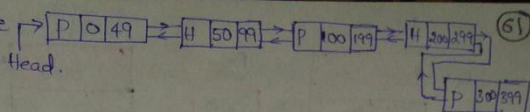
Consider the heap regions are in

The sequence of if we use

- a) either first fit/
- b) first fit but no

60

Now, after allocation the LL will be



Next Fit

The Next fit is same exactly as first fit but the difference is we will start scanning the list exactly from the point where we left earlier.

⇒ Next fit is not better than first fit

Best Fit

⇒ Best fit Algo says that among all the holes find out the smallest hole which can accommodate the process.

⇒ The disadvantage of Bestfit is, it is slower compared to first fit. and it will create small small holes.

Worst fit

⇒ This says scan the entire list and assign the biggest hole possible and assign the process.

↳ Not biggest enough hole

Quick fit

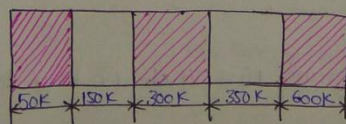
⇒ This says that there are fixed sizes for the most frequently used process and maintain a linked list for them.

⇒ We might have to maintain separate list.

⇒ The malloc() function is going to use First fit Algorithm.

16. GATE 2004 QUESTION ON FIRST FIT AND BEST FIT

Consider the Heap in which blank regions are not in use and hatched regions are in use (occupied)



The sequence of Requests for blocks of size 300, 25, 125, 50 can be satisfied if we use

- either first fit / Best fit policy (anyone)
- first fit but not best fit policy
- Best fit but not First fit policy
- (d) None



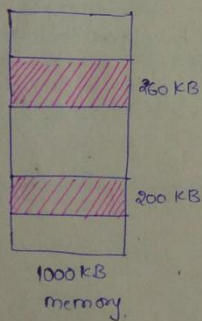
50 cannot be assigned.

∴ BF fail.

17. VARIABLE PARTITIONING

A 1000KB memory is managed using variable partitions but no compaction. It currently has 2 partitions of sizes 200KB and 260KB respectively. The smallest allocation request in KB that could be denied is for

- a) 151 b) 181 (c) 231 (d) 541



Now, Total = 1000KB

Already occupied = 200 + 260 = 460KB.

Now, Remaining space = 540KB. ∴ The max possible ans is 541 but it is not minimum.

⇒ The max no. of partitions present in the partition is 3.

⇒ Now we need to take care that the three partitions should be minimum (this occurs when the three partitions are of equal size)

So the remaining space = 540KB should be divided into 3 partitions of equal size = $\frac{540}{3} = 180$ KB.

∴ The min size of the program that it should have to get rejected is

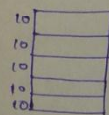
181

18. GATE 98

3rd Comput jobs to me

19. SUMMARY

Equal



If ↑

Ef ↑

GATE 98 QUESTION ON FIXED PARTITIONING

(63)

In a computer system where the best fit algo is used for allocating jobs to memory partitions the following situation was encountered

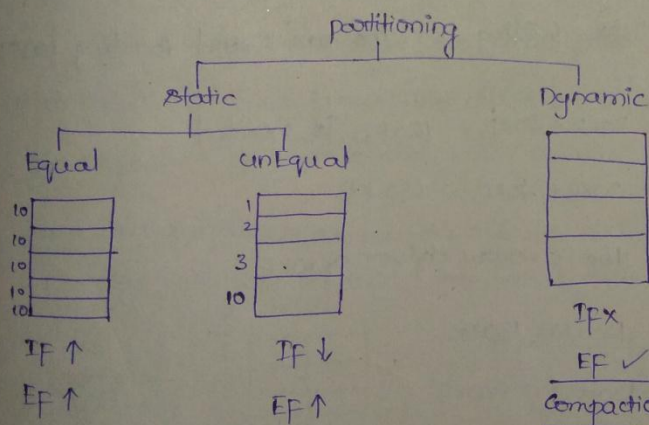
partition sizes in KB	4K		8K		20K		2K
Jobs	2K	14K	3K	6K	10K	20K	2K
Time for execution	4	10	2	1	1	8	6

when will 20K Job complete?

4K	3K (0-2)		
8K	6K (0-1)		
20K	14K (0-10)	10K (10-11)	At t=11, 20K (11+8=19)
2K	2K (0-4)		
partitions	Jobs	At time t=10, both 8K and 20K partition gets freed.	

Ans = 19

19. SUMMARY ON PARTITIONING



20. OVERLAYS

⇒ Sometimes the size of the biggest partition will be less than the size of the process, In that case we should go with "OVERLAYS"

⇒ Only a part of the program is loaded in the memory at any given instant of time.

⇒ The example of overlay is "Assembler"

Consider a 2 pass assembler

pass1: 70KB	pass2: 80KB	Symbol table: 30KB
Common Routine: 20KB	Total memory: 200KB	

2 pass means at any time it will be doing only one thing either 1st pass/2nd pass

In the first pass it needs pass1 needs Symbol Table: 30KB, Common Routine: 20KB
 Pass1 need: $70 + 30 + 20 = 120\text{KB}$.

⇒ But at any time only one pass will be in use and both the passes always need Symbol Table and common Routine. If overlay driver is 10KB, then what is the min portion size required?

⇒ overlay driver is responsible for pulling out the pass1 and loading pass2

Now, In pass1: $70\text{KB} + 30\text{KB} + 20\text{KB} = 120\text{KB}$ is needed

In pass2: $80\text{KB} + 30\text{KB} + 20\text{KB} = 130\text{KB}$

Now, pass1/pass2 requires the overlay driver of 10KB

∴ pass1 = $120 + 10 = 130\text{KB}$

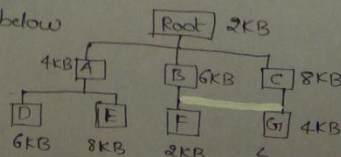
pass2: $130 + 10 = 140\text{KB}$

Now, we need in total a partition of size ↓

big enough that can hold the partition of size 140.

21. GATE 98 QUESTION ON OVERLAYS

The overlay tree for a program is shown below



64

of the

start of

30KB

2nd pass

routine: 20KB

at 20

120KB.

30KB

is

pass 2

partition

3

what will be the size of partition (in physical memory) required to load (and run) this program?

65

- a) 12KB
- b) 14KB
- c) 10KB
- d) 8KB.

We might need $(R+A+D)$ in the memory = $2+4+6 = 12$ KB

Similarly, $(R+A+E) = 14$ KB

$(R+B+F) = 10$ KB

$(R+C+G) = 14$ KB

$\therefore$ The size of partition = $\max \{12, 14, 10, 14\}$

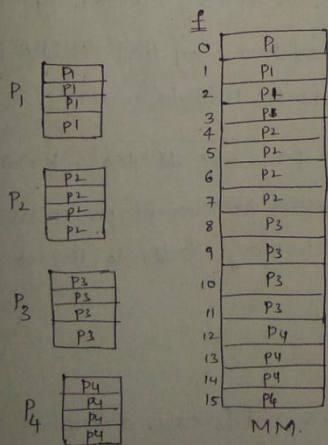
Min size of partition = 14

22. NEED FOR PAGING

$\Rightarrow$ The main problem that even the dynamic partitioning could not eliminate is External Fragmentation.

$\Rightarrow$ so what they have proposed is divide the process into small parts called pages and also divide the memory into small parts called frames and load each page in a single frame. (The division should be in such a way that the page size and frame size are equal).

$\Rightarrow$ Now, let us say the Main memory size is 16KB and I have 4 process each of size 4 KB and the MM is divided into frames each of size 1KB.



$\Rightarrow$ Now say P_2 and P_4 want for I/O (they are removed from MM and stored on hard disk)

$\Rightarrow$ Now a New process P_5 of size 8KB has come it is divided into 8 pages and these are stored at 8 vacant places created by P_2 and P_4 and thus External Fragmentation can be eliminated.

$\Rightarrow$ Cpu generates the logical Address and it is divided into 2 parts <page num, page offset>

$\Rightarrow$ The above logical address should be converted into physical address by Memory Manager instruction.

23. PAGING EXPLAINED WITH EXAMPLE

Now Let us say the $\Rightarrow$ Main Memory size = 64B then $2^6 = 64 \Rightarrow 6$ bits are needed in the physical address

$$\Rightarrow \text{frame size} = 4B$$

$$\Rightarrow \text{No. of frames} = \frac{64B}{4B} = 16 \text{ frames (4 bits are enough to uniquely identify a frame in MM)}$$

$$\Rightarrow \text{process size} = 16B, \text{ page size} = 4B$$

$$\Rightarrow \text{No. of pages} = \frac{16B}{4B} = 4 \text{ pages (2 bits are enough to identify a page)}$$

$\Rightarrow$ Now, the structure of the process will be

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

No. of frames = 16

process size = 0-15B
= 16B

$\Rightarrow$ Now, the structure of Main memory will be

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

and say the process is loaded into MM

starting from frame 2 (say frame 0, frame 1) are occupied.

loaded in Main Memory

MM = 64B $\Rightarrow$ divided in frames of size 4B.

$\Rightarrow$ Now cpu generates the logical address say 6 which means the cpu needs 6th Byte in the program but it is loaded in the memory, The 6th byte in the process should be converted to 14 in such a way that it identifies the 14th byte in the MM (6th Byte corresponds to 14th Byte in MM).

$$\therefore 6 = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline \end{array}$$

page no | page offset

$\Downarrow$

Now this Address should be converted in such a way that it identifies 14th Byte in MM.

with the help of above info <page no, page offset> we go to page table.

$\Rightarrow$ Every

that

$\Rightarrow$ The P

$\Rightarrow$ The

CP

pag
= 1

66

re needed
physical address

naught to
identify a
mm)

ugh to
tify a page)

frames = 4
size = 0-15B
= 16B

Memory

s of size 4B.

pu needs
byte to
identifies

it, $\Rightarrow$ 2 bits
icular byte
into 2

that

$\Rightarrow$ Every process has its own page table, the pagetable contains the frame no's that corresponds to diff page no's.

67

0	f2
1	f3
2	f4
3	f5

<page no> <frame number>

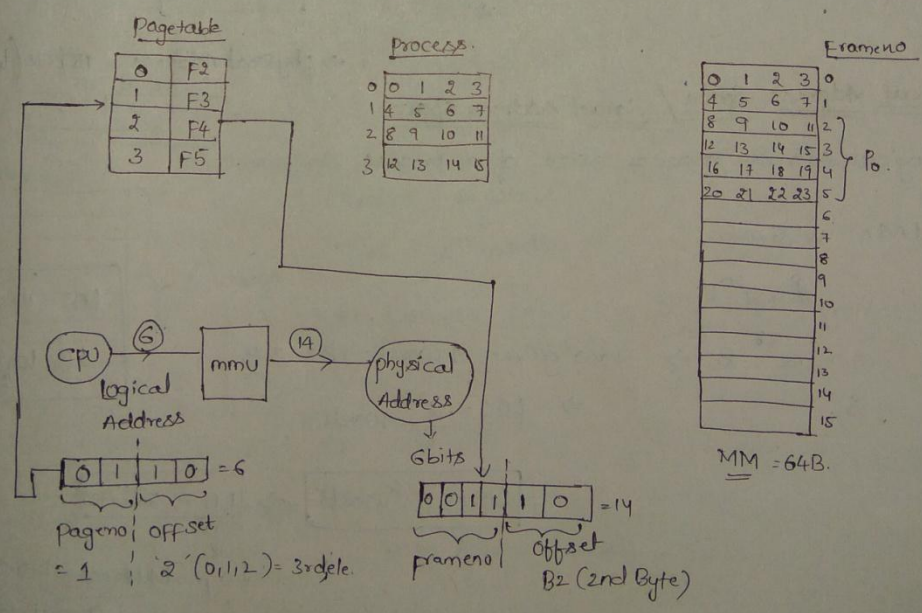
page no = $(01)_2 = (1) \Rightarrow$ page 1
corresponds to frame 3.
page offset = frame offset = $(10)_2 = 2$
 $\therefore$ Go to frame 3 and fetch the
Byte at 2nd position (0th, 1st, 2nd)
 $= \langle f_3, 10 \rangle$

$\Rightarrow$ The physical address is limited by size of MM (Gbits in this case) = 14 Bytes.
 $\Rightarrow$ The logical address size is limited by size of process (4 bits in this case)

physical address:

0	0	1	1	1	0
---	---	---	---	---	---

 = 14
frame number offset (same as offset in logical address)
 $\downarrow$ $\downarrow$
0011 2
= 3. (2)



24. BASICS OF BINARY ADDRESS

$2^1 = 2$	$2^5 = 32$	$2^9 = 512$	$2^{10} = K$	$\Rightarrow 128 GB = 2^7 \times 2^{30} B$ $= 2^{37} B$ $\Rightarrow 2^{28} B = 2^8 \times 2^{20} B$ $= 256 MB$
$2^2 = 4$	$2^6 = 64$	$2^{10} = 1024$	$2^{20} = M$	
$2^3 = 8$	$2^7 = 128$		$2^{30} = G$	
$2^4 = 16$	$2^8 = 256$		$2^{40} = T$	

25. PHYSICAL ADDRESS SPACE AND LOGICAL ADDRESS SPACE

$\Rightarrow$ The smallest addressable unit in the computer is word

Physical Address Space

$\Rightarrow$ physical address space is nothing but size of main memory

$\Rightarrow$ Now, $PAS = 128 KB$

$$PAS = 2^7 \times 2^{10} B$$

$$PAS = 2^{17} \text{ Bytes} \Rightarrow \text{Now given 1 word} = 4B = 2^2 B$$

$$\Rightarrow PAS = 2^{17} \text{ Bytes}$$

$$= \frac{2^{17}}{2^2} \text{ words} \Rightarrow \boxed{PAS = 2^{15} w}$$

$$\begin{aligned} PAS &= 'm' \text{ words} \\ PA &= \log_2(m) \text{ bits} \end{aligned}$$

$$\Rightarrow \text{physical address} = 15 \text{ bits } (\log_2(2^{15}))$$

Logical Address Space / Virtual Address Space

$\Rightarrow$ logical address space = size of a process

$$LAS = 256 MB$$

$$= 2^8 \times 2^{20} B$$

$$= 2^{28} B \Rightarrow \text{Now given 1 word} = 4B = 2^2 B$$

$$\Rightarrow LAS = \frac{2^{28}}{2^2} \text{ words}$$

$$\boxed{LAS = 2^{26} \text{ words}} \Rightarrow \boxed{LA = 26 \text{ bits}}$$

$$\begin{aligned} LAS &= 'm' \text{ words} \\ LA &= \log_2(m) \text{ bits} \end{aligned}$$

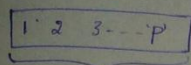
Logical Address = No. of bits required to address

$\Rightarrow$ Whenever the size of the process is larger than that of MM then we are putting a part of main memory and we will be able to run it.
process in

$\Rightarrow$ virtual memory was introduced to increase the degree of multiprogramming.

68

page size = frame size = 'p' words $\Rightarrow (\log_2 p)$ bits are needed



$\Rightarrow$ 'p' words are present in the page

page size

$\Rightarrow$ page size = 4KB.

word size = 4B.

Page size = $2^{10} \times 4B$.

page size = 2^{10} words $\Rightarrow$ 10 bits = page offset.

69

PAGE TABLE

PAS = Main memory = M words

LAS = process size = 'L' words.

page size = 'p' words.

PA = $\lceil \log_2 M \rceil$ bits. = m bits (say)

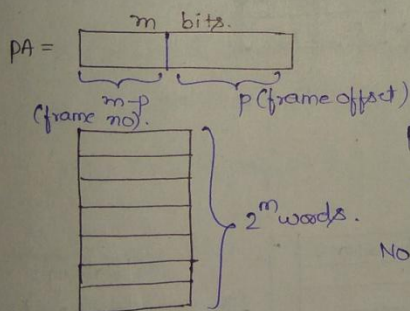
LA = $\lceil \log_2 L \rceil$ bits = 'l' bits (say)

page offset = $\lceil \log_2 p \rceil = p$ bits (say).

LAS = 128MB $\Rightarrow 2^{27} B = 27$ bits

PAS = 1MB = $2^{20} B = 20$ bits

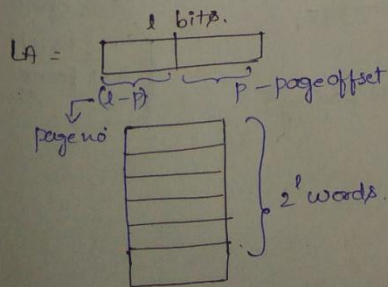
ps = 4KB = $2^{12} B = 12$ bits.



Frame = 2^p words

PAS = 2^m words

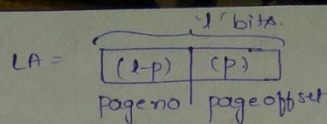
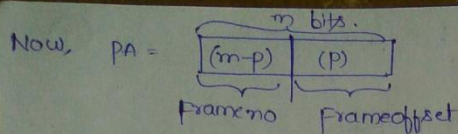
No. of frames = $\frac{PAS}{\text{frame size}} = \frac{2^m}{2^p} = 2^{m-p}$ frames



page size = 2^p words.

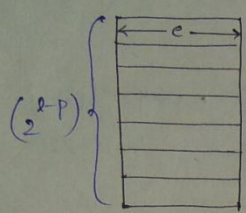
LAS = 2^l words

No. of pages = $\frac{2^l}{2^p} = 2^{l-p}$ pages



(70)

page table



$\Rightarrow$ No. of entries in the page table = No. of pages in the process = 2^{l-p}

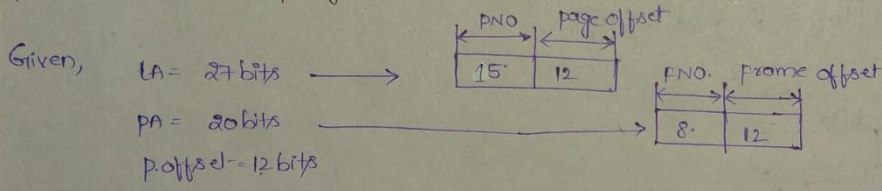
$\Rightarrow$ Let 'e' be the size of each entry

$\Rightarrow$ page table size = $2^{l-p} * (e)$ bytes.

$\Rightarrow$ If the value of 'e' is not given then, page table size = $2^{l-p} * (m-p)$

Example

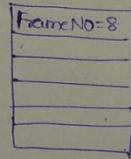
$LAS = 128MB = 2^{27} B$	$LA = 27 \text{ bits}$	}	page table size = ?
$PAS = 1MB = 2^{20} B$	$PA = 20 \text{ bits}$		
$PS = 4KB = 2^{12} B$	$\text{page offset} = 12 \text{ bits}$		



$\therefore$ No. of pages = 2^{15}

$\therefore$ page table size = $2^{15} \times 8 \text{ bits} = 2^{15} \times 1 \text{ Byte}$

PTS $\Rightarrow 2^{15} \text{ Bytes} = 32 \text{ KB}$



$\left. \vphantom{\begin{matrix} \text{Frame NO} \end{matrix}} \right\} 2^{15} \text{ pages}$

Nothing is mentioned about the size of each entry so take it as frame size.

$PTS = 32 \text{ KB.}$
$PS = 4 \text{ KB.}$

$PTS \gg PS$ (Multilevel paging).

LAS
128 KB
256 KB
1 MB
$2^{20} B$
$2^{22} B$
$2^{22} B$

1) LAS
PAS
pag
NO

NO. pr

PTE

PTS

2) LAS:
page

(70)

27. NUMERICAL ON PAGING

(71)

pages
 $cess = 2^{1-p}$

LAS	PAS	LA	PA	Page size	Page offset	pages	frames	PTE	PTS.
128 KB	128 KB	17 bits	17 bits	4 KB	12 bits	32	32	5 bits	$(\frac{2^5 \times 5}{8})^8$
256 KB	4 MB	18 bits	20 bits	4 KB	12 bits	64	2^8	2 B	$(2^6 \times 2) \text{ Bytes}$
1 MB	2^{18} B	20 bits	18 bits	2^{10} B	10 bits	2^{10}	256	8 bits	2^{13} (bits)
2^{20} B	512 MB	20 bits	29 bits	2^{12} B	12 bits	256	2^{17}	17 bits	$(256 \times 17) \text{ bits}$
2^{22} B	2^{21} B	22 bits	21 bits	2^{12} B	12 bits	2^{10}	2^9	9 bits	
2^{22} B	2^{22} B	22 bits	22 bits	2^{14} B	14 bits	2^8	2^8	8 bits	256 B

$$1) \text{ LAS} = 128 \text{ KB} \Rightarrow \text{LA} = \lceil \log_2 128 \text{ KB} \rceil = 17 \text{ bits} = 128 \text{ KB} = 2^7 \times 2^{10} \text{ B} = 2^{17} \text{ Bytes} = 17 \text{ bits}$$

$$\text{PAS} = 128 \text{ KB} \Rightarrow \text{PA} = \lceil \log_2 128 \text{ KB} \rceil = 17 \text{ bits}$$

$$\text{page size} = 4 \text{ KB} \Rightarrow \text{page offset} = 4 \times 2^{10} \text{ B} = 2^{12} \text{ B} \Rightarrow \text{page offset} = 12 \text{ bits}$$

$$\text{No. of pages} = \frac{\text{LAS}}{\text{PS}} = \frac{128 \text{ KB}}{4 \text{ KB}} = 32 \text{ pages}$$

$$\text{No. frames} = \frac{\text{PAS}}{\text{FS}} = \frac{\text{PAS}}{\text{PS}} = \frac{128 \text{ KB}}{4 \text{ KB}} = 32 \text{ frames}$$

PTE = PTE contains frame nos, here frames = 32 $\Rightarrow$ 5 bits will be needed to represent each frame uniquely.

$$\text{PTS} = (\text{No. of pages}) \times (\text{PTE size}) \text{ bits} = 2^5 \times 5 \text{ bits} = \frac{2^5 \times 5}{8} \text{ Bytes} = 20 \text{ B}$$

thing is
 mentioned
 out the
 size of each
 page so take
 frame
 size

$$2) \text{ LAS} = 1 \text{ MB} = 1 \times 2^{20} \text{ B} = 2^{20} \text{ B} \Rightarrow \text{LA} = \lceil \log_2 2^{20} \rceil = 20 \text{ bits}$$

$$\text{page offset} = 10 \text{ bits} \Rightarrow \text{Now, the page size} = 2^{10}$$

$$\Rightarrow \text{LAS} = 1 \text{ MB}, \text{ page size} = 2^{10} \Rightarrow \text{No. of pages} = \frac{\text{LAS}}{2^{10}} = \frac{2^{20}}{2^{10}} = 2^{10}$$

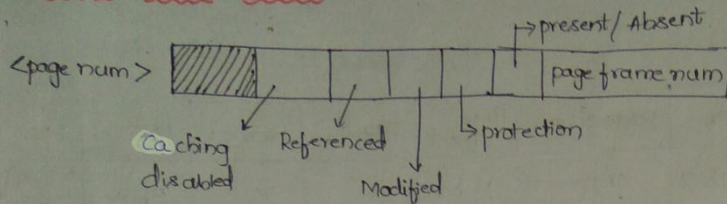
$$\Rightarrow \text{Now, No. of frames} = \frac{\text{PAS}}{\text{Frame size}} = \frac{\text{PAS}}{\text{page size}} \Rightarrow \text{PAS} = \text{page size} \times \text{No. of frames}$$

$$\Rightarrow \text{PAS} = 2^{10} \times 256 = 2^{10} \times 2^8 = 2^{18}$$

$$\Rightarrow \text{PTE} = 8 \text{ bits} \Rightarrow \text{PTS} = \text{No. of pages} \times \text{PTE} = 2^{13} \text{ bits}$$

$$\Rightarrow \text{PA} = 18 \text{ bits}$$

28. PAGE TABLE ENTRY



- ⇒ This bit/field Represent whether a particular page should be present in Cache or not
- ⇒ Referenced: This bit say whether this page has been referenced or not.
- ⇒ Modified: whether the page has been modified or not. (Also called Dirty bit)
- ⇒ protection: what kind of protection you want to assign to a particular page (Read, write, Read/write, ...)
- ⇒ present / Absent: Represent whether the page is present / Absent in memory.
 - ⇒ Demand paging is loading the pages into MM only when they are needed.
 - ⇒ Useful in Virtual memory concept.
- ⇒ page frame num ⇒ Contains the frame no. corresponding to particular page number.

29. GATE 2004 QUESTION ON PAGETABLE ENTRY

In virtual memory system, size of virtual address is 32 bit, size of physical Address is 30 bits, page size is 4KB and size of each page table entry = 32 bit, The MM is Byte addressable. which of the following is max no. of bits that can be used for storing protection and other information in each page table entry.

- A) 2 B) 10 C) 12 D) 14

- ⇒ Logical Address = LA = 32 bits
 PA = 30 bits
 PS = 4 KB.
 PTE = 32 bits.

$$PA = 30 \text{ bits}$$

$$LA = 32 \text{ bits}$$

Now, page

30. NEED

$$LA = 22 \text{ bits}$$

$$LAS = 2^{22} \text{ Bytes}$$

$$PS = 4 \text{ KB} = 2^{12} \text{ B}$$

$$PTE = 4 \text{ B}$$

$$PTS = (\text{No. of})$$

⇒ Now, To

should

big error

should

(CPU)

[B]

Th

of

(72)

$$PA = 30 \text{ bits} \Rightarrow PAS = 2^{30} B.$$

$$LA = 32 \text{ bits} \Rightarrow LAS = 2^{32} B.$$

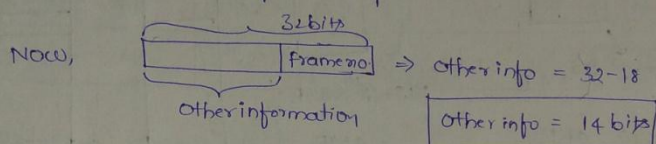
{ Size of process > Size of Main Memory }

(73)

$$\text{Now, page size} = \text{Frame size} = 4 \text{ KB} = 2^{12} B.$$

$$\text{No. of frames} = \frac{PAS}{FS} = \frac{2^{30} B}{2^{12} B} = 2^{18} B.$$

$$\Rightarrow \text{No. of bits required for frame no} = \lceil \log 2^{18} \rceil = 18 \text{ bits}.$$



30. NEED FOR MULTILEVEL PAGING

$$LA = 22 \text{ bits}.$$

$$LAS = 2^{22} \text{ Bytes}.$$

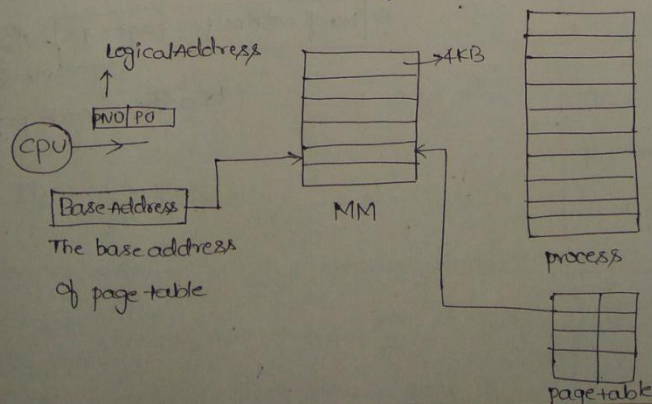
$$PS = 4 \text{ KB} \Rightarrow \text{page offset} = 12 \text{ bits} \Rightarrow \text{No. of pages} = \frac{LAS}{PS} = \frac{2^{22} B}{2^{12} B} = 2^{10}$$

$$\text{No. of pages} = 1 \text{ K}$$

$$PTE = 4 B$$

$$PIS = (\text{No. of pages}) \times PTE = 1 \text{ K} \times 4 B = 4 \text{ KB}.$$

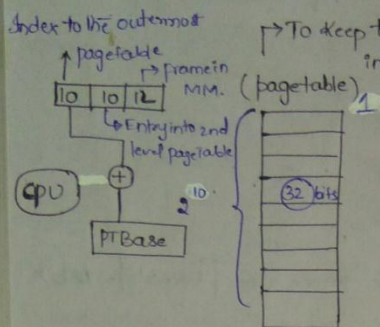
$\Rightarrow$ Now, To run a process/program, the process pages and the page table should be loaded in the Main memory. Now if the page table is very much big enough to fit in one frame of the main memory then the page table should also be divided into pages.



If page table size > the page size then we go for multilevel paging.

31. TWO-LEVEL PAGING EXAMPLE

(14)



$$\text{Here } \text{PTS} = 2^{10} \times 4 \text{ B}$$

$$= 4 \text{ KB}$$

$$\text{PS} = 4 \text{ KB}$$

$$\text{PTS} = \text{PS}$$

$$\Rightarrow \text{No. of entries} = 2^{10}$$

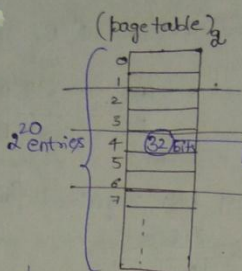
$\Rightarrow$ To identify a particular entry we need 10 bits.

$\Rightarrow$ This pagetable should always be in main memory.

$\Rightarrow$ The PT Base Register stores the Address of the frame that contains the (pagetable).

$\rightarrow$ To keep track of pages of PT_1 are loaded in which frames of MM.

$$\text{PS} = 4 \text{ KB} = \text{FS}$$



$$\text{PTS} = (2^{10} \times 4) \text{ B}$$

$$= 4 \text{ MB}$$

$$\text{PS} = 4 \text{ KB}$$

$$\text{No. of frames} = \frac{2^{14}}{2^{12}} = 2^2 \text{ frames}$$

$$\text{No. of pages} = \frac{2^{32}}{2^{12}} = 2^{20} \text{ pages}$$

$$\text{Now } \text{PTS} \gg \text{PS} = \text{FS} \Rightarrow \text{Hence the page table cannot}$$

fit in one frame of the MM

so divide the page table into no. of pages

$$\text{Now we get no. of pages} = \frac{4 \text{ MB}}{4 \text{ KB}} = 1 \text{ K pages}$$

$\Rightarrow$ Now the page table is divided into pages and loaded in main memory. Now we need to keep track of pages of the pagetable are loaded

which in which frame of MM, so we need another page table

$\Rightarrow$ Now here assume each entry size = 4B.

$$\text{Each page size} = 4 \text{ KB}$$

$$\Rightarrow \text{No. of entries per page} = \frac{4 \text{ K}}{4} = 2^{10} = 10 \text{ bits are needed to address each page of pagetable}$$

32. EXA

VA=LA	PO
48b	
64b	
72b	
72b	2
72b	16

$$1) \text{ VA} = 4$$

$$\text{PS} = 16$$

$$\therefore \text{PT}_1 =$$

$$\Rightarrow \text{VA} = 6$$

$$\text{PS} = 16$$

$$\text{PTE} =$$

$$\text{No. of P}$$

$$\text{PT}_2 =$$

$$\text{PT}_3 =$$

1) Index the

$$\Rightarrow \text{PTE} = 4 \text{ B}$$

$$\text{PS} = 16 \text{ B}$$

$$\Rightarrow \text{PTE} = 4 \text{ B}$$

14

32. EXAMPLES ON MULTI LEVEL PAGING

45

VA=LA	page size	PTE	PT1	PT2	PT3	Address splitting
48b	16KB	4B	$(2^{34} \times 4)B$	$2^{22} \times 2^{24}$	$2^{12}B$	10 12 12 14 PT3 PT2 PT1 → page offset
64b	1MB	4B	2^{46}	$2^{28}B$	$2^{10}B$	8 18 18 20 PT3 PT2 PT1 → page offset
72b	1GB	4B	$(2^{42} \times 2^2)B$	$(2^{14} \times 2^2)$	—	14 28 30 PT2 PT1 → page offset
72b	256MB	4B	$(2^{44} \times 2^2)B$	$(2^{18} \times 2^2)$	—	18 26 28 PT2 PT1 → page offset
72b	16MB	4B	$(2^{48} \times 2^2)B$	$(2^{26} \times 2^2)$	$(2^4 \times 2^2)$	4 22 22 24 PT3 PT2 PT1 → page offset

$VA = 48 \text{ bits} \Rightarrow VAS = 2^{48}B$
 $PS = 16KB = 2^{14}B \Rightarrow \text{No. of pages} = \frac{VAS}{PS} = \frac{2^{48}}{2^{14}} = 2^{34} \text{ pages}$
 $\therefore PT_1 = 2^{36}B$

$\Rightarrow PT_2 = \frac{PT_1}{PS} = \frac{2^{36}}{2^{14} \times 2^{10}} = 2^{12}B \times PTE = 2^{22} \times 2^{24} = 2^{46}$
 $\Rightarrow PT_3 = \left(\frac{PT_2}{PS} \times PTE \right) = \frac{2^{46}}{16 \times 2^{10}} \times 4B = 2^{26}B$

$\Rightarrow PT_2 = \frac{PT_1}{PS} = \frac{2^{36}}{2^{14} \times 2^{10}} = 2^{12}B \times PTE = 2^{22} \times 2^{24} = 2^{46}$
 No. of pages in PT_2

$\Rightarrow PT_3 = \left(\frac{PT_2}{PS} \times PTE \right) = \frac{2^{46}}{16 \times 2^{10}} \times 4B = 2^{26}B$

$PT_1 = (\text{No. of pages}) \times (\text{size of each entry in } PT_1)$

$PT_2 = \left(\frac{PT_1}{PS} \right) \times PTE$
 (gives no. of pages in PT_2)

$VA = 64 \text{ bits} \Rightarrow VAS = 2^{64}B$

$PS = 1MB \Rightarrow 2^{20}B$

$PTE = 4B$

$\text{No. of pages} = \frac{VAS}{PS} = \frac{2^{64}}{2^{20}} = 2^{44} \Rightarrow PT_1 = \text{No. of pages} \times PTE$
 $= 2^{44} \times 4B$
 $= 2^{46}B$

$PT_2 = \frac{PT_1}{PS} \times PTE = \frac{2^{46}}{2^{20}} \times 2^2B = 2^{28}B$

$PT_3 = \frac{PT_2}{PS} \times PTE = \frac{2^{28}}{2^{20}} \times 2^2B = 2^{10}B$

$PT_3 \text{ entries} = 2^8 \text{ (No. of pages)}$

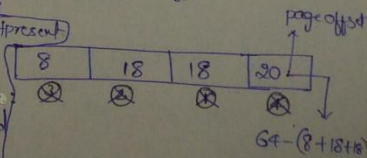
$PT_2 \text{ entries} = 2^{18}$

$PT_1 \text{ entries} = 2^{44}$

Index the outermost $PT = PT_3 = 8 \text{ bits are needed (2 pages are present)}$

$PT_3 = 4B, PS = 1MB \Rightarrow \text{No. of entries per page} = \frac{2^{20}}{2^4} = 2^{16} = 186432 \text{ entries}$

$PT_2 = 4B, PS = 1MB \Rightarrow \text{No. of entries per page} = 18$



34. EXAMPLES ON HOW TO MAKE PAGE TABLE FIT IN ONE PAGE

(76)

1) $VAS = 4MB$

$PS = 4KB$

$PTE = 4B$

$VAS = 4MB = 2^{22}B$

$PS = 4KB = 2^{12}B$

$PTE = 4B$

No. of pages = $\frac{VAS}{PS} = \frac{4MB}{4KB} = 2^{10}$

Now $PIS = \text{page table pages} * e$

$PIS = 1K * e$

Now, $PIS \leq PS$ (To fit PIS in one page)

$1K * e \leq PS$

$e \leq \frac{2^{12}}{2^{10}}$

$e \leq 4B$

2) $VAS = 4GB = 4 * 2^{30} = 2^{32}B$

$PS = 128KB = 2^{17}B$

No. of pages = $\frac{VAS}{PS} = 2^{15}$ pages

$= 32K$ pages

Now, $PIS = \text{No. of pages} * e$

$PIS = 32K * e$

$\Rightarrow PIS \leq PS$

$\Rightarrow 2^{15} * e \leq 2^{17}$

$e \leq 4B$

35. GATE 01 AND 99 QUESTION ON PAGING

1) consider a machine with 64MB physical memory and 32 bit virtual Address space. If the page size is 4KB, what is approximate size of PT?

- a) 16MB b) 8MB c) 2MB d) 24MB

$PA = 64MB = 2^{26}B$

$VA = 32 \text{ bits} \Rightarrow VAS = 2^{32}B$

$PS = PS = 4KB = 2^{12}B$

No. of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$ pages

$\Rightarrow PIS = \text{No. of pages} * e$

$= \text{No. of pages} * \text{Frame size num}$

$= 2^{20} * \text{Frame num}$

$= (2^{20} * 14) \text{ bits} = 14M \text{ bits}$

$= \frac{14M}{8} \text{ Bytes} = 2MB$

$e = \text{frame no} = (26-12)$

2) Which

a) Fastest

b) proce

c) linked

in th

d) progre

37. GATE

$PA = 36 \text{ bit}$

as follow

$\rightarrow \text{Bits} : 3$

$\rightarrow \text{Bits} : 2$

$\rightarrow \text{Bits} : 12$

$\rightarrow \text{ " } : 0$

Then PTE

a) 20, 20, 20

$PA = 36 \text{ bits}$

$PS = 4KB$

$\Rightarrow$ when we
in all the

38. GATE 20

Theory ques

41. GATE 20

$VA = 46 \text{ bits}$

$PTE = 32 \text{ bit}$

Let size of P

(76)

② Which of the following is/are advantages of VM?

$$= 2^{10} \text{ B}$$

$$= 1 \text{ K pages}$$

*e

is in one page)

- Fast access to memory on an average (All pages are in hard disk so slower)
- process can be given protected address spaces (Not specific to VM)
- linker can assign address independent of where the program will be loaded in the physical memory. (Relocation, this exists in contiguous Allocation also)
- programs larger than the physical memory size can be run.

(77)

38. GATE 2008 QUESTION ON MULTILEVEL PAGING

PA = 36 bits, VA = 32 bits, PS = 4KB, PTE = 4B, 3 level PT is used. VA is divided as follows

→ Bits: 30-31 are used to index I level PT

→ Bits 21-29 " " " " II " "

→ Bits 12-20 " " " " III " "

→ " 0-11 are used as offsets within a page

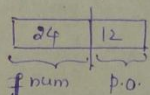
Then PTE sizes in I, II and III level PT are:

- A) 20, 20, 20 B) 24, 24, 24 C) 24, 24, 20 D) 25, 25, 24

$$PA = 36 \text{ bits}$$

of Address

$$PS = 4 \text{ KB} \Rightarrow \text{page offset} = 12 \text{ bits}$$



⇒ when we have multi level paging we use same no. of bits to represent a frame in all the levels.

39. GATE 2009 QUESTION ON MULTILEVEL PAGING

Theory question easily answerable

the num

41. GATE 2013 QUESTION ON MULTILEVEL PAGING

M bits

VA = 46 bits, PA = 32 bits, 3 level - paging, first level table has exactly one page

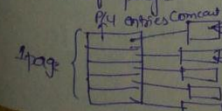
PTE = 32 bits, what is the size of the page in computer?

$$= 2 \text{ MB}$$

Let size of page = p bytes, and PTE = 4B. ⇒ No. of entries per page = $P/4$

$$\therefore VAS = (P/4)^3 * p = 2^{46}$$

$$P = 8 \text{ KB}$$



42. FINDING OPTIMAL PAGE SIZE

$$VAS = 4GB$$

$$PS = 4KB$$

$$\Rightarrow \text{No. of pages} = \frac{VAS}{PS} = 1m \text{ pages}$$

$$VAS = 4GB$$

$$PS = 4MB$$

$$\text{No. of pages} = \frac{VAS}{PS} = \frac{4GB}{4MB} = 1K \text{ pages}$$

Therefore $PS \uparrow$ $PTS \downarrow$

Page size = p Bytes

PTE = e bytes

on Average $VAS = S$ Bytes

$$\text{overheads} = \left(\frac{p}{2}\right) + \left(\frac{s}{p}\right) * e \rightarrow \text{wasted in page Table}$$

↓
The last page might not be completely filled

To minimize the overheads, differentiate w.r.t $p \Rightarrow \frac{d}{dp} \left(\frac{p}{2} + \left(\frac{s}{p}\right) * e \right) = 0$

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

$$\Rightarrow p = \sqrt{2Se} \quad \left\{ \begin{array}{l} S = \text{Avg. process size} \\ e = \text{PTE} \end{array} \right.$$

↓
optimal page size

S	e	Page size
4KB	8B	256B
16MB	2B	8KB
256GB	32B	4MB

$$\Rightarrow VAS = 4KB = 2^{12}B$$

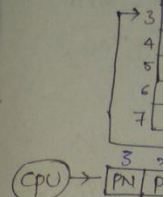
$$PTE = e = 8B$$

$$\Rightarrow p = \sqrt{2 \times 2^{12} \times 2^3} = 2^8 = 256B$$

$\Rightarrow$ In general the OS takes the process and calculate the overhead based on that formula, But some OS takes the process, divides them into sections (code section, data section, stack section) and divide each section into pages based on the above formula. Then the overhead on a particular section will be

$$\text{overhead} = \left[\frac{np}{2} + s \left(\frac{1}{p} \right) * e \right] \quad n = \text{no. of pages the section is divided into.}$$

43. VM



11
PT Base Address

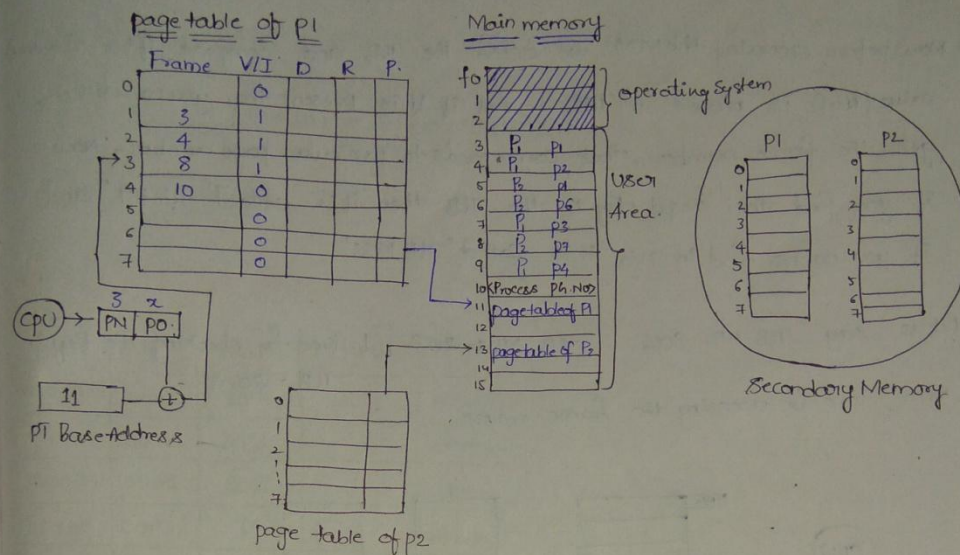
$\Rightarrow$ The VM is
 $\Rightarrow$ VM is
 $\Rightarrow$ The process
MM, In
the page
is design
P2 and

44. TLB

Now, If $VAS =$
 $\Rightarrow N$

further divided
the disad van
to access a frame
going to take a lot

43. VM INTRODUCTION



⇒ The VM concept is used when the process size is greater than the MM.

⇒ VM is provided by operating system but overlays is by Manual intervention

⇒ The process in the Secondary memory is divided into pages and loaded in the MM, In the MM $\langle P_i, p_i \rangle$ represent $\langle \text{process no}, \text{page no} \rangle$, Now to keep track of the pages of program are loaded in which frame numbers the page table is designed, and when there is context switch the control switches from p_1 to p_2 and the value in "PT Base Address" will now be 13.

44. TLB

Now, If $VAS = 46B$, $PS = 4B$

⇒ No. of pages = 1 million.

Now, the size of the pagetable will contain 1 million entries and it cannot be loaded in main memory. So the page table is also

further divided into pages and this concept is called multilevel paging but the disadvantage with this method is the memory access time, in order

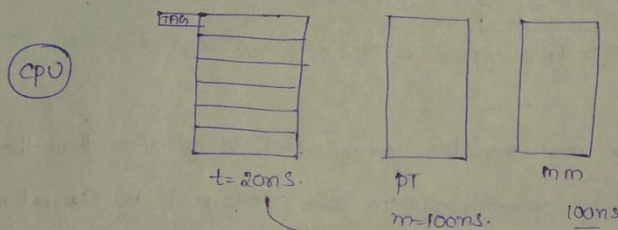
to access a frame in MM we have to access the series of page tables and it is going to take a lot of time. So they have discovered a memory (Not Registers & MM) called (CACHE) OR (TLB) and the page table is loaded in CACHE (Translation lookaside buffer).

⇒ The cache is faster than MM and cheaper than Registers.

⇒ The TLB is also sometimes called ^{MEMORY} ASSOCIATIVE ↑ Because, every ele of this memory is having 'TAG'

⇒ Now before accessing the MM we access the TLB and compare if the required value (Key) is present in TLB or not if it is present then you can directly get the frame number, that corresponds to particular page number. Now if you find the reqd key in the TLB then it is called "TLB HIT" and if you cannot find in TLB it is called "TLB MISS"

Let us say TLB HIT = 80% TLB MISS = 20%, t = time for checking the key in TLB = 20ns.
 m = time for accessing the frame in MM.



$$\text{Effective Access time} = \frac{80}{100} (20ns + 100ns) + \frac{20}{100} (20ns + 100ns + 100ns)$$

$$\boxed{EAT = 120ns.}$$

If TLB HIT = $p \Rightarrow$ TLB MISS = $1-p$

$$EAT = p(t+m) + (1-p)(t+m+m)$$

$$\boxed{EAT = p(t+m) + (1-p)(t+2m)} \quad (\text{Single level page Table})$$

If there are k levels then $\boxed{EAT = p(t+m) + (1-p)(t+km+m)}$

45. GATE

A paging takes 500

A) 54

⇒ EAT =

EAT =

46. NUM

TLBA	
20ns	10
20ns	10
20ns	10
20ns	10
20ns	10
20ns	10

1> EAT =

2> EAT =

3> EAT =

⇒ 20/10

⇒

(80)

45. GATE 2008 QUESTION ON TLB

of this

A paging scheme uses TLB. A TLB access takes 10ns and main memory access takes 50ns. What is the EAT (in ns) if TLB hit ratio = 90% and there is no page fault?

A) 54 B) 60 C) 65 D) 75

$$\Rightarrow \text{EAT} = \frac{90}{100}(50+10) + \frac{10}{100}(10+50+50)$$

$$\boxed{\text{EAT} = 65 \text{ ns}}$$

46. NUMERICALS ON TLB

TLBA	MA	TLBH	PT levels	EMAT
20ns	100ns	80%	1	140ns
20ns	100ns	80%	2	160ns
20ns	100ns	80%	3	180ns
20ns	100ns	90%	1	130ns
20ns	100ns	60%	1	160ns
20ns	100ns	50%	1	170ns

$$1) \text{ EAT} = \frac{80}{100}(20+100) + \frac{20}{100}(20+100+100)$$

$$= 96 + 44 = 140 \text{ ns}$$

$$2) \text{ EAT} = \frac{80}{100}(20+100) + \frac{20}{100}(20+2(100)+100)$$

$$= \frac{80}{100} \times 120 + \frac{20}{100} \times (320) = 96 + 64 = 160 \text{ ns}$$

$$3) \text{ EMAT} = 130 \text{ ns}$$

$$\Rightarrow \frac{x}{100}(20+100) + \left(100 - \frac{x}{100}\right)(20+100+100) = 130$$

$$\Rightarrow \frac{x}{100} \left(\frac{120}{5}\right) + \frac{100-x}{100} \left(\frac{220}{5}\right) = 130$$

$$= \frac{6x}{5} + \frac{(100-x)11}{5} = 130$$

$$\boxed{x = 90 \text{ ms\%}}$$

$$p(20+100) + (1-p)(320) = 130$$

$$\Rightarrow 120p + 320 - 320p = 130$$

$$= -100p = 130 - 320 \Rightarrow p = \frac{90}{100} = 90\%$$

(81)

47. TLB SUMMARY

(82)

TLB access time = t

Main memory access = m

TLB miss rate = $p \Rightarrow$ TLB Hit rate = $(1-p)$

$$EMAT = p(t+m+m) + (1-p)(t+m)$$

$$= p(t+2m) + (1-p)(t+m)$$

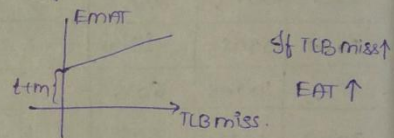
$$= pt + 2pm + t + m - pt - pm$$

$$EMAT = t + m + p(2m) \Rightarrow \text{let } EMAT = y$$

$t+m = \text{Constant} = C$

$p(m) = p \times m$
 $\downarrow \quad \downarrow$ Represented by.
 $x \quad a$

$$\Rightarrow y = C + x(m) \Rightarrow y = mx + C$$



If no. of levels = K then

$$EMAT = p(t + Km + m) + (1-p)(t+m)$$

$$= p(t + (1+K)m) + (1-p)(t+m)$$

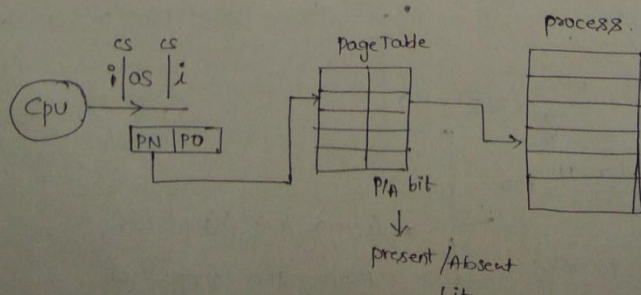
$$= pt + (1+K)pm + t + m - pt - pm$$

$$EMAT = t + m + p(Km) \quad p = \text{miss rate not Hit Rate.}$$

48. PAGE FAULT

$\Rightarrow$ Referring to a page that is not present in the memory is called pf.

$\Rightarrow$ Demand paging = Don't load any page until it is required



Now, the CPU generates the logical address and with that address we go to corresponding entry in the pagetable now if the entries corresponding P/A bit is 0 then it says that the page that you are requesting is not loaded in the main memory. Then there will be context switch (CS) from user process to operating system, now the OS try to load the Required page from the secondary storage and the context is again switched to the user process.

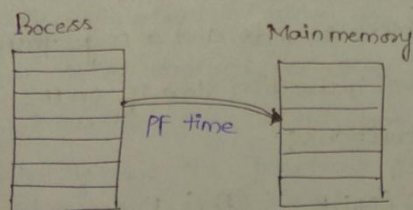
⇒ No. of page references = No. of page faults then it is called Thrashing.

⇒ EMAT in case of page fault is.
$$= p(\text{service time / page fault time}) + (1-p)(\text{memory access time})$$

$\downarrow$ page fault rate

$$\text{EMAT} = p * (\text{service time}) + (1-p)(\text{memory Access time})$$

Also called page Fault time



⇒ Now, initially the process is going to start without loading any page in MM then CPU whenever it generates 'LA' it will find out that the 'page no' is not in MM, then page should be loaded

into MM and again the same instruction will start which will generate the same logical address, the entire time (To stop the process, copy the page, again start the time is called page fault time) and again after loading again in need to refer to the page in Memory. ∴

$$\text{EMAT} = p * (\text{Service time} + \text{ma}) + (1-p)(\text{ma})$$

$\downarrow$ Very Negligible

49. GATE 2011 QUESTION ON PAGE FAULT

Let the pf service time be 10ms in a computer with average memory access time being 20ns. If one page fault is generated for every 10^6 memory access, what is the EAT for the memory?

- A) 21ns B) 30ns (C) 23ms (D) 35ms

$$\begin{aligned}
 \text{EMAT} &= p * (\text{servicetime} + \text{MA}) + (1-p)(\text{MA}) \\
 &= p * (\text{servicetime}) + [p * (\text{MA})] + (\text{MA}) - [p * (\text{MA})] \\
 &= p * (\text{service-time}) + (\text{MA}) \\
 &= \frac{1}{10^6} * (10\text{ms}) + (20\text{ns}) \rightarrow \text{Access time without page faults.} \\
 &= 10\text{ nsec} + (20\text{ns})
 \end{aligned}$$

$$\boxed{\text{EMAT} = 30\text{ns}}$$

50. GATE 98 QUESTION ON PAGE FAULT

If an instruction takes 'i' micro seconds and a page fault takes an additional 'j' μsec , the effective instruction time if, on the average a page fault occurs every K instructions?

- a) $i + (j/k)$ b) $i + j * K$ (c) $(i+j)K$ (d) $(i+j) * K$

$$\text{EMAT} = p * (\text{Service time}) + \text{MA}$$

$$\text{EIT} = i + \frac{1}{K}(j) = \text{Instruction access time} + p * (\text{service time})$$

$$\begin{aligned}
 \boxed{\text{EIT} = i + (j/k)} &= i + p * (j) \\
 &= i + \frac{1}{K} * j = (i + j/k)
 \end{aligned}$$

51. GATE 2000 QUESTION ON PAGE FAULT

Suppose the time to service a page fault is on average 10msec, while a memory access takes 1 μsec . Then a 99.99% hit ratio results in avg. memory access time of?

- A) 1.9999 msec B) 1msec

(84)

memory access

memory access,

MA = 200ns

me = 10ms

Given MA = 4μsec

Service time = 10ms

Hit ratio = 99.99% ⇒ miss rate = 0.01% ⇒ 0.0001

$$\begin{aligned} \text{EMAT} &= p * (\text{service time}) + \text{MA} = (4\mu\text{sec}) + (0.0001)(10 \times 10^{-3}) \\ &= 4\mu\text{sec} + (10^{-4} \times 10^{-2}) \\ &= 4\mu\text{sec} + 1\mu\text{sec} \end{aligned}$$

$$\boxed{\text{EMAT} = 5\mu\text{sec}}$$

(85)

52. GATE 2007 QUESTION ON PAGE FAULT AND DIRTY BIT

A demand paging system takes 100 time units to service a page fault and 300 time units to replace a dirty page. Memory access time is 1 time unit. The probability of page fault is 'p'. In case of page fault the probability of page being dirty is also 'p'. It is observed that the average access time is 3 time units. Then value of 'p' is

- a) 0.194 b) 0.233 c) 0.514 d) 0.98

⇒ Here the Dirty page represent when you are accessing a particular page check whether it is modified/not using dirty bit if you find it is modified then writeback the page in the process and then replace it again when there is a dirty page the page fault service time will increase.

⇒ service time / page fault time (in case of no Dirty pages) = 100 time units.

" " " " (" " Dirty pages) = 300 time units.

MA = 1 time unit

$$p_s = (1-p)(100) + p(300)$$

$$\boxed{\text{PFST} = 100 + 200p}$$

$$\text{EMAT} = 1 \text{ unit} + p(p_s)$$

$$3 = 1 + p(100 + 200p)$$

$$\Rightarrow 1 + 100p + 200p^2 = 3$$

$$200p^2 + 100p - 2 = 0 \Rightarrow p = 0.0194$$

53. GATE 2003 QUESTION ON TLB AND PAGING

A processor uses 2 level paging VA=PA= 32 bits, Byte addressable

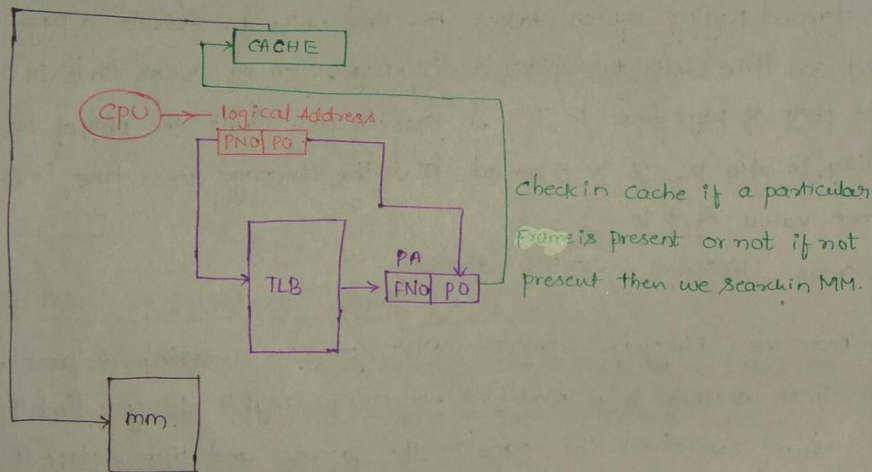
VA:

10	10	12
----	----	----

 PTE = 4 Bytes, TLB has hit rate of 96% Cache has hit rate of 90% main memory access time is 10ns, Cache access time is 1ns and TLB access time is 1ns.

a) Assuming no page faults, the average time taken to access a VA is approximately (to the nearest 0.5 ns)

- a) 1.5 ns b) 2 ns c) 3 ns d) 4 ns



$$EMAT = \text{Avg. time taken to access Virtual Address} = (\text{VA} \rightarrow \text{PA}) + (\text{fetch the word from process})$$

Take the help of TLB

if not present then go to MM.

$$\Rightarrow \text{Avg time} = \underbrace{t + (1 - p_t)(K \times m)}_{\text{conversion from (VA} \rightarrow \text{PA)}} + \underbrace{c + (1 - p_c)(m)}_{\text{fetching}}$$

$$= 1 \text{ ns} + \frac{4}{100} (2 \times 10) + 1 \text{ ns} + \frac{10}{100} (0)$$

$$\boxed{\text{Avg time} = 3.8 \text{ ns}}$$

a) Suppose two contiguous virtual addressing

Now, Address

$$\Rightarrow \text{PTE} = \text{PTE}$$

PTS

$$\Rightarrow 2 \text{ pages} =$$

54. GATE 2003

Consider access to instruction ratio is is the effective

A) 645 ns

$$EMAT = \text{CP}$$

$$EMAT = (V$$

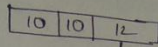
85

8) Suppose a process has only the following pages in virtual address space two contiguous code pages starting at virtual address 0x00000000, two contiguous datapage starting at VA 0x00400000 and a stack page at virtual address 0xFFFF000. The amount of memory required for storing page tables of this process

- a) 8KB b) 12KB c) 16KB d) 20KB

87

Now, Address split



→ Represent page offset ⇒ size of page = $2^{12}B$.

= 4KB.

⇒ $PT = PTS = \text{No. of pages} \times \text{Size of each PTE}$

$$PTS = 4MB.$$

⇒ 2 pages = 0x00000000 to 0x0000FFFF
to
0x00000000 to 0x0000FFFF
to
0x00000000 to 0x0000FFFF
0000000000

2 pages: 0x00400000 to 0x004000FF
0x00400000 to 0x004000FF
0x00400000 to 0x004000FF
0x00400000 to 0x004000FF
004 = 0000000000100

64. GATE 2004 QUESTION ON TLB AND PAGE FAULT

Consider a system with 2-level paging scheme in which a regular memory access takes 150 nsec and servicing a page fault takes 8 msec. An average instruction takes 100 nsec of CPU time and two memory access. The TLB hit ratio is 90% and page fault rate is one in every 10,000 instructions. What is the effective average instruction execution time?

- a) 645 ns b) 1050 ns c) 1215 ns d) 1230 nsec

$$EAT = \text{CPUTime} + 2 * (EMAT)$$

$$EMAT = (VA \rightarrow PA) + (\text{Access the byte from PM})$$

$$= t + (1 - P_f)(2 * MAT) + M + P_f * PS$$

$$= 0 + (0.1)(2 * 150n) + 150n + \left(\frac{1}{10,000}\right)(8m)$$

$$= (30 + 150 + 800) \mu\text{sec} = 980 \mu\text{sec} \Rightarrow EAT = 100ns + 2 * (980) = 2060ns.$$

$$= (30 + 150 + 800) \mu\text{sec} = 980 \mu\text{sec}$$

55. GATE 2014 QUESTION ON TLB

Consider a paging s/w with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes $10 \mu\text{sec}$ to access the TLB and 80 msec to search the physical memory. If the TLB hit rate/ratio is $0.6 = 60\%$, the EMAT is ____?

$$\begin{aligned} \text{EMAT} &= p(t+m) + (1-p)(t+tm) \\ &= 0.6(10+80) + (0.4)(10+160) \\ &= \frac{60}{100} \times 90 + \frac{40}{100} \times 170 \\ &= 54 + 68 \\ &= 122 \text{ ms.} \end{aligned}$$

56. INVERTED PAGE TABLE

Now, assume there are 10 process running, now the pagetables of all the process should be loaded now, the main memory will be occupied by 10 pagetables which will take more space so the concept of Inverted page table has been proposed. The indexes in the Inverted page table will be Frame no not the page no.

Inverted pagetable

Frame no.	
0	<P1, P1>
1	<P1, P3>
2	<P1, P5>
3	<P2, P2>
4	<P2, P3>
5	<P2, P4>
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

<process no, page no>

page	PT of P2
0	X
1	X
2	f4
3	f5
4	f6
5	X
6	X

page	PT of P1
0	X
1	f1
2	X
3	f2
4	X
5	f3
6	X

If $PA = 32$
 $PS = 4 \text{ KB}$
 $IPTE = 4 \text{ B.}$

what is the size of IPT?

$$\begin{aligned} \text{IPIS} &= (\text{No. of frames}) * \text{IPTE} \\ &= \frac{2^{32}}{2^{12}} \text{ B} * 4 \text{ B} \\ &= 2^{20} * 4 \text{ B.} \\ &= 4 \text{ MB.} \end{aligned}$$

57. IMPORTANT

Equal Allocation
 ↓
 30 frames
 3 process
 ⇒ 1 process = 10 frames

⇒ when all the it wants to used.

58. PAGE RE

Optimal page

Least Recently time

FIFO: first

59. QUESTION

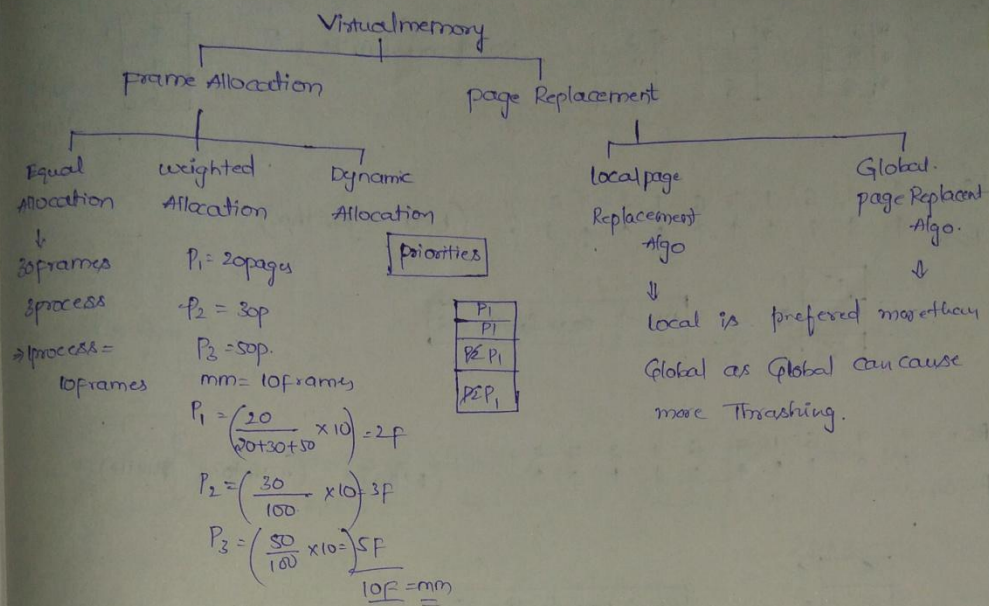
Demand paging

GATE-2014

Reference string =
 Frames = 3.

(88)

57. IMPORTANCE OF FRAME ALLOCATION AND PAGE REPLACEMENT (89)



se process

getables

s been

not

⇒ when all the frames allocated to a particular process gets filled and if it wants to load another page of it then page replacement Algorithms are used.

58. PAGE REPLACEMENT

Optimal page replacement: Replace the page that will not be referred for a long time (least no. of page-faults) (Used as Benchmark)

Least Recently used: Replace the page which has not been referred for a long time

FIFO: first inserted page should be replaced.

59. QUESTION ON PAGE REPLACEMENT ALGORITHMS 1

Demand paging is used find the No. of page faults

GATE-2014

Reference String = 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

Frames = 3.

RS = 4 7 6 1 7 6 1 2 7 2 (Optimal page Replacement Algo)

F=3 $\Rightarrow$

4
7
6

1
7
6

2
7
6

No. of page faults = 5

4
7
6

= No. of letters
= 4, 1, 2, 7, 6
= 5 PF

61. GAT
RF = 1, 2,
F = 3

RS = 4 7 6 1 7 6 1 2 7 2 } (LRU Algorithm)
F=3 $\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$

4	1
7	2
6	7

No. of page faults = 6

Optimal
RF = 1, 2,
 $\uparrow \uparrow$
F = 3

RS = 4 7 6 1 7 6 1 2 7 2
F=3. $\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$ (FIFO) (maintain queue)

4	1
7	2
6	7

No. of page fault = 6

FIFO

4
7
6

1
2

Queue

4	7	6	1	2
---	---	---	---	---

QUESTION ON PAGE REPLACEMENT EXAMPLE 2

②.

0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370
100 Records per page with 1 free main memory frame.

The above no. represents the decimal no. $\Rightarrow$ now we need to convert them into page numbers.

0100 = 01	0499 = 04	0120 = 01	0320 = 03
0200 = 02	0510 = 05	0220 = 02	0370 = 03
0430 = 04	0530 = 05	0240 = 02	
	0560 = 05	0260 = 02	

RS = 01, 02, 04, 04, 05, 05, 05, 01, 02, 02, 02, 03, 03

P=1 $\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$

$\Rightarrow$

1

~~2~~ ~~4~~ ~~4~~ ~~5~~ ~~5~~ ~~5~~ ~~1~~ ~~2~~ ~~2~~ ~~2~~ ~~3~~ ~~3~~

$\Rightarrow$ No. of page faults = 7 = No. of misses

$\Rightarrow$ Miss Rate = $7/13$ = pfrate.

$\Rightarrow$ No. of references made = 13. (length of RS)

64. BELA
RF: 0
F: 3

Optimal
F=3:

0
1
2

F=4:

0
1
2
3

PF = 6

FIFO
F=3:

0
1
2

(90)

$$\text{Hit rate} = 1 - \text{miss rate}$$

$$= 1 - 7/13 = 6/13.$$

(91)

No. of letters

4, 1, 2, 7, 6
= 5 PF

61. GATE QUESTION ON PAGE REPLACEMENT ALGORITHM 3

RF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1

F = 3

OptimalRF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

F = 3

1	
2	4
3	2

No. of page faults = 5

$$\text{PFR} = 5/10$$

LRURF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

F = 3

1	2
2	1
3	4

No. of page faults = 9

$$\text{PFR} = 9/10$$

FIFO

1	4
2	1
3	2

No. of page faults = 10

$$\text{PFR} = \text{miss rate} = 6/10 = 60\%$$

$$\text{Hit rate} = 1 - 6/10 = 4/10 = 40\%$$

1	2	3	4	1	2
---	---	---	---	---	---

64. BELADY ANOMALY

RF: 0 1 2 3 0 1 4 0 1 2 3 4

F = 3

Optimal

0	2 3
1	
2	3 4

page faults = 7

(6 < 7)

0	3
1	
2	
3	4

PF = 6

LRU

0	3 4 2
1	0 3
2	1 4

PF = 10

0	4
1	
2	3
3	2

PF = 8

(10 > 8)

(8 < 10)

FIFO

0	3 4
1	0 2
2	1 3

PF = 9

0	1	2	3	0	1	4	2	3
---	---	---	---	---	---	---	---	---

F = 4

0	3 4
1	0 4
2	1
3	2

PF = 10

Belady anomaly.

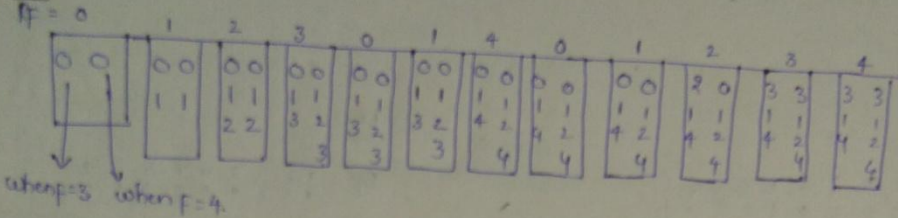
Here it is different

0	1	2	3	4	0	1	2	3	4
---	---	---	---	---	---	---	---	---	---

65. STACK ALGORITHMS

- ⇒ The main reason for Belady Anomaly is "stack property."
- ⇒ optimal property is a stack algorithm
- ⇒ whenever a Algo is a stack algorithm it does not follow Belady Anomaly

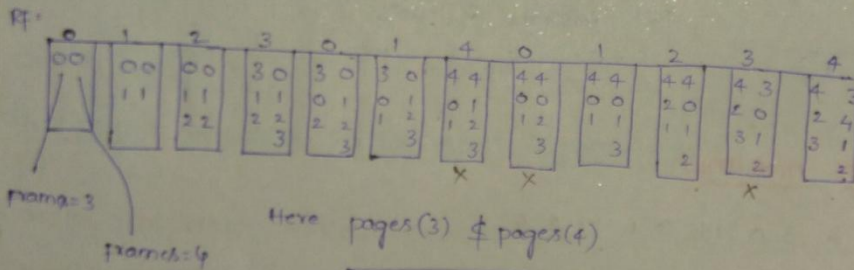
Optimal



pages(3) $\subseteq$ pages(4)

pages(m) $\subseteq$ pages(m+1) → If an Algo follows this property then it is called Stack Algorithm and it does not follow Belady Anomaly

FIFO Algorithm



pages(m) $\not\subseteq$ pages(m+1) ⇒ It is not stack Algorithm and so it follows Belady Anomaly

66. 94 QUESTION ON LRU, LFU AND FIFO

A memory page containing a heavily used variable that was initialized very early and is in constant use is removed when _____ is used

- a) LRU
- b) FIFO ✓
- c) LFU
- d) None

Heavily used: frequently used

very early = one of the oldest pages which is available

constant use = Recently used page

(92)

68. GATE 2010 QUESTION ON FIFO

A system uses FIFO policy for page replacement. It has 4 page frames with no pages loaded to begin with. The system first access 100 distinct pages in some order and then access the same in Reverse order. How many page faults occur?

a) 196 b) 192 c) 197 d) 195

1 2 3 4 5 6 7 8 100

↓
100 page-faults.

97
98
99
100

lastly the frame contains.

100 99 98 97 96 1
↓
page fault don't occur for these 4 pages because they are already present in frames

96 PF

∴ Total PF = 100 + 96

= 196 PF's.

69. SOME INTERESTING BEHAVIOUR OF OPTIMAL PAGE REPLACEMENT

⇒ In case of loops optimal and MRU (Most Recently used) are same

⇒ In case of loops both LRU and FIFO behaves in worst manner (more PF)

RS = 1 2 3 4 5 6 12 3 4 5 6 12 3 4 5 6 1 2 3 4 5 6

optimal = 12 PF

MRU = 12 PF

LRU = 24 PF

FIFO = 24 PF

1	2	3
2	3	4
3	4	5
4	5	6

12 PF

1	2	3
2	3	4
3	4	5
4	5	6

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	1

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	1

24 PF

RS = 1 2 3 4 5 6 7 8 9 { 9 8 7 6 5 4 3 2 1

⇒ optimal behaves as LRU/FIFO and both give same PF's

⇒ MRU uses a single page frame and it gives worst case

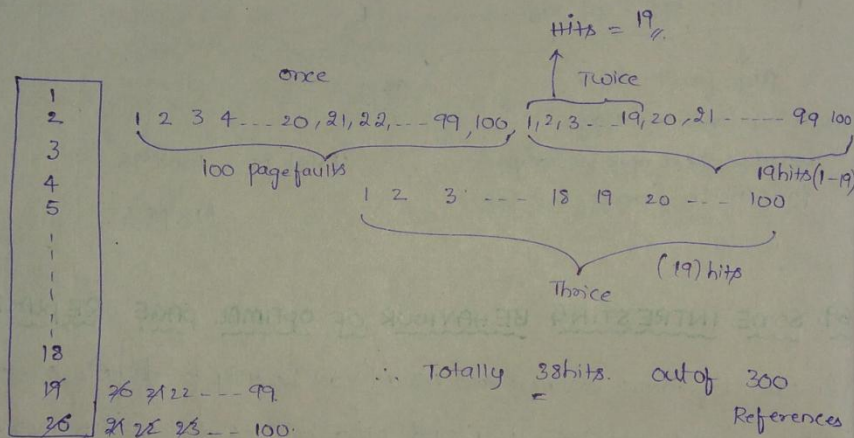
⇒ It is very difficult to implement optimal practically.

70. GATE 2014 QUESTION ON LRU, MRU, LIFO, FIFO AND OPTIMAL

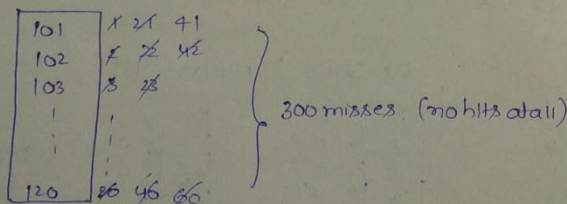
(A4)

A computer has 20 physical page frames which contains pages numbered 101 through 120. Now a program access the pages numbered 1, 2, ..., 100 in that order and repeats the access sequence thrice. which one of the following page replacement policies experience the same no. of page faults as optimal page replacement policy for this program?

- a) LRU
- b) FIFO
- c) LIFO
- d) MRU



a) LRU.



b) Similarly 300 misses

c) Similarly also 300 misses. The actual Que should be which of the following give same pattern of page faults. then option 'd' will be the ans

71. WOR

Initially frames

$\Rightarrow$ This a

RS: 1 2

window

Now

w=

w=

w=

w=

w=

Ex:

a d e | c

w = {a, d, e}

A = 4

w = {a, d, e}

w = {d, e, c}

w = {d, e, c}

(14)

3.1. WORKING SET ALGORITHM

(95)

bered

100 in

the following

optimal

Initially the process requires less frames but gradually the necessity of the frames increases.

⇒ This algo has the parameter "window size (A)"

RS: 1 2 3 1 2 4 1 4 2 1 5 1 2 4 2 1

Window Size $A = 4$.

Now working set when $A=1$ is $\{1\}$

$A=2$ is $w = \{1, 2\}$

$A=3$ is $w = \{1, 2, 3\}$

$A=4$ is $w = \{1, 2, 3\}$

working set size $\leq$ window size

$w \subseteq A$

$w = \{1, 2, 3\}$ $A = [2, 3, 1, 2]$

$w = \{1, 2, 3, 4\}$ $A = [3, 1, 2, 4]$

$w = \{1, 2, 4\}$ $A = [1, 2, 4, 1]$

$w = \{1, 2, 4\}$ $A = [2, 4, 1, 4]$

$w = \{1, 2, 4\}$ $A = [4, 1, 4, 2]$

⇒ proposed to allocate frames dynamically
⇒ very difficult to implement practically.

Ex:

ade | c c d b c e c e a d

$W = \{a, d, e\}$ = Initially window's Snapshot

$A = 4$

$w = \{a, d, e, c\} = 4F$

$w = \{d, e, c\} = 3F$

$w = \{d, e, c\} = 3F$

$w = \{c, b, d\} = 3F$

$w = \{c, b, d\} = 3F$

$w = \{d, b, c, e\} = 4F$

$w = \{b, e, c\} = 3F$

$w = \{c, e\} = 2F$

$w = \{a, e, c\} = 3F$

$w = \{c, e, a, d\} = 4F$

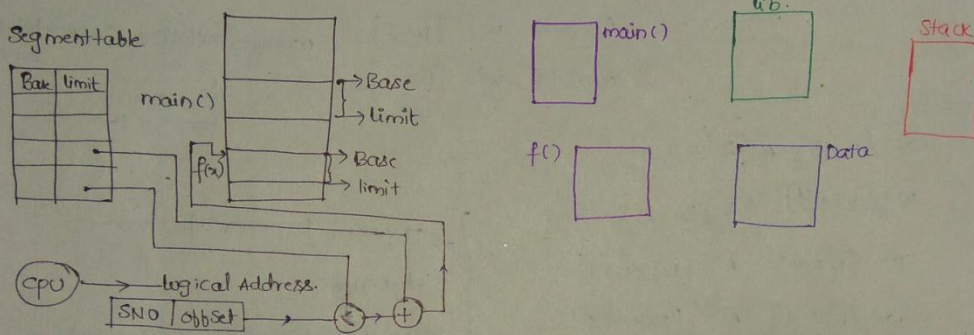
Avg. Frame Requirement = $\frac{4+3+3+3+3+4+3+2+3+4}{10}$
 $= 3.2$

15. SEGMENTATION

⇒ When we do paging some useful instructions of the program may fall in different pages which are supposed to be together. So segmentation is used

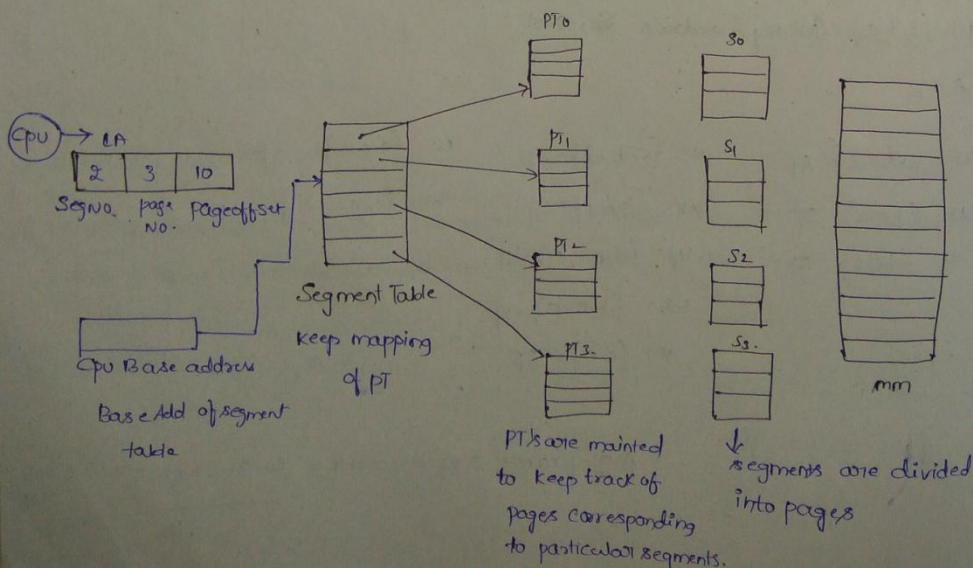
⇒ Segmentation gives the impact of division as how we assume the division should be

⇒ The program is divided into no. of segments and each segment is loaded in main memory. It has Base and Limit Registers that says the starting Address of a particular segment and end of a particular segment



⇒ offset should always be less than limit

76. SEGMENTED PAGING



77. GATE 99

LA = 32 bits

⇒

SNO	PNO	PO
-----	-----	----

↓ ↓ ↓
3bits 3bits 3bits

Now, PTS

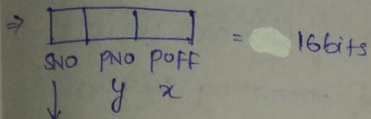
2^x

⇒ 2

77. GATE 99 QUESTION ON SEGMENTATION PAGING

97

1A = 32 bits



3 bits because 8 segments are present ⇒ $3 + x + y = 16$

$$\Rightarrow \boxed{x + y = 13}$$

Now, $PTS = \text{No. of pages} * (\text{PTE})$

$$2^x = 2^y * 2$$

$$\Rightarrow 2^x = 2^{y+1} \Rightarrow x = y + 1$$

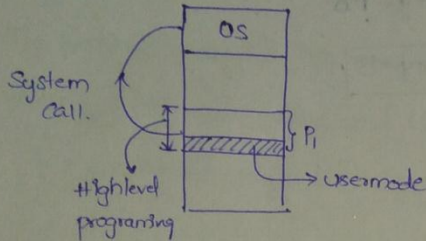
$$\Rightarrow x + y = 13$$

$$\Rightarrow y + 1 + y = 13 \Rightarrow \boxed{y = 6, x = 7} \therefore \text{Size of page} = 2^x = 2^7 = 128B.$$

6. THREAD AND SYSTEM CALLS

1. SYSTEM CALLS VS FUNCTION CALLS

⇒ System call is same as the function call but then in system call we will be trying to call the function which is present in the operating system.



```
printf()
{
    write()
}
```

2. PROCESS CONTROL SYSTEM CALLS

Process control

- End, Abort
- load, Execute
- Create process, Terminate process
- Get process attributes, set process attributes
- wait for time
- wait event, Signal event
- Allocate, and Free memory

System calls required to control a process.

3. FILE MANIPULATION

- create file, delete file
- open, close
- Read, write, reposition
- Get file attributes, (set file attributes.)
 - if you are owner of file you will be provided to manipulate the file
 - (date of modification), (owner of file), size of file,
 - (date of creation), (permissions)

7. FORK

⇒ fork()

⇒ when

⇒ when and

⇒ fork for

8. PROCES

Web

Stack
Register set
PC
data code

Thread: No
block

9. USER

→ The disc

Kernel level

7. FORK SYSTEM CALL

⇒ `fork()` / `execat()` is the system call

⇒ when any process calls `fork()` system call it will create a child process

○ parent process
○ child process

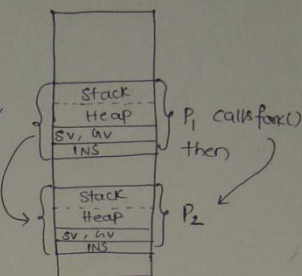
⇒ when `fork()` is called there should be context switch and we should shift from user mode → system mode
Overhead.

⇒ `fork` returns `i=0` for child process.

`fork` returns ○ for parent process

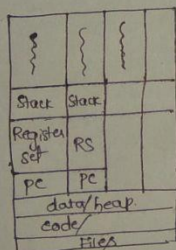
↓
`pid of child`

} `fork()` creates two process (one is already existing one and new one)



8. PROCESS VS THREADS

Web server



process (<code>fork()</code>)	Thread (user level)
→ System calls are involved	→ No system call
→ context switch	→ Register set switch
→ diff. copies of code and data	→ same copies of code and data

Thread: New, Ready, Running, blocked, Terminated

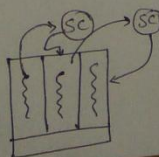
9. USER LEVEL V-S KERNEL LEVEL THREADS

→ The disadvantages of user level threads is

→ Blocking system call will block the whole task.

→ unfair scheduling

Kernel level threads



⇒ parallelism is guaranteed.

⇒ Disadv: Expensive compared to UI

⇒ switching requires more time (requires system call for switching)