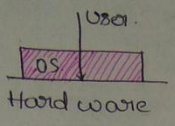
ekxam.com

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:

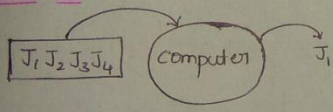
primary goal = Convienience

Secondary goal = Efficiency.

Types of operating system

Batch OS, Multiprograming, Multitasking, multiprocess, Realtime.

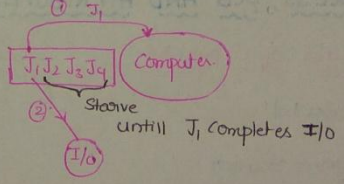
Batch OS



⇒ 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.

Multi Programming

⇒ cpu will not be kept idle

⇒ whenever a Job needs I/o then the cpu takes the next process and process it.

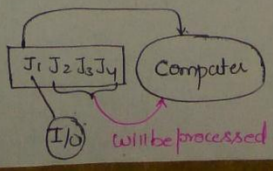
∴ cpu will be Busy all the time.

⇒ No Starvation.

⇒ Efficiency is more

⇒ More Throughput.

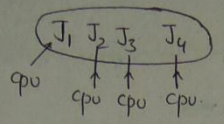
⇒ No- Preemption.



Multi Tasking:

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

⇒ preemption is present.



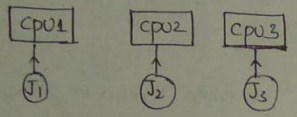
Multi processing:

⇒ Instead of single cpu we will have lots of cpu's 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 cpu's) Quad core (4 cpu's)

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

Realtime:

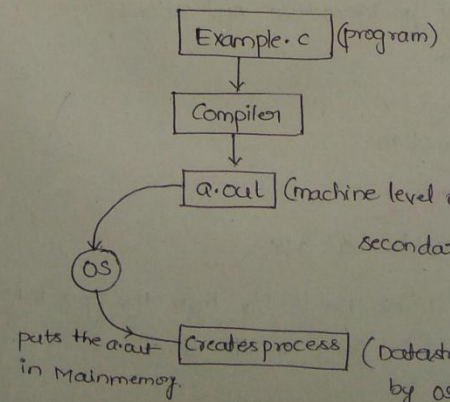
⇒ we will have jobs with Deadlines.

⇒ Result produced after the deadline is waste

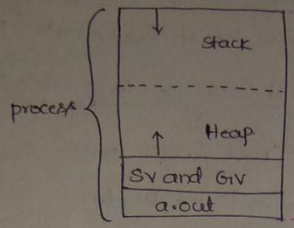
2. PROCESS, PCB AND ATTRIBUTES

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.



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

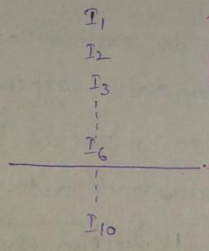


SV = static variables
GV = Global variables
Heap = For dynamic allocation malloc(), calloc() etc.

⇒ The execution should not cross process Boundaries, if it crosses we get "segmentation fault."

→ process Id: Unique no. that is given to a process to identify it.

→ program counter: Stores the starting point of next Instruction.

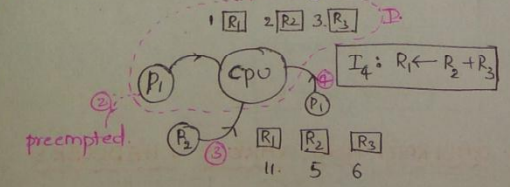


Executed till here and then we stopped this process and tried to execute the next one and again whenever we come to this process we start from I_7 and I_7 will be stored in program counter.

→ process state: New, Ready, Running, Blocked

→ priority: Number that is assigned to the process whenever it is created.

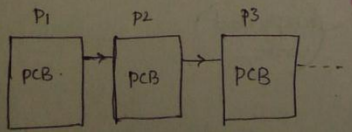
→ GPR:



→ List of open files: While execution some files are opend for Reading and some for writting we should remember which files we have read/ which files we have written.

⇒ All this information will be present in process control Block

⇒ Every process gets its own PCB.



All the PCB are linked using linked lists.

3. PROCESS STATES AND MULTIPROGRAMMING

States of process:

1> New = program present in Sencondary 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 - Mainmemory - Run → Block /wait state

Block /wait → Ready

Redy → 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 mainmemory should be suspendend and. Room must be made for newprocess.

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.

Ready, Run, Blockandwait - Mainmemory

New - Secondary memory.

Operations on processes

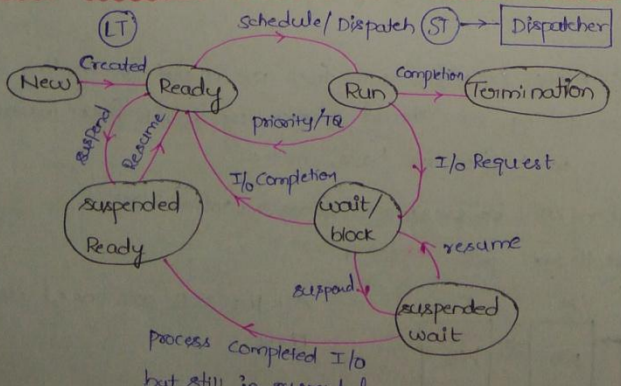
1> Creation

2> Scheduling

3> Execute it

4> Killing/ Delete

4. PROCESS STATE TRANSITION DIAGRAM AND VARIOUS SCHEDULERS



TQ= Time Quantum

LT= long Term scheduler

ST= short term scheduler

MT= Medium Term scheduler

⇒ Degree of

State at

⇒ Short Term

deciding

for movin

" Context-

⇒ The medi

is the

memory

5. PROCESS

6. QUESTION

Consider a sy

Stat
Read
Runn
Block

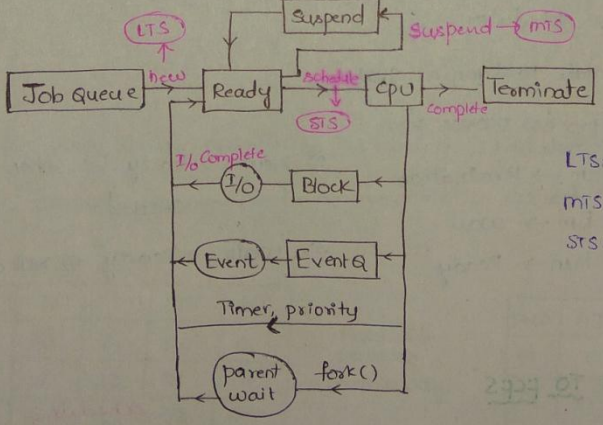
⇒ Degree of multiprograming = No. of processes that can be present in the Redy state at maximum. (This factor is decided by Long Term scheduler)

⇒ Short Term scheduler decides which process must be executed next and after deciding it will call "Dispatcher". Now, Dispatcher is the slw 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 swit ching time
Medium Term scheduler - Impacts Swapping Time

5. PROCESS QUEUES



LTS = Long Term Scheduler
MTS = Medium " " "
STS = short Term scheduler

6. QUESTION ON PROCESS STATES

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

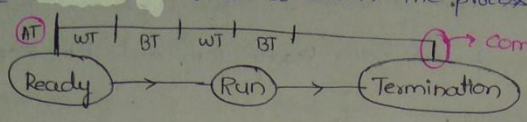
State	min	max
Ready	0	M
Running	0	N
Block	0	M

7. 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

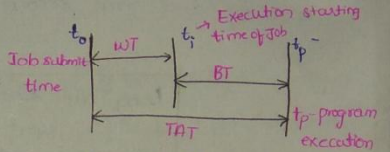
3) Completion time : The time at which the process finishes execution



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

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

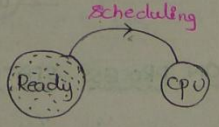
2) New → Ready i.e when a process is just created

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

9. INTRODUCTION TO FCFS

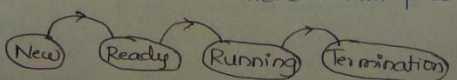
Criteria : Arrival Time

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



Gannt chart

⇒ In case

lower p

⇒ IN the

waiting ti

10. CONVOY

⇒ when a

all the

PNO
1
2
3

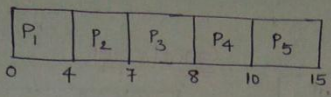
Gannt Chart

P_1
0

Avg WT = 19

Avg TAT = 20

Gannt chart :



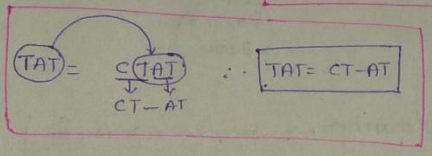
⇒ In case if 2 process have same arrival times then pick the process with lower process id/PNO.

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

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

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

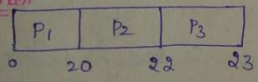
⇒ when a process with heavy Burst time is being scheduled by the cpu then all the other process are going to starve, this is called "convoy Effect."

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

Gannt chart

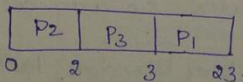


Avg WT = $\frac{19+21}{3} = \frac{40}{3}$

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

$= 21$.

Gnatt chart



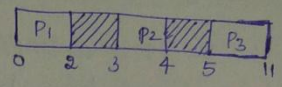
Avg. WT = $\frac{0+2+2}{3} = 4/3$.

∴ 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



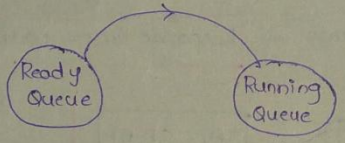
↳ Here no process is waiting for cpu, Infact cpu is waiting for the processes.

⇒ Queue is the simplest Data structure that is used to Implement FCFS.

12. FCFS WITH OVERHEAD

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

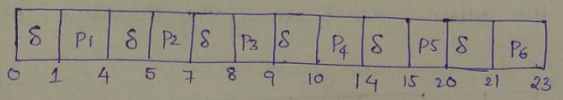
⇒ The context switching time (δ) is when a process is running, we are going to stop it and schedule the other process. In between many things will happen Now,



{ 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

Gannt chart



⇒ Before the first process is called the scheduler will be called which inturn 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.

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

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

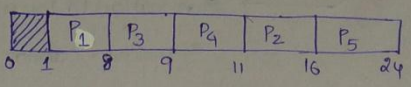
13. INTRODUCTION TO SJF

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

Gannt chart



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

⇒ victim of convoy effect.

⇒ The datastructure used is "Minheap." (Root will be the process with least Bursttime)

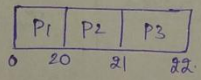
14. ANALYSIS OF SJF

⇒ 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

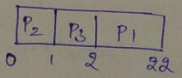
Avg. WT = $39/3$ = 13ms.

Throughput = $3/22$



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

Gant chart :



Avg. WT = 0.

Throughput = $3/22$ (3 → No of process, 22 → Total schedule time)

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

15. SJF WITH PREDICTION OF BT

SJF
- Advantages → Maximum Throughput, Min Avg WT and TAT
- Disadvantages → Starvation to longer jobs, Not implementable because BT of a process cannot be known ahead.

Solution: **SJF with predicted BT's.**

Prediction Techniques
- Static
  - process size
    - P_{old} = 200KB = 20 units
    - P_{new} = 201KB = 20 units
  - process Type
    - OS = (3-5) units
    - User
      - Interactive (5-8)
      - Foreground (10-15)
      - Background (15-20)
- Dynamic
  - Simple Averaging
  - Exponential Averaging / Aging.

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

Simple Averaging

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

→ 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

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

t_n = Actual BT, τ_n = predicted BT, α = smoothning factor

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

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

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

$\tau_5 = 6.875$

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

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

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

$\tau_1 = 10$.

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 sch

because

EXAMPL

PNO
1
2
3

18. EXAMPLE

PNO
1
2
3
4

16. INTRODUCTION TO SRTF

SRTF = Shortest Remaining Time first.

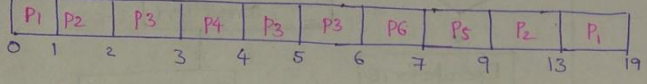
Criteria = BT's.

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 6 0
2	5 4 0
3	3 2 1 0
4	1 0
5	2 0
6	1 0

Gantt chart :



⇒ cannot be practically implemented.

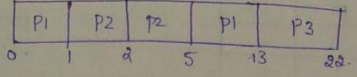
⇒ Any scheduling Algorithm related to BT cannot be implemented practically because BT cannot be predicted accurately.

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?

Avg. WT = 15/3 = 5ms.

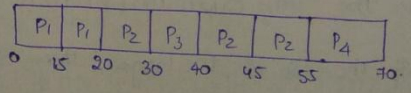


18. EXAMPLE ON SRTF GATE 2007

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?

Gannt chart



∴ WT of P_2 = 15

19. ROUND ROBIN ALGORITHM

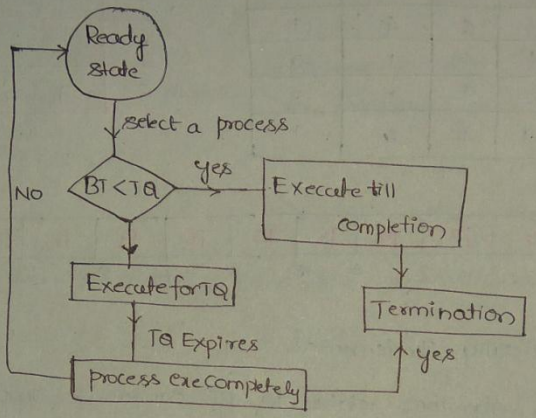
⇒ Many of the operating systems in practice use Round Robin scheduling.

⇒ 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)

ND 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: P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_5 P_2 P_6 P_5

Gant chart: P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_5 P_2 P_6 P_5

0 2 4 6 8 9 11 13 15 17 18 19 21

context switch.

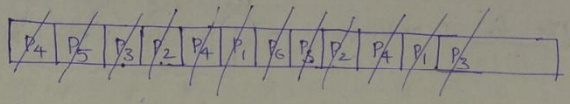
⇒ As the Time Quantum [↑] then no of context switches [↓]

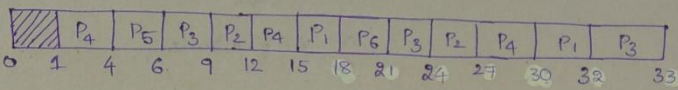
⇒ If the TQ is too large then Starvation occurs.

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

Process Queue : 

Gannt chart : 

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

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

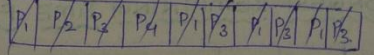
TQ	CS	RT
↑	↓	↑
↓	↑	↓

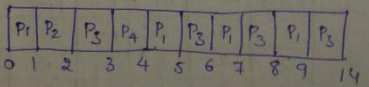
If TQ = ∞ then, Round Robin will become FCFS.

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

Queue : 

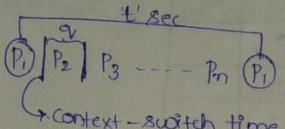
Gannt chart : 

$\therefore (CT)_{P_1} = 9$

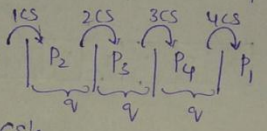
23. ROUND ROBIN EXAMPLE 4

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 guarenteed to get the turn at the cpu for every 't' seconds?

Let the processes be



Now, let's analyze for 4 process



⇒ 4 CS's

⇒ 3 quantums.

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

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

24. LONGEST JOB FIRST

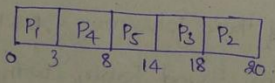
⇒ process having longest BT gets scheduled first

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 Gannt chart is:



25. LONGEST REMAINING TIME FIRST

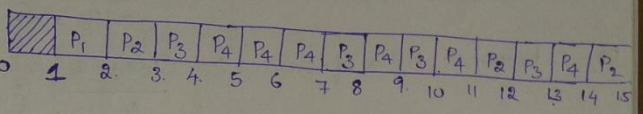
PNO	AT	BT	CT	TAT	WT	RT
1	1	2	18	17	15	0
2	2	4	19	17	13	0
3	3	6	20	17	11	0
4	4	8	21	17	9	0

itching time
f context
to get the

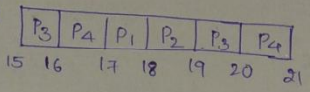
one process

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

The Gant chart will be



Ginat chart continued



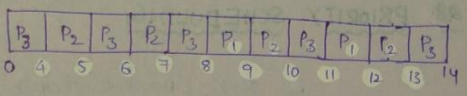
26. LONGEST REMAINING TIME FIRST GATE 2006 QUESTION

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 X 0
2	0	4 3 2 X 0
3	0	8 7 3 2 X 0

Gannt chart:



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

27. HIGHEST RESPONSE RATIO NEXT

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

w = waiting time for a process so far

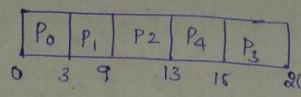
s = service time of a process or BT.

→ 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

Gant chart:



Now at time t = 9ms.

RR_2 = Response ratio of process 2 = $\frac{(9-4)+4}{4} = (2.25)$

→ P_2 arrived at 4 and now the time is 9 so the waiting time till now = 5 = (9-4) //.

↓

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 = 13ms

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

$RR_4 = \frac{5+2}{2} = 7/2 = (3.5)$ → P_4 will be executed after t = 13ms.

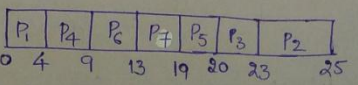
28. PRIORITY SCHEDULING

priority
- static (Doesnot change throughout the execution of process)
- pre emptive
- Non- pre emptive
- Dynamic (changes at Regular intervals of time)

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

Gant chart:



30. PR

PNO
1
2
3
4
5
6
7

Gant c

⇒ Run

31. SR

PNO
1
2
3
4

Read

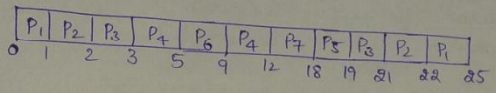
34. Mu

30. PRE-EMPTIVE PRIORITY SCHEDULING

PNO	priority	AT	BT	CT	TAT	WT	RT
1	2	0	4	25	25	21	0
2	4	1	2	22	21	19	0
3	6	2	3	21	19	16	0
4	10	3	5	12	9	4	0
5	8	4	1	19	15	14	14
6	12	5	4	9	4	0	0
7	9	6	6	18	12	6	6

Throughput = 7/25

Gant chart:

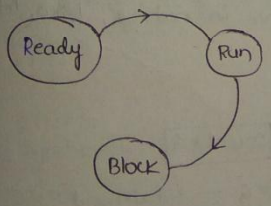


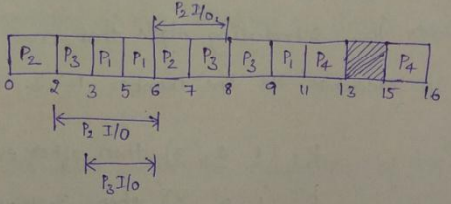
⇒ Run the process until a process with higher priority appear.

31. SRTF WITH PROCESS CONTAINS CPU AND IO TIME EXAMPLE 1

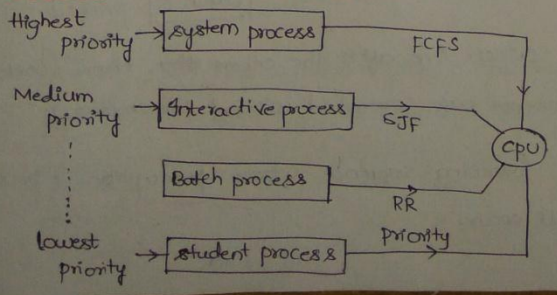
PNO	AT	BT (cpu BT)	IOBT	BT (cpu BT)
1	0	3	2	2) = 5 (sum of cpu BT's)
2	0	2	4	1) = 3 (" " ")
3	2	1	3	2) = 3 (" " ")
4	5	2	2	1) = 3 (" " ")

IOBT = I/O Burst Time





34. MULTILEVEL QUEUE SCHEDULING



⇒ There will be starvation for lower level Queues.

⇒ For various types of process various scheduling Algo's can be applied, this is the advantage.

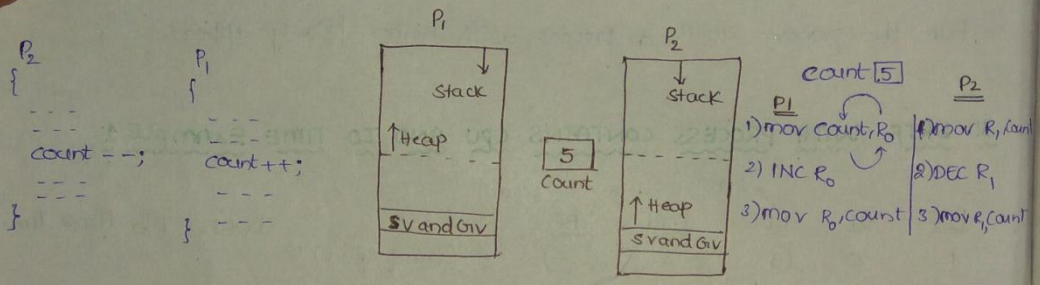
2. PROCESS SYNCHRONISATION

1. NEED FOR SYNCHRONISATION

Interprocess communication and synchronisation

⇒ whenever two process need to communicate they need to communicate using shared memory, whenever they communicate using shared memory there may be some problem / some inconsistency.

⇒ Consider two process P_1 and P_2 in the same system need to access the shared variable "count" and 'P_1' increment the count value and P_2 decrement the count value.



The the 1st Instruction in the Assembly language code is "Reading" (mov, count, R_0)

2nd Instruction " " " " " " "updating" (INC, R_0)

3rd Instruction " " " " " " "changing" (mov R_0, count)

Now, consider the scheduling Algo's that are preemptive in nature, Now, let us say P_1 executes its line number 1, 2, 3.

P_1: (1 2 3) then after execution count = 6

P_2: (1 2 3) then Initially count = 6 — 1 statement

count -- = 5 - 2nd Stmt.

count = 5 - printed

If these two process occur one after the other then there wont be any inconsistency (If the process are executed without pre emption)

⇒ Now, consider the following sequence where preemption is present. because of which some fault occur.

P_1 : 1, 2 | P_2 : 1, 2, 3 | P_1 : 3 (| = preempted)

1) count = 5
2) count = 4
3) R_1 = 4

Here the count value is incremented but it is not written/updated in the Register(R_0) so if P_2 tries to read count it will read value '5'.

3) mov, R_0, count

$R_0 \leftarrow 6$ //.

But '6' is the wrong ans

'5' should be the Right ans.

Count = 6

Now, another scheduling is. P_1 : 1, 2 | P_2 : 1, 2 | P_1 : 3 | P_2 : 3.

- P_1 : 1, 2 → count = 6 but not updated
- P_2 : 1, 2 → count = 4 but not updated
- P_1 : 3 → count = 6 updated
- P_2 : 3. → count = 4 updated

∴ The final value of count = 4

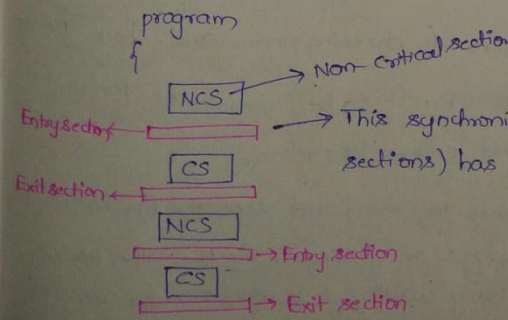
⇒ Now, all the above conditions are called "RACE CONDITIONS."

⇒ The part of the code inside the process that acess 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). Therfore there must be synchronisation mechanism in such away that if one process is acessing the critical section then other process is not allowed to acess the critical section (only one process is allowed to acess critical section at any instant of time).

2. INTRODUCTION TO SYNCHRONISATION MECHANISMS

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



3. CONDITIONS FOR SYNCHRONISATION MECHANISIMS

Requirements for synchronization mechanisims

Primary : Mutual Exclusion, Progress

secondary : Bounded waiting , portability (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 acess 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 one one platform and fails on another platforms.

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

4. LOCK VARIABLES

→ software mechanisim implemented in user mode
→ Busy waiting solution
→ Can be used even for more than two process

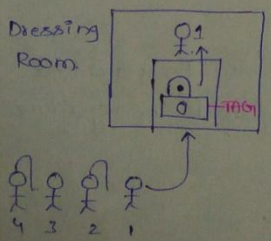
Entry section
```
While (lock != 0);
Lock = 1
```
Critical section

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 P_2 will be busy waiting for

Now, say P_1: 1 2 3 | P_2: 1 | P_1: 4 | P_2: 1 2 3

Busy wait (Infinite loop)

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

3. Critical section

Exit
4. lock = 0;

→ P_1 comes initially with lock = 0.

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

→ Now P_2 arrives and checks and finds lock = 1 (CS is occupied by some another process) and keeps on waiting.

→ Now, P_1 executes line 4 and makes lock = 0 (Room is vacant)

→ P_2 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, R_0
2) cmp R_0, 0
3) Jump if Not Zero (JNZ) to step 1
4) Store #1, Lock

CS

Exit section ← 5) Store #0, lock.

Now, Initially Lock = 0

preempted | preempted

P_1: 1 | P_2: 1, 2, 3, 4, CS | P_1: 2, 3, 4 CS

P_1 will not look at the Tag

} No Mutual Exclusion.

5. TSL (TEST - SET - LOCK)

1) Load Lock, R_0
2) store #1, lock

If preemption occurs here then the mutual exclusion fails, The soln to this is steps 1,2 should be executed as single unit (Atomic compression) and btn 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)

3) Cmp R_0, #0.
4) JNZ step 1

Now, what TSL does is, if i write TSL Lock, R_0 then it will load the lock value at R_0 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 → P_1

Exit section

Now, → P_1 is executing CS and priority of $P_1 = 0$

→ Now a process 'P_2' will be having high priority than 'P_1' will try to execute CS.

→ Now the scheduler preempts 'P_1' and schedule P_2.

→ Now, P_2 is blocked by the Entry section and P_2 will never be executed until 'P_1' completes but 'P_1' 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(P_1) and other is in running state(P_2)

6. 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

1> TSL never causes Deadlock. (TRUE)

2> Starvation is possible

P_1	P_2	P_1	P_3	P_2	P_3	P_4
CS	(while loop)	x=0	CS	(while loop)	x=0	(CS)

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). (If FIFO is guarenteed there wont be any starivation)

4> False, TSL guarenteess 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;
}
Release lock (L){
    L=0;
}
```

This implementation

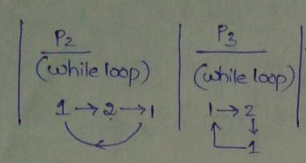
a) fails as 'L' can overflow

b) fails as 'L' can take non zero value when the lock is actually available

c) works correctly but may starve some process

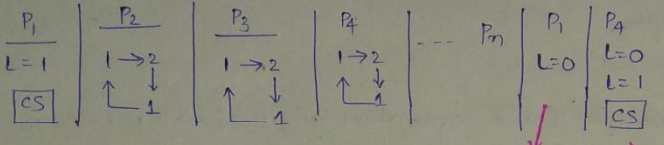
d) works correctly without starvation.

Now, Assume Initially L=0. Now, P_1 got the chance to execute CS.



Now, Any process which is in accessing the while loop will get blocked as P_1 is executing CS.

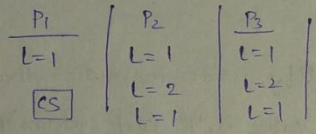
Now at some point of time 'P_1' leaves the critical section then. any of P_2, P_3, P_4 ... P_n have the chance to enter the critical section, it need not be P_2 (FIFO order)



P_1 left the critical section

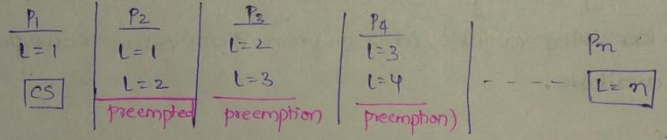
Let us say P_4 entered the critical section. so 'P_2' will starve

Now,



This occur if there is no preemption

Now, if there is preemption at this point

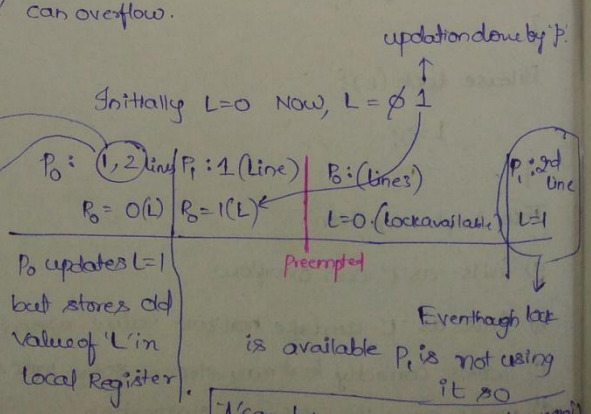


∴ 'L' can overflow beyond its bounds.

∴ This implementation fails 'L' can overflow.

Now,

```
Aquire lock(L){
1. while(fetch_and_add(L,1))
2. L=1;
}
   critical section
Release lock(L){
   3. L=0;
}.
```



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 shares boolean variables S_1 and S_2 are randomly assigned

Method used by P1	Method used by P2
while (S_1 == S_2); CS $S_1 = S_2$;	while (S_1 != S_2); CS $S_1 != S_2$

⇒ If (S_1 == S_2) 'P1' will fall in Infinite loop (; is imp)

⇒ If (S_1 != S_2) P2 will fall in Infinite loop.

Which of the following is true?

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

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

∴ 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 intrested to enter CS right now at at this point of time 'P_1' tries to enter CS but 'P_1' is blocked because 'P_2' is turn. ∴ progress is not guaranteed. (A process (P_2) which is not intrested to enter CS is not allowing another process to access it. which is intrested to enter CS (P_1)

10. DISABLING INTERRUPTS

One of the way of handling the Synchronisation mechanism is "Disabling the Interupts": A process should disable the interupt before accessing the CS.

⇒ If we disable interupt preemption is not possible

Disabling the Interupts.
CS
Enable the interupt

⇒ Both ME and progress is Guarenteed.

⇒ Bounded waiting is not gaarenteed unless you maintain the Queue.

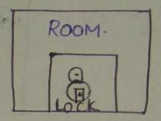
⇒ Architecture Dependent

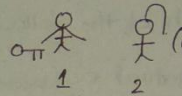
11. TURN VARIABLE OR STRICT ALTERATION METHOD

Strict Alteration Approach or Turn variable

→ S/w mechanism implemented at user mode
→ Busy waiting solution.
→ 2-process solution (P_0, P_1) or (P_i, P_j)
→ Mutual Exclusion is guaranteed but progress is not guaranteed
→ Bounded wait is present (Turn)
Busy waiting is Absent

For process P_0: (int turn)	For process P_1:
Non critical section	Non-CS.
while (turn != 0);	while (turn != 1);
CS	CS
turn = 1;	turn = 0;
Non CS.	Non-CS.





After 1, 2 should go :.
if 1 is not intrested to go and he is not giving Key to '2' then P_1 is blocking 'P_2'.
⇒ NO progress.

⇒ Cannot be implemented for more than 2 process.

Global Variable

P_0	(int turn)	P_1
Non CS		Non-CS
while (turn != 0);		while (turn != 1);
CS		CS
turn = 1;		turn = 0;
Non CS		Non-CS.

If turn = 0 ⇒ P_0 enters the CS.
If turn = 1 ⇒ P_1 enters the CS.

Now, Incase of Lock variables

LV	Turn variable
L = 0 – P_0, P_1 has chance to enter CS.	0 – P_0 can enter the CS
L = 1 ⇒ No chance	1 – P_1 can enter the CS.

Now, lets try by adding another variable
Now, let us consider another variable Intrested[0] = T → I'm intrested to enter CS
Intrested [1] = F → I'm (P_1) not intrested to enter CS.

Now) Before entering the critical section a process should say whether it is intrested / not intrested to enter the critical section. (27)

P_0

Non CS.

Entry
```
Intrested[0]=T
while(int[1]==T);
```
CS

Exit
```
Int[0]=F
```

P_1

Non-CS.

Entry
```
Int[1]=T
while(int[0]==T)
```
→ We are checking whether the other process is also intrested / not because it might already be present in CS / it might have shown intrest previously.

CS.

Exit
```
Int[1]=F
```

⇒ By the above manipulations ME, progress is guarenteed, Bounded wait (Not possible)

Deadlock is possible

P_0	P_1	P_0	P_1
$I_0=T$	$I_1=T$	$I_0=F$	$I_1=T$
CS	ω_2	$I_0=T$	ω_2

Here both the process are intrested and both will enter the cs.

Both process are Intrested.

↓

If P_0 again wants to access the CS then ($I_0=T$) will be set

12. INTRESTED VARIABLE

Same concept in video 11. (Repeated).

13. PETERSON SOLUTION

→ Sw mechanism at user mode
→ Busy waiting solution
→ 2-process solution (P_0, P_1)
→ It uses (Turn + intrested) 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 intrested[N] = FALSE
int turn;
void Entry_Section(int process)
{
    int other;
    other = 1 - process;
    intrested[process] = TRUE;
    turn = process;
    while(intrested[other] == TRUE && TURN = process);
}
void Exit_section(int process)
{
    intrested[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 process

Ex1:

intrested $\overset{0}{[T]}$ $\overset{1}{[T]}$

F

Turn [0] 1.

P_0: 1 2 3 4 5 CS | P_1: 1 2 3 5 (loop.)

P_0: 6 | P_1: 5 CS

Both ME, progress are guaranteed.

Ex;2: intrested $\overset{0}{[T]}$ $\overset{1}{[T]}$

Turn [1] 0 1

P_0: 1 2 3 | P_1: 1 2 3 4 5 (loop.) | P_0: 4 5 (loop.) | P_1: 5 CS.

preempted

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 N2
#define TRUE 1
#define FALSE 0.
int intrested[N] = FALSE;
int turn;
void - Entry section(int process)
{
1) int other;
2) other = 1-process;
3) intrested[process] = TRUE;
4) turn = process;
5) while(intrested[other] == TRUE &&
                    TURN = process);
}
void Exit_section(int process)
{
6) intrested[process] = FALSE;
}
```

15. SYNCHRONISATION MECHANISMS WITHOUT BUSY WAITING

⇒ 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 Buffers are full the producer thinks that the consumer will wakeup him and producer also sleeps. ultimately both endup in sleeping forever. so we need to have "Integer" that counts the no of wakeupcalls() and if any process wakeups 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 userlevel 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 acess the CS at the same time some CS are/can be acessed by many process.
    Queue type L;  ——→ Represent the set of process that are blocked whenever they try to enter the critical section because they dont have permission.
}
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();
    }
}
```

Entry Section

Critical section

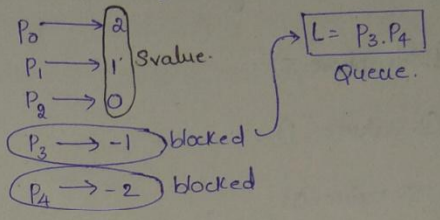
Exit Section.

Now say three processes can access the CS at a time say P_0 P_1 P_2.

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

→ when a process exists then it increments the Semaphore value and Incase if semaphore value is less than Zero. then it can understand that some process 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, 6p and 4v

⇒ 10 − 6 + 4

= 8//

(2) S=7, then 20p, 15v

⇒ 7 − 20 + 15

= 22 − 20

= 2//

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;
        wake up();
    }
}
```

21. GATE 92 QUESTION ON MUTEX

```
mutex=1
P1 = 1,2 --- 9
while(1)
{
    Down(mutex)
    <cs>
    up(mutex)
}
```

```
P1= 10
while(1)
{
    up(mutex)
    <cs>
    up(mutex)
}
```

How many process can be present in cs at max?

Now,

mutex = 1

P_i = 1, 2 - - - 9 → P_1

while (1)

{

Down (mutex) ⇒ mutex = 0.

<cs> (P_1)

up (mutex)

}

→ {P_2 P_3 P_4 P_5 - - P_9} ①

Now the other process {P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9} are in Queue.

P_{10}.

while (1) ②

{

up (mutex) mutex = 1.

<cs> P_{10}

up (mutex)

}

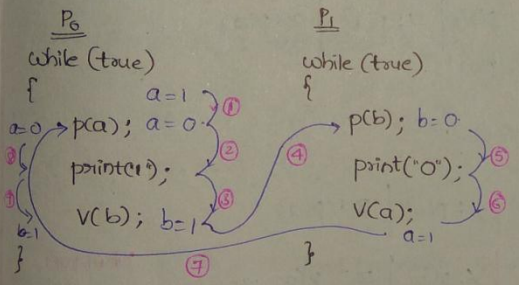
{ ~~P_2~~ P_3 P_4 P_5 P_6 P_7 P_8 P_9 }

∴ 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. ∴ {P_1 - - P_9} can be present in cs at any time.

22. MUTEX EXAMPLE

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

Ans : 10101010- - - - -



101010- - -

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_0	P_1
while (true)	while (true)
{	{
1. p(a);	5. p(a);
2. p(b);	6. p(b);
<cs>	<cs>
3. V(a);	7. V(a);
4. V(b);	8. V(b);
}	}

P_0 : 1 | P_1 : 5 ↓ Blocked | P_0 : 2 | cs | 3 | P_1 : 6 ↓ Blocked | P_0 : 4 |

P_1 : | cs

ME is guarenteed.

24. GATE 2013 QUESTION ON MUTEXES

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.

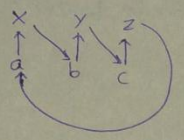
a)

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

X: p(b) → Blocked; Y: p(c) → Blocked; Z: p(c) p(a) → 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)

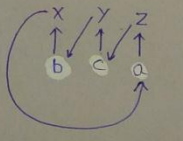
Wait state

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

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) → B | Y: p(b) → B | Z: p(a) p(c) → B — DEADLOCK



= 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

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

Y: p(c) → Blocked | Z: p(a) → B

X: p(c) → Blocked

DEADLOCK

DEADLOCK

25. GATE 2000 QUESTION ON MUTEX

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

```
wait (m[i]); wait(m[(i+1)mod4]);   } This could cause
- - - - Critical section
release (m[i]); release(m[(i+1] mod4));  }
```

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

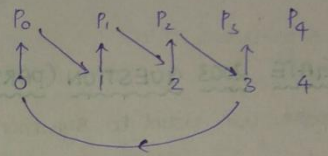
Now, P_0 : P(m[0]) P(m[1]) ⇒ P_0 executes down operation on m[0] and m[1].

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])



$P_0 - P_1 - P_2 - P_3 - P_0$

~~P_0: P(m[1])~~ | P_1: ~~P(m[2])~~ | P_2: ~~P~~m[2] | P_3: ~~P~~(m[0])

∴ Deadlock is possible

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

⇒ P_0: P(m[0]) P(m[1]) ⇒ cs. } P_0 & P_2 are operating on 2 different mutexes
P_2: P(m[2]) P(m[3]) ⇒ cs } so they can enter critical section at same time

(P_1 o 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($m_{(i+1)}$ mod 4);
<cs>
V(m_i) : V($M_{(i+1)}$ mod 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_0)

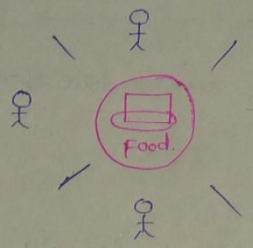
} This sequence results in Deadlock we have seen before
Now to avoid DL let us modify the

Now, after modifying the sequences will be

P_0 : $p(m_0)$ $p(m_1)$
P_1 : $p(m_1)$ $p(m_2)$
P_2 : $p(m_2)$ $p(m_3)$
P_3 : $p(m_0)$ $p(m_3)$.

$P_0 : p(m_0)$ | $P_1 : p(m_1)$ | $P_2 : p(m_2)$ | $P_3 : p(m_0)$ ↓ Blocked

$P_0 : p(m_1)$ ↓ Blocked | $P_1 : p(m_2)$ ↓ Blocked | $P_2 : p(m_3)$ CS release must ex 2/3.



⇒ 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 process '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:
}
```

Synchronisation statements can be inserted only at points W, X, Y, Z

a) which of the following will always lead to an output starting with 00110011001?

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)

S=1, T=0

P: W: P(S) ⇒ (S=0); print(00); X: V(T) ⇒ (T=1)

Q: P(T): ⇒ T=0; print(11); Z: V(S) S=1

∴ Alternate 00110011001- will be printed.

28: GATE 2003 QUESTION (PART 2)

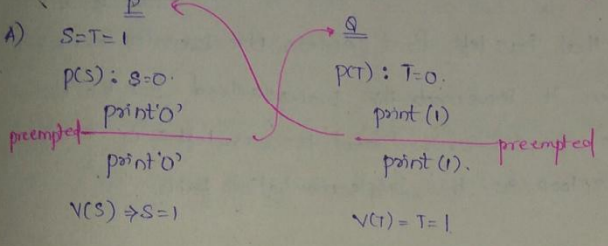
b) which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where 'n' is odd.

A) (P(S) at W) (V(S) at X) (P(T) at Y) (V(T) at Z) (S=T=1)

B) (P(S) at W) (V(T) at X) (P(T) at Y) (V(S) at Z) (S=T=1)

C) (P(S) at W) (V(S) at X) (P(S) at Y) (V(S) at Z) (S=1)

D) (V(S) at W) (V(T) at X) (P(S) at Y) (P(T) at Z) (S=T=1)



010 = 01^n0 ⇒ n=1
: n=odd

∴ 01^n0 substring is possible

similarly B,D also produce the sequence $01^n0/10^n1$ ∴ 'C' is the Ans.

29. GATE 06 QUESTION ON BARRIER (PART 1)

⇒ Barrier is also one kind of synchronisation 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 no's 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?

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 sucession. → 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

⇒ 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 sucessions are given) then it increments the process arrived but the last process leaving the CS makes process_arrived and process_left ≠ 0 eventhough 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 untill 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): Pb(Xb);
      S = S-1;
      if (S<0) {
These cannot be interchanged (deadlock)
          Vb(Xb);
          Pb(Yb);
      }
      else Vb(Xb);
```

```
V(s): Pb(Xb);
      S = S+1;
      if (S<=0) Vb(Yb);
      Vb(Xb);
```


The initial values of x_b and y_b are respectively?

(A) 0 and 0 (B) 0 and 1 (C) 1 and 0 (D) 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. ∴ A|B are eliminated.

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

∴ The value of Y_b should be '0'

∴ Ans = 1,0.

OR

The counting Semaphore has 2 parts 1> value of the semaphore itself

2> list (Queue having blocked process).

S = 4

L

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

∴ Let $X_b = \{0,1\}$. L_1 $\quad$ $Y_b = \{0,1\}$ L_2 → 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 th semaphore 'S' it should down the X_b and ∴ 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 ∴ Blocking occurs when we perform down on '0' ∴ $Y_b = 0$

32. GATE 2010 QUESTION ON MUTEXES

The following program consists of 3 concurrent process and 3 binary semaphores. The semaphores are initialised as $S_0=1$, $S_2=0$. (40)

Process P_0	Process P_1	process P_2
while (true)	wait(S_1)	wait (S_2)
{		
1) wait(S_0)	Release(S_0);	Release (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, $S_0=1$, $S_2=0$ $S_1 <^{0}_{1} \Rightarrow$ $S_0=1$, $S_2=0$, $S_1=1$ } Try with both combinations
$S_0=1$ $S_2=0$ $S_1=0$ }

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

P_0 : 1 2 3 4 | P_1 : 1 2 |
P_0 : 1 2 3 4 | P_2 : 1 2 |
P_0 : 1 2 3 4 | P_0 1 ☺

(000) : ⇒ 3 times

P_0 : 1 2 3 4 | P_1 : 1 2
P_2 : 1 2 | P_0 : 1 2 3 4

☺☺ 2 0's printed.

P_2 : 1 2 | P_1 : 1 2 | P_0 : 1 2 3 4

0 (on'o' printed)

∴ Almost 3 times

33. GATE 2013 QUESTION ON COUNTING SEMAPHORE

int x=0

W: p(s)
x = x+1
V(s)

X: p(s)
x = x+1 ;
V(s)

Y: p(s)
x = x-2;
V(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

Now, there are only two process that are incrementing the value of 'x' : (i+1) and (i+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 ⇒ S_2, S_3 can be executed.

begin : S1; V(a); V(b) end;

begin : P(a); S2 ; V(c); V(d); end;

begin: P(c); S4 ; 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;

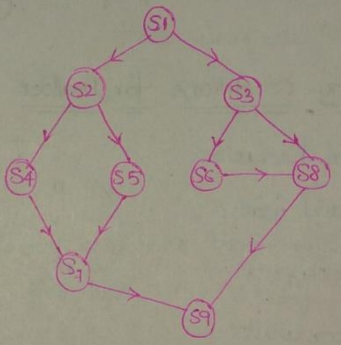
begin : P(g); S6 ; V(i); end;

begin : P(h); P(i); S8 ; V(i); end;

begin : P(j); P(k); S9 end;

coend;

end.



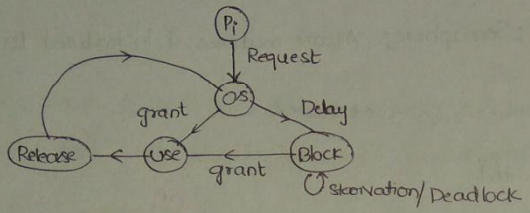
3. DEADLOCKS

1. INTRODUCTION TO DEADLOCKS

→ A set of process 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



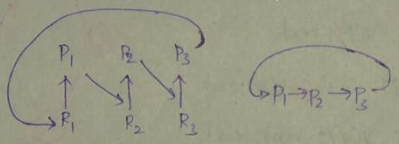
Necessary Conditions for Deadlock

→ Mutual Exclusion

→ Hold and wait:

→ No preemption

→ Circular wait



GATE 1997 QUESTION AND MORE EXAMPLES

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

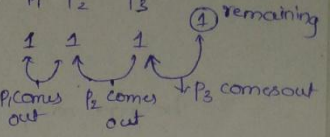
a) 3 b) 5 c) 4 d) 6

⇒ There are 3 process and each require 2 units ⇒ 3×2=6 units, so, if there are 6 units of Resource type 'R' then there wont be any deadlock. (But in Question they have asked for minimum no. of units.

⇒ Now we might think that 2 units is the minimum. ⇒ P_1 uses 2 resources and releases then P_2 takes resource and release 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 I have 3 process say P_1 - 1unit, P_2 - 1unit, P_3 - 1unit and here P_1 waits for P_2, P_2 waits for P_3 and P_3 waits for P_1 ⇒ DEADLOCK

⇒ Now, if I have 4 units it will solve my problem



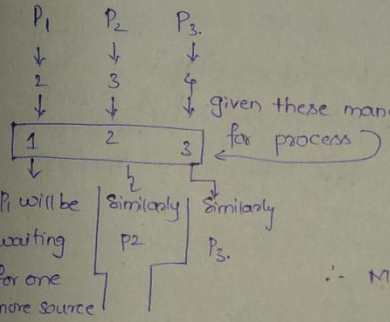
⇒ The max no of Resource (units) that could cause deadlock = 3.

⇒ The min no of units req'd so that No deadlock will occur = 4 ⇒ 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
2	3	4	Resources

} then how many resources are req'd in minimum so that there wont 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

Let us say P_1 P_2 P_3 --- P_n

x_1 x_2 x_3 ---- x_n

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

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

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

∴ 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 P_i = 2 instances what is the max no of pro

Given R=6 instances P_i = 2 instances.

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 (6-1)

↳ Total instances.

R=6, P_i = 3. (Each require 3 instances so give 2 instances for them and make them wait for one more instance)

↳ No. of instances needed by the process (each process).

P_1 P_2 P_3. ∴ Then three process results in DL

2 2 2. ∴ No of process that doesnt cause DL = 2.

R=100, P_i = 2 instances. then

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

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

Max. no. of process reqd such that there wont be DL = 100 - 1 = 99

R=100, P_i = 3 instances

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

Max no of process reqd suchthat there wont be DL = 49

R = 100, P_i = 4 instances.

$n \times (3) = 100$

$\Rightarrow n = \lceil \frac{100}{3} \rceil$

$\boxed{n = 34}$ process

Min no of process lead to DL = 34

Max no of process donotlead to DL = 33.

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

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

Now Give 2 instances to each process.

$\Rightarrow$

P_1	P_2	P_4
2	2	2
↓	↓	↓
wait for 1 instance	=	=

∴ Min no of process that can cause deadlock = 3

Max no of process that results in DL free = 2.

4. GATE 1993 QUESTION ON MINIMUM RSOURCES 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?

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

m = ?

A	B	C
3	4	6

Max 'm' where there will be deadlock = 2+3+5 = 10

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

5. GATE 2005 QUESTION

Suppose 'n' processes $P_1 -- P_n$ 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 doesnot 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 $P_1 P_2 -- P_n$

$P_1 - S_1$ units of Resource

$P_2 - S_2$

$P_n - S_n$ units of Resource

P_1	P_2	P_3	---	P_n
$S_1 - 1$	$S_2 - 1$	$S_3 - 1$		$S_n - 1$

$= \left(\sum_{i=1}^{n} S_i - n\right)$ ∴ $m \leq \sum_{i=1}^{n} S_i - n$ ⇒ $\sum_{i=1}^{n} S_i < (m+n)$

For deadlock to occur

For DL not to occur.

6. GATE 06 QUESTION ABOUT NECESSARY CONDITION FOR DEADLOCK (46)

Consider the following snapshot of a system running 'n' process. process 'i' is holding 'x_i' instances of a Resource 'R' for $1 \le i \le n$. 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 guarentee that the system is not approaching a deadlock?

a) $\min(x_p, x_q) < \max_{k \neq p,q} y_k$ (b) $x_p + x_q \geq \min_{k \neq p,q} y_k$

c) $\max(x_p, x_q) > 1$

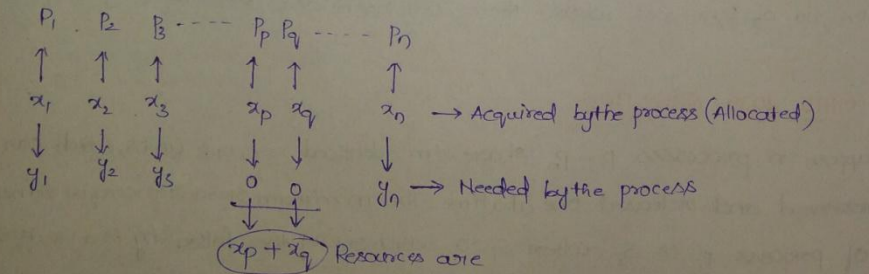
d) $\min(x_p, x_q) > 1$

R=9

P_1	P_2	P_3	
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 Remaing process need (atleast it should satisfy the min need ~~of the~~ among the Remaining process.

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

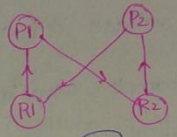
7. DEADLOCK HANDLING MECHANISMS

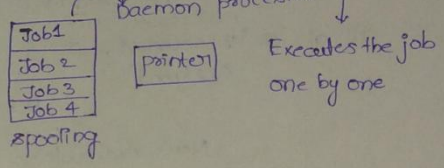
Strategies for Handling Deadlock

1) Deadlock Ignorance – Both windows and Linux uses Deadlock Ignorance
2) Deadlock prievention – 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 $R_3, R_4 \cdots R_n (n>2)$ but not R_1) (Can be implemented practically).

9. SAFE, UNSAFE, DEADLOCK AVOIDANCE AND BANKERS ALGORITHM

At any instant of time the snap shot of the system is

E=(6 3 4 2)
P=(5 3 2 2)
~~A=(1 0 1 0)~~
A=(1 0 2 0)

	Tape drivers	plotters	scanners	CDRom's
A	3	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

Process	Tape drives	plotters	scanners	CD-Roms
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 - It is a vector that represents the total no of resources available in system

E (6 3 4 2)
- 6 → Tape drives
- 3 → plotters
- 4 → scanners
- 2 → CD Roms.

P and (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

→ whenever 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" ⇒

A =	1	0	2	0	
D =	1	1	0	1	'D' satisfied
New Available =	2	1	2	1	

⇒ with the New Available A can be satisfied

⇒ New available =

	2	1	2	1
	3	0	1	1
	5	1	3	2

'A' satisfied

⇒ with New Available B can be satisfied

A =	5	1	3	2	'B' satisfied
B =	0	1	0	0	
	5	2	3	2	

⇒ with New Available A = 5 2 3 2

'C' can be satisfied

A =	5	2	3	2	'C' satisfied
C =	1	1	1	0	
	6	3	4	2	

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

∴ DABCE = safe state

⇒ The above phase is called Deadlock Avoidance and the Algo is called Bankers Algorithm.

10. GATE 2007 QUESTION ON SAFE STATE

Three resource types X, Y, Z

There are 5 units of each resource type

consider the following table and say which process finishes last.

	Alloc			Request		
	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

(A) P_0

(B) P_1

(C) P_2

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

	X	Y	Z
Total =	5	5	5
Allocated:	5	4	3
Available	0	1	2

Now, the Available satisfies the need of P_1 ⇒ after execution P_1 releases the resources

present available = (0 1 2)

P_1 Resource are Released = (2 0 1)

(2 1 3) = New Available

Now, (2 1 3) will satisfy the need of P_0 (1 0 3)

∴ (2 1 3)

(1 2 1)

(3 3 4) = New available

Now, New available satisfies the need of P_2 ⇒ (3 3 4) (0 1 2)

∴ New available = (3 3 4)

(2 0 1)

Total = 5 5 5

$P_1 P_0 P_2$ is the safe sequence

11. 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) ⇒ The future need matrix will be

⇒ 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)

∴ Available = (0 0 1 1 1)

P_D Resources = (1 1 1 1 0)

(1 1 2 2 1) = New Available

Now, New Available (1 1 2 2 1) doesnt satisfy any of the process need.

∴ Deadlock for x=1

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)

P_D (1 1 1 1 0)

New Avail = (1 1 3 2 1) → This solves/ leaves the need of all other processes ∴ for x=2 safe state exists

∴ Min value of x=2 ∴ No deadlock

12. GATE 2014 QUESTION ON BANKERS ALGORITHM

	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.		

Available (3,2,2)

Req 1: P_0 (0,0,2)

Req 2: P_1 (2,0,0).

Now given the system is in safe state then will the Req 1, Req 2 be granted?

Req 1: P_0 (0 0 2)

Available - (3 2 2)

Now, P_0 needs (002) Now assume i allocated it then, Allocation, Need matrices look as, when i allocate it the allocated Resources gets incraased and Need of the process gets deraased

Allocation

	X	Y	Z
P_0	0	0	3
P_1	3	2	0
P_2	2	1	1

Need

	X	Y	Z
P_0	8	4	0
P_1	3	0	0
P_2	1	2	2

Available will become (3,2,0).

⇒ satis fy P_1

⇒ 320 → New Available

320 → P_1 Release resources.

640 → New Available

Here the need of P_1 is satisfied, P_0 / P_2 doesnot get satisfied so the Req 1 will not be granted.

Similarly perform with P_2 then you will get a safe sequence

13. GATE 1996 ON BANKERS ALOGORITHM

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 P_0 : (0 1 0)

whid will the Req be granted

Now, assume i allocated the resource then Allocated, Future Need, Available are going to change.

Allocated

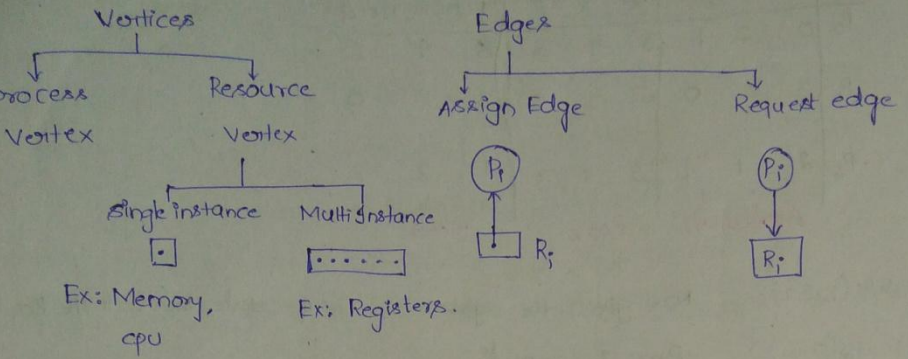
P_0 : (1 1 2)

Need (Future)

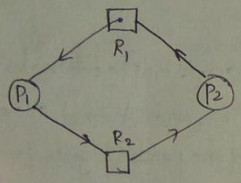
P_0 : (3 0 0)

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

14. RESOURCE ALLOCATION GRAPH

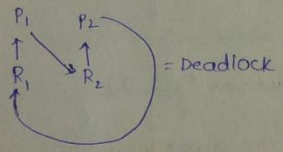


15. RESOURCE ALLOCATION GRAPH EXAMPLES

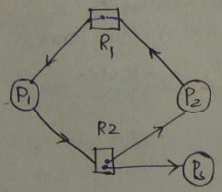


⇒ In case of single instance types if there is a cycle then there will be deadlock

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



= Deadlock



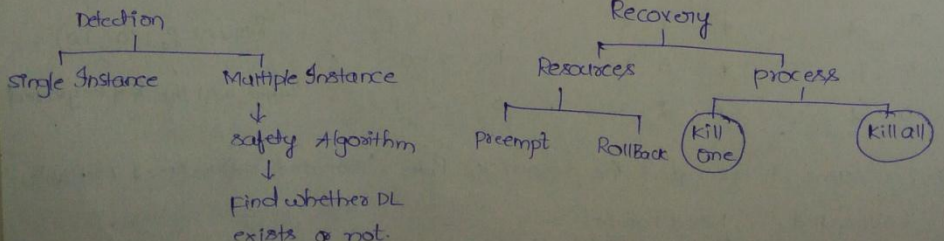
	Alloc		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Available = (0 0)

NO deadlock, safe sequence exits.

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

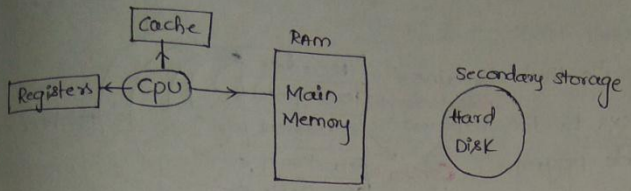
16. DEADLOCK DETECTION AND RECOVERY



4. MEMORY MANAGEMENT

1. NEED FOR MULTIPROGRAMMING AND MEMORY MANAGEMENT

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



1) 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.

MM = 4mB. Process size = 4mB

cpu utilisation = $(1-p)$ = 20%. (80% 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 pbty that all the process do I/o at same time?

⇒ MM = 16mB, ps = 4mB } = 4 process ∴ Now pbty that all process do I/o at same time = $p \times p \times p \times p = p^4$

∴ cpu utilisation = $1-p^4$ ≅ 60% (assuming 80% of time spends in I/o)

⇒ MM = 32mB ps = 4mB ⇒ 8 process

cpu utilisation = $1-p^8$ ≅ 83%

∴ cpu utilisation = $1-(p)^n$

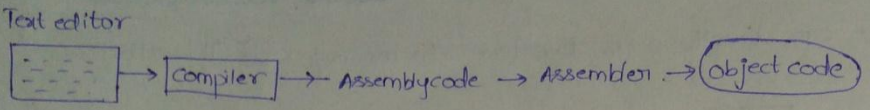
$= 1 - (\text{fraction of time spent in I/o})^n$

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

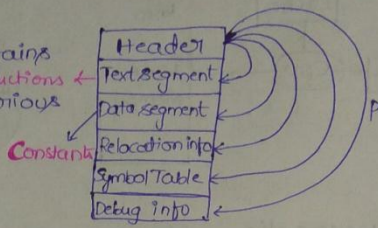
∴ cpu utilisation = $1-p^n$ (n = no. of process in MM).

2. OBJECT CODE, RELOCATION AND LINKER

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

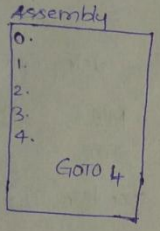


Now, the object code is a file that contains The Header contains pointers to the various segments of the object code program.



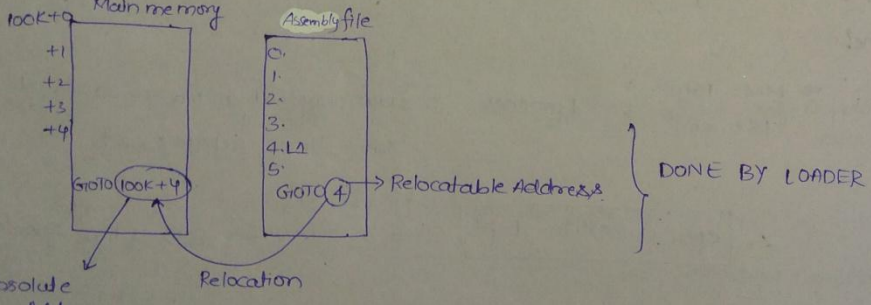
Now this object code file contains "Relocation info" Now the compiler/assembler assume that the line no'0' starts with Address '0'.

Now,



This works fine when the address starts from '0' but there is no guarentee that the file (object file) will be loaded ~~from~~ into 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 → obsolute Address).

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

main()	100 ← Address
printf()	—
scanf	—

} unresolved Symbols (their address cannot be found)

This contain
Relocatable
Addresses bec
linker doesn't
know where the
program is loa

phase (like
contain th
so we wo

⇒ The mai

3. LOADER

Object linked

1) program loa
2) Relocation
3) Symbol Reso

Linker:

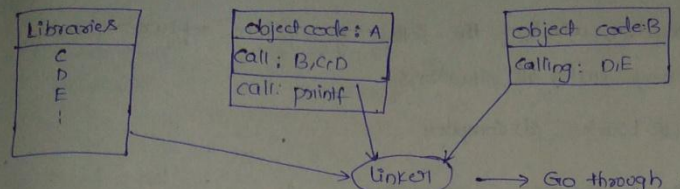
Symbols Resoluti

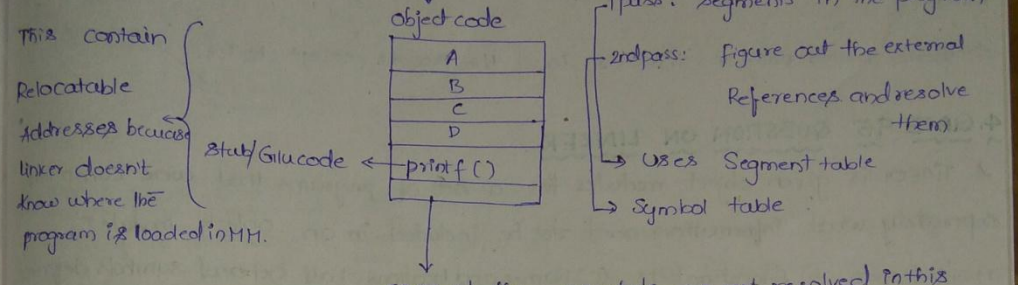
loader

Program loading

⇒ Loading is no
something and
Main memory.

⇒ Loading time





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 Glue code) 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 ∝ process size

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

1. Compile time: whatever you do during the compilation is called.
   Symbolic names → 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 → Absolute Address
4. Run time: Dynamic linking happens
   → 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 loder.

4. GATE 95 QUESTION ON LINKER

A linker is given object modules for a set of programs that were compiled seperately. 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 definetely.

b) ~~Absolute~~ 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 is called —

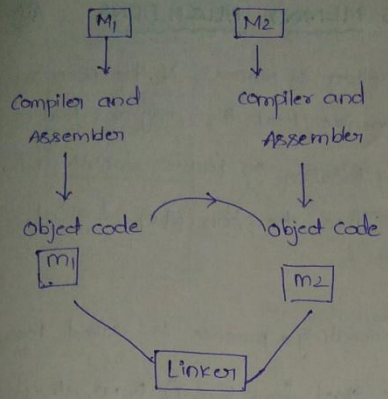
a) Assembly b) parsing c) Relocation d) Symbol Resolution

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 pagefault Rate in the system.

c) Faster program startup.

d) Existing programs need not be re-linked to take advantage of newer versions of libraries.

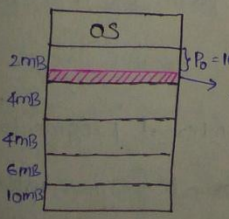
a) → Yes, when we have dynamic sharing we are not going to have object code of the library function in our executable code but we just pass the reference to where the code is present.

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

⇒ $P_0 = 1MB$ ⇒ smallest partition that I could assign = 2MB

If $P_0 = 1MB$ → Internal Fragmentation (cannot be assigned to any of the process)

⇒ If $P_2 = 11MB$ (No partition could be assigned)

= External Fragmentation

⇒ Degree of Multiprogramming is fixed / limited

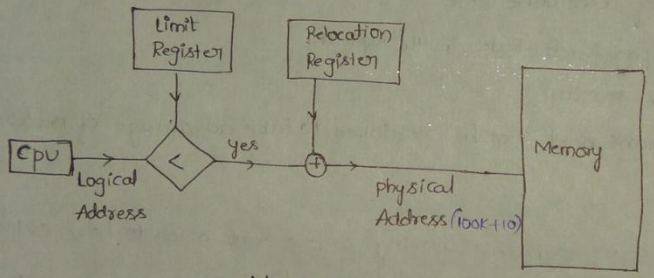
11. RELOCATION AND PROTECTION IN CONTIGUOUS MEMORY ALLOCATIONS

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

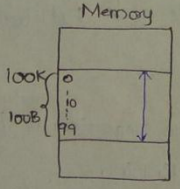
⇒ The Addresses generated by the cpu while generating a process is called Logical Addresses. (Relocatable Address (assume address start from Zero)). Now, the logical Address should be converted to physical address

⇒ Involving loader to convert Relocatable Address to obsolute 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

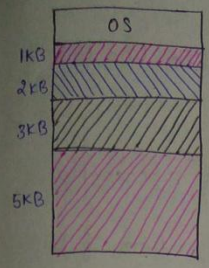
⇒ Dynamic partitioning / variable partitioning.

⇒ 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 wont be any internal Fragmentation.

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

⇒ If there is no internal fragmentation then there may or maynot be external fragmentation



⇒ 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 freed is not contigrous 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.

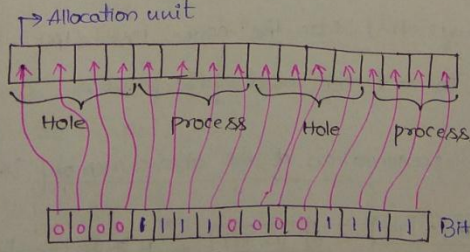
⇒ The Allocation and deallocation of memory is complex.

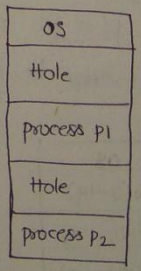
13. BITMAP FOR DYNAMIC PROGRAMMING

⇒ The Data structures that are used to keep track of free and occupied spaces
⇒ The Data structure is Bitmap and Linked lists.

Bitmap:

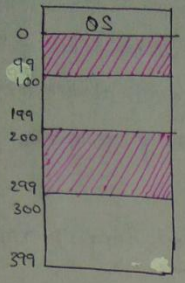
⇒ The memory is divided into small parts called "Allocation units".
⇒ For every Allocation unit the single bit 0 or 1 is assigned (0 represents the allocation unit is open/free and 1 represents the allocation Bit unit is occupied.



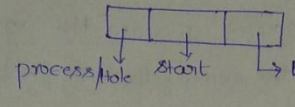


⇒ Now, suppose the size of allocation unit is 4 Bytes = 32 Bits Now 1 bit in the Bitmap is used to represent whether these 4B (Allocation unit) is occupied or not. so the Bitmap is going to take $\frac{1}{(32+1)} = \frac{1}{33}$rd part of memory is taken by Bitmap.
⇒ Bitmap is not widely used /Not used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

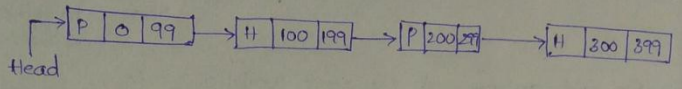
14. LINKED LIST FOR DYNAMIC PARTITIONING



⇒ 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 freeging 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

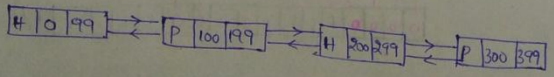
1) First fit
2) Next Fix
3) Best fit
4) worst fit

First Fit

⇒ The first fit algorithm says that scan the entire linked list and find the hole that is bigenough 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 50bytes, 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.

Now, after a

Next Fit

The Next Fit

will start

⇒ Next fit is

Best Fit

⇒ Best fit Algo

which can

⇒ The disadvan

it will

Worst fit

⇒ This says

assign the

Quick Fit

⇒ This says th

and maintain

⇒ We might hav

⇒ The mallo()

16. GATE 2004 Qu

Consider the heap

regions are in

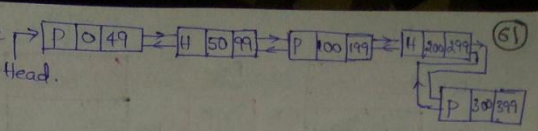
The sequence of

if we use

a) either first fit/

b) First fit but no

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 accomodate 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

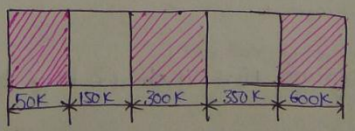
⇒ This says that there are fixed sizes for the most frequently used process and maintain a linked list for them.

⇒ We might have to maintain seperate 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

a) either first fit / Bestfit policy (any one)

b) First fit but not bestfit policy

c) Bestfit but not Firstfit policy (d) None

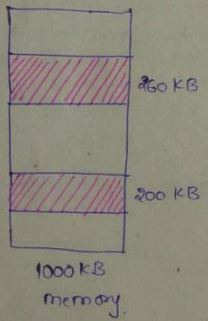
(50) cannot be assigned.

∴ BF fails.

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

In a compa

jobs to me

19. Summary

Equal

10
10
10
10
10

FF ↑

Ef ↑

18. GATE 98 QUESTION ON FIXED PARTITIONING

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

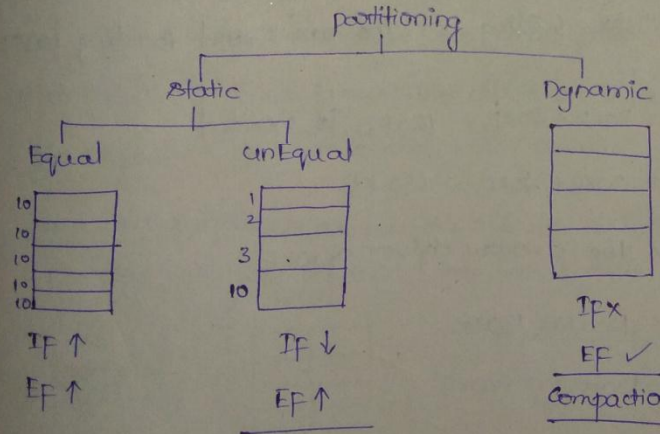
Partition sizes in KB	4K	8K		20K		2K
Jobs	2K	14K	3K	6K	10K	20K	2K
Time for execution	4	10	2	1	1	8	6

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



Compaction, Non contiguous allocation → paging, Segmentation

overlays ⇒ overlays says that when a program is running then all the entire program need not be needed at a given time, only a part of the program is required at a instant so load that part and free the memory when you come/take/load the other part

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 overlays 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

=120KB.

⇒ But at any time only one [redacted] 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: 70KB + 30KB + 20KB = 120KB is needed

In pass2: 80KB + 30KB + 20KB = 130 KB

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

∴ pass1 = 120+10 = 130KB.

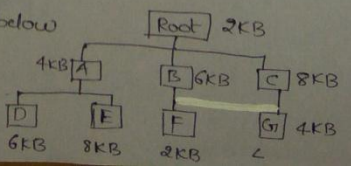
pass2: 130+10 = 140KB

Now, we need in total a partition of size ↓ [redacted]

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



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

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

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

Similarly. (R+A+E) = 14KB

(R+B+F) = 10KB

(R+C+G) = 14KB

∴ The size of partition = max {12, 14, 10, 14}

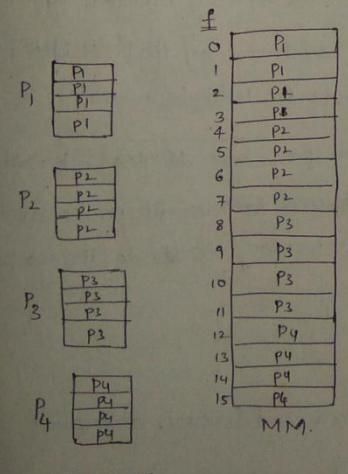
Min Size of partition = 14

22. NEED FOR PAGING

⇒ The main problem that even the dynamic partitioning could not eliminate is External Fragmentation.

⇒ 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).

⇒ Now, lets 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.



⇒ Now say P_2 and P_4 event for I/O (they are removed from MM and stored on hard disk)

⇒ 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.

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

⇒ The above logical Address should be converted into physical Address by Memory Manager into <frame

23. PAGING EXPLAINED WITH EXAMPLE

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

⇒ frame size = 4B

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

⇒ process size = 16B, page size = 4B

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

⇒ Now, the structure of the process will be

P_0

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

No. of frames = 4
process size = 0-15 = 16B.

⇒ Now, the structure of Main memory will be ⇒

and say the process is loaded into MM starting from frame 2 (say frame 0, frame 1) are occupied.

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				

P_0, P_0, P_0, P_0 } loaded in Main Memory

MM = 64B ⇒ divided in frames of size 4B.

⇒ 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).

∴ 6 = | 0 | 1 | 1 | 0 |

{ Now, Each page has 4 Bytes in it ⇒ 2 bits are sufficient to identify particular byte in the memory so divide it into 2 parts

| 0 | 1 | 1 | 0 |

page no | page offset

⇓

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.

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

0	f2
1	f3
2	f4
3	f5

<pageno> <Frame number>

page no = $(01)_2 = (1)$ ⇒ page 1 corresponds to frame 3.

page offset = Frame offset = $(10)_2$ = 2

∴ Go to frame 3 and fetch the Byte at 2nd position (0th, 1st, 2nd)

= <f_3, 10>

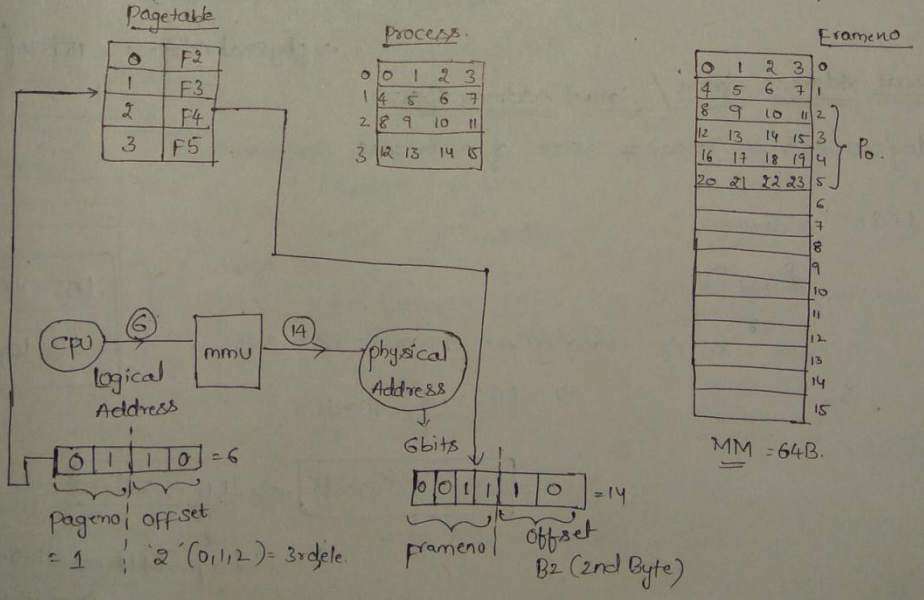
⇒ The physical address is limited by size of MM (6bits in this case) = 14 Byte no.

⇒ 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 ⇓ 0011 = 3.

offset ⇓ (2) (same as offset in logical address)



24. BASICS OF BINARY ADDRESS

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

25. PHYSICAL ADDRESS SPACE AND LOGICAL ADDRESS SPACE

⇒ The smallest addressable unit in the Computer is word

Physical Address Space

⇒ physical Address Space is nothing but size of main memory

⇒ Now, PAS = 128 KB

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

PAS = 2^{17} Bytes ⇒ Now given 1 word = 4B. = 2^2 B.

⇒ PAS = 2^{17} Bytes

= $\frac{2^{17}}{2^2}$ words ⇒ PAS = 2^{15} words

⇒ physical Address = 15 bits ($\log_2(2^{15})$)

PAS = 'm' words

PA = $\log_2(m)$ bits

Logical Address Space / Virtual Address Space

⇒ logical Address Space = size of a process.

LAS = 256 mB

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

= 2^{28} B ⇒ Now given 1 word = 4B = 2^2 B

⇒ ⇒ LAS = $\frac{2^{28}}{2^2}$ words

LAS = 2^{26} words ⇒ LA = 26 bits

LAS = 'm' words

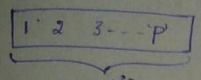
LA = $\log_2(m)$ bits

Logical Address = No. of bits required to address

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

⇒ Virtual memory was introduced to increase the degree of multiprogramming.

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



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

$\Rightarrow$ page size = 4KB.

word size = 4B.

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

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

26. 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)

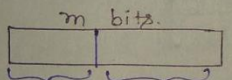
page offset = $\lceil \log_2 P \rceil$ = 'p' bits (say).

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

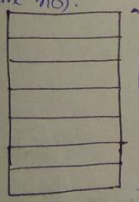
PAS = 1MB = $2^{20}B$. = 20 bits

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

PA =



(frame no).

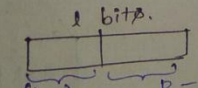


Frame = 2^p words

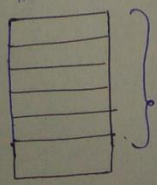
PAS = 2^m words

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

LA =



page no



page size = 2^p words.

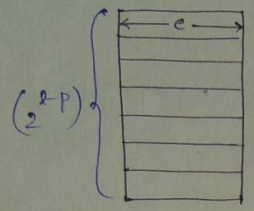
LAS = 2^l words

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

Now, PA = | (m-p) | (p) | — m bits.
frame no | frame offset

LA = | (l-p) | (p) | — l bits.
page no | page offset

page table



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

⇒ Let 'e' be the size of each entry

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

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

Example

LAS = 128MB = 2^{27} B
PAS = 1MB = 2^{20} B
PS = 4KB = 2^{12} B

LA = 27 bits
PA = 20 bits
page off = 12 bits.

} page table size = ?

Given, LA = 27 bits → | PNO | page offset | = | 15 | 12 |

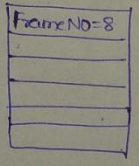
PA = 20 bits → | PNO. | frame offset | = | 8 | 12 |

p.offset = 12 bits

∴ No. of pages = 2^{15}

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

PTS ⇒ 2^{15} Bytes = 32 KB



2^{15} pages

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

PTS = 32 KB.
PS = 4 KB.

PTS >> ps (Multilevel paging).

pages
cess = 2^1-p

thing is
ntioned
out the
ze of each
y so take
s frame
size

27. NUMERICAL ON PAGING

LAS	PAS	LA	PA	Page size	Page offset	pages	Frames	PTE	PTS.
128KB	128KB	17bits	17bits.	4KB	12bits.	32	32	5bits.	$\left(\frac{2^5\times5}{8}\right)$B
256KB	1MB	18bits	20bits.	4KB	12bits	64	2^8	2B	$(2^6\times2)$Bytes
1MB	2^{18}B.	20bits.	18bits	2^{10}B	10bits.	2^{10}.	256	8bits.	2^{13}(bits)
2^{20}B	512MB	20bits.	29bits.	2^{12}B.	12bits.	256	2^{17}	17bits.	(256×b) bits
2^{22}B.	2^{21}B.	22bits	21bits	2^{12}B.	12bits	2^{10}	2^9	9bits	
2^{22}B.	2^{22}B.	22bits	22bits.	2^{14}B	14bits	2^8	2^8	8bits	256B

1> LAS = 128KB ⇒ LA = $\lceil \log_2 128K \rceil$ = 17 bits = 128KB = $2^7 \times 2^{10}$B = 2^{17} Bytes = 17bits.

PAS = 128KB ⇒ PA = $\lceil \log_2 128K \rceil$ = 17 bits.

page size = 4KB ⇒ page offset = 4×2^{10}B = 2^{12}B ⇒ page offset = 12bits.

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

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

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

PTS = (No. of pages) × (PTE size) bits = $2^5 \times 5$ bits = $\frac{2^5\times5}{8}$ Bytes. ≅ 20B

2>. LAS = 1MB = 1×2^{20}B. = 2^{20}B ⇒ LA = $\lceil \log 2^{20} \rceil$ = 20bits

page offset = 10bits ⇒ Now the page size = 2^{10}.

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

⇒ Now, No. of frames = $\frac{PAS.}{Frame\,size} = \frac{PAS}{page\,size}$ ⇒ PAS = page size × No. of frames

⇒ PAS = $2^{10} \times 256$

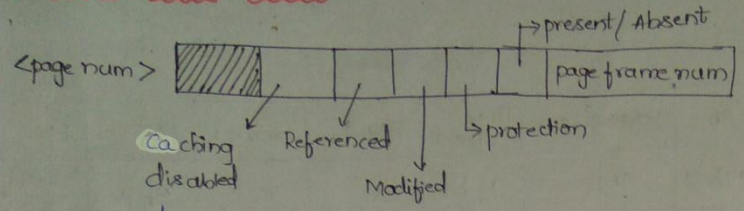
= $2^{10}\times2^8 = 2^{18}$.

⇒ PA = 18 bits.

⇒ PTE = 8bits ⇒ PTS = No. of pages × PTE

= 2^{13} 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 (Red, 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 PAGE TABLE ENTRY

In Virtual memory system, size of virtual address is 32bit, size of physical Address is 30bits, page size is 4KB and size of each page table entry = 32bit, The MM is Byte addressable. which of the following is ~~main~~ 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.

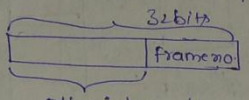
PA = 30 bits ⇒ PAS = 2^{30} B.
LA = 32 bits ⇒ LAS = 2^{32} B.
{ Size of process > Size of Main Memory }.

Now, page Size = Frame Size = 4KB. = 2^{12} B.

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

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

Now,



⇒ other info = 32 − 18

Other info = 14 bits.

30. NEED FOR MULTILEVEL PAGING:

LA = 22 bits.

LAS = 2^{22} Bytes.

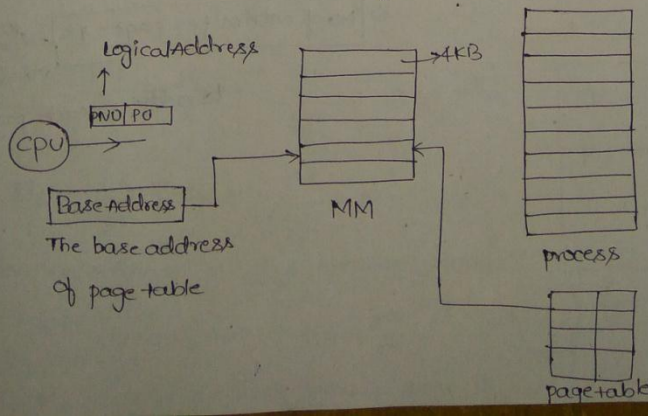
PS = 4KB = 2^{12}B ⇒ page offset = 12 bits ⇒ No of pages = $\frac{LAS}{PS} = \frac{2^{22}\,B}{2^{12}\,B} = 2^{10}$

∴ **No of pages = 1K**

PTE = 4B

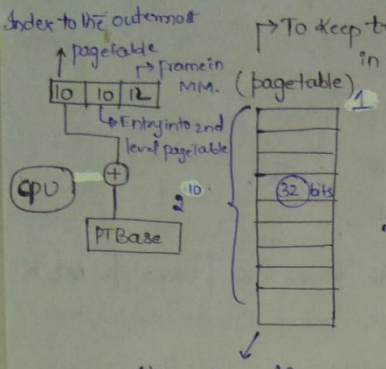
PTS = (No of pages) * PTE = 1K * 4B = 4KB.

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



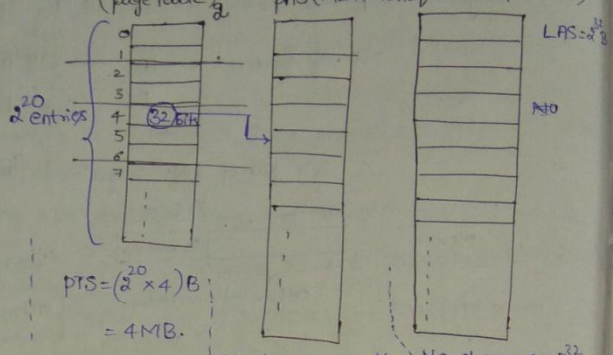
If page table size > the page Size then we go for multilevel paging.

31. TWO-LEVEL PAGING EXAMPLE



↳ To keep track of pages of PT_2 are loaded in which frames of MM).

PS = 4KB = FS.



Here PTS = $2^{20} \times 4$ B

= 4KB.

PS = 4KB

PTS = PS

⇒ NO. of entries = 2^{10}

⇒ To identify a particular entry we need 10 bits.

⇒ This pagetable should always be in main memory.

⇒ The PT Base Register stores the Address of the frame that contains the (pagetable)$_1$

PTS = $(2^{20} \times 4)$B

= 4MB.

PS = 4KB

No. of frames = $\frac{2^{44}}{2^{12}} = 2^{32}$ frames

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

Now PTS $\gg$ PS = FS ⇒ Hence the page table cannot fit in one frame of the MM so divide the page table into no of pages

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

⇒ Now the page table is divided into pages and loaded in main memory. Now we need to keep track of which pages of the pagetable are loaded in which frame of MM, so we need another page table

⇒ Now, here assume each entry size = 4B.

Each page size = 4KB

⇒ No. of entries per page = 1K = 2^{10} = 10 bits are needed to address each page of pagetable

32. EXA

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

1> VA = 4

PS = 16

∴ PT_1 =

2> VA = 6

PS = 1

PTE =

No. of P

PT_2 =

PT_3 =

1> Index the

2> PTE = 4B.

PS = 1MB

3> PTE = 4B,

32. EXAMPLES ON MULTILEVEL PAGING

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

1) VA = 48 bits. ⇒ VAS = 2^{48} B

PS = 16KB = 2^{14}B ⇒ No. of pages = $\frac{VAS}{PS} = \frac{2^{48}}{2^{14}} = 2^{34}$ pages. ∴ (PT1 size) = $2^{34} \times 2^2$ = 2^{36}B.

∴ PT1 = 2^{36}B, Now, PS = 16KB

⇒ $PT_2 = \frac{PT_1}{PS} = \frac{2^{36}}{2^4 \times 2^{10}} = 2^{22}$ B × PTE = $2^{22} \times 2^2$B = 2^{24}

(2^{22} = No. of pages in PT_2)

⇒ $PT_2 = \left(\frac{PT_2}{PS} \times PTE\right) = \frac{2^{24}}{16 \times 2^{10}} \times 4B = 2^{12}$B.

No. of PT_1 = (pages)(x)(size of each entry in PT_1).

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

2) VA = 64bits ⇒ VAS = 2^{64}B

PS = 1MB ⇒ 2^{20}B.

PTE = 4B.

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

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

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

PT^3 entries = 2^8 (No. of pages)

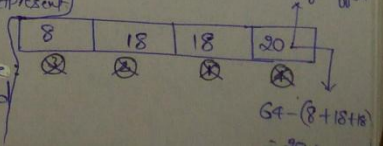
PT_2 entries = 2^{26}

PT_1 entries = 2^{44}

1) Index the outermost PT = PT_3 = 8 bits are needed (2^8 pages are present)

2) PTE = 4B. PS = 1MB. ⇒ No. of entries per page = $\frac{2^{20}}{2^2} = 2^{18}$ = 18 bits are needed

3) PTE = 4B. PS = 1MB ⇒ No. of entries per page = 18.



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

1> VAS = 4mB
PS = 4KB.
PTE = 4B.

VAS = 4mB = 2^{22}B.
PS = 4KB = 2^{12}B.
PTE = 4B.

No.of pages = $\frac{VAS}{PS} = \frac{4mB}{4KB} = 2^{10}$B = 1K pages

Now, PTS = page table pages * e
PTS = 1K * e

Now, PTS ≤ PS (To fit PTS in one page)
1K * e ≤ PS
e ≤ $\frac{2^{12}}{2^{10}}$.
e ≤ 4B.

2> VAS = 4GB. = $4 \times 2^{30} = 2^{32}$B.
PS = 128KB. = 2^{17}B.

No.of pages = $\frac{VAS}{PS} = 2^{15}$ pages.
= 32K pages.

Now, PTS = No.of pages * e
PTS. = 32K * e
⇒ PTS ≤ PS
⇒ $2^{15} * e \leq 2^{17}$
e ≤ 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

PAS = 64mB. = 2^{26}B
VA = 32 bits ⇒ VAS = 2^{32}B
PS = FS = 4KB. = 2^{12}B

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

e = frame no = (26-12)

⇒ PTS = No.of pages * e
= No.of pages * Frame size num
= 2^{20} * Frame num
= $(2^{20} * 14)$ bits = 14M bits
= $\frac{14m}{8}$ Bytes = 2mB

(2) Which
a) Faster
b) proce
c) Linker
in th
d) progra

37. GATE
PA = 36 bi
as follow
→ Bits : 3
→ Bits
→ Bits 12
→ "
Then PTE
A) 20,20,20

PA = 36 bits
PS = 4KB.

⇒ when we
in all the

38. GATE 20
Theory Ques

41. GATE 20
VA = 46 bits
PTE = 32 bi
Let size of p

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

a) faster access to memory on an average (All pages are in hard disk so slower)

b) process can be given protected address spaces (Not specific to VM)

c) linker can assign address independent of where the program will be loaded in the physical memory. (Relocation, this exists in contiguous Allocation also)

d) programs larger than the physical memory size can be run.

37. 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 " "

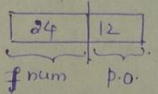
→ " 0-11 are used as offsets within a page

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 bits

PS = 4 KB. ⇒ page offset = 12 bits



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

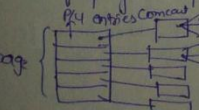
38. GATE 2009 QUESTION ON MULTI LEVEL PAGING

Theory question easily answerable

41. GATE 2013 QUESTION ON MULTILEVEL PAGING

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?

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

No. of entries = $(P/4)^2$ = $(P/4)^3$ ∴ VAS = $(P/4)^3 * P = 2^{46}$

P = 8KB.

42. FINDING OPTIMAL PAGE SIZE

VAS = 4GB.

PS = 4KB

⇒ No. of pages = $\frac{VAS}{PS}$

= 1m pages

VAS = 4GB

PS = 4MB.

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

= $\frac{4GB}{4MB}$ = 1K pages

Therefore PS↑ PTS↓

page size = p Bytes

PTE = 'e' bytes

on Average VAS = 's' Bytes

overheads = $(p/2) + (s/p) * e$ → wasted in page Table

↓

The last page might not be completely filled

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

⇒ $1/2 - s/p^2 * e = 0$

⇒ $p = \sqrt{2se}$ | s: Avg. process size, e = PTE

↓

optimal page size

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

1) VAS = 4KB = 2^{12} B

PTE = e = 8B.

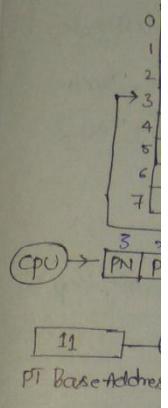
⇒ $p = \sqrt{2 \times 2^{12} \times 2^3} = 2^8 = 256B.$

⇒ 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

overhead = $\left[\frac{np}{2} + s(1/p) * e\right]$

n = no. of pages the section is divided into.

43. VM



⇒ The VM

⇒ VM is

⇒ The process

MM, In

the page

is design

P2 and

44. TLB

Now, If VAS=

⇒ N

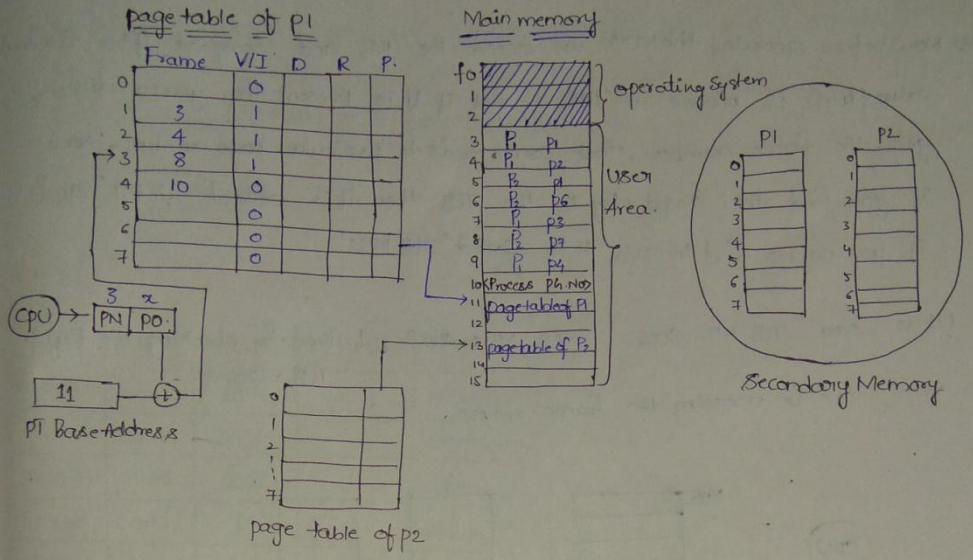
further divided

the disad van

to access a fram

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 are loaded in the MM, So the MM <P_1, P_1> represent <process no, page no>, Now to keep track of the pages of program are loaded in which frame number is the page table is designed, and when there is context switch the control switches from p1 to p2 and the value in "PT Base Address" will now be 13

44. TLB

Now, If VAS = 4GB, PS = 4B

⇒ No. of pages = 1 million.

} Now, the size of the page table 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)

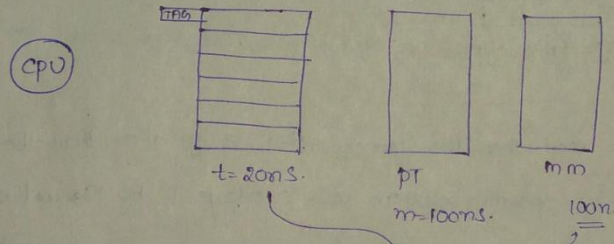
⇒ The CACHE is faster than MM and cheaper than Registers.

⇒ The TLB is also sometimes called ASSOCIATIVE MEMORY ↑ 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.



Effective Access time = $\frac{80}{100}(20ns + 100ns) + \frac{20}{100}(20ns + 100ns + 100ns)$

EAT = 120ns.

If TLB HIT = p ⇒ TLB MISS = 1-p

EAT = $p(t+m) + (1-p)(t+m+m)$

EAT = $p(t+m) + (1-p)(t+2m)$ (Single level page Table)

If there are k levels then EAT = $p(t+m) + (1-p)(t+km+m)$

45. GATE

A paging

takes 50

A) 54

⇒ EAT =

EAT

46. NUM

TLBA	m
20ns	1
20ns	1
20ns	1
20ns	1
20ns	1
20ns	

1> EAT =

2> EAT =

3> EMA

→ 2

10

⇒

45. GATE 2008 QUESTION ON TLB

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.

$\Rightarrow$ EAT = $\frac{90}{100}(50+10) + \frac{10}{100}(10+50+50)$

EAT = 65ns

46. NUMERICALS ON TLB

TLBA	MA	TLB H	PT levels	EMAT
20ns	100ns	80%	1	140ns
20ns	100 ns	80%	2	160ns.
20ns	100 ns	80%	3	180ns.
20 ns	100ns	90%	1	130 ns
20ns	100ns	60%	1	160ns
20ns	100ns.	50%	1	170ns.

1) EAT = $\frac{80}{100}(20+100) + \frac{20}{100}(20+100+100)$

= 96 + 24 = 140ns.

2) EAT = $\frac{80}{100}(20+100) + \frac{20}{100}(20+2(100)+100)$

= $\frac{80}{100} \times 120 + \frac{20}{100} \times (320) = 96+64 = 160$ns.

3) EMAT = 130ns.

$\Rightarrow \frac{x}{100}(20+100) + \left(100 - \frac{x}{100}\right)(20+100+100) = 130$

$\Rightarrow \frac{x}{100}(120) + \frac{10^4 - x}{100}(220) = 130$

$= \frac{6x}{5} + \frac{(10^4 - x)11}{5} = 130$

$x = 90$ nos%

$p(20+100) + (1-p)(20+100+100) = 130$

$= 160 = 0.6(t+100) + (0.4)(t+100+100)$

$170 = 0.5(20+m) + 0.5(20+2m)$

$p(20+100) + (1-p)(220) = 130$

$\Rightarrow 120p + 220 - 220p = 130$

$= -100p = -120 + 90 \Rightarrow p = 90/100 = 90\%$.

47. TLB SUMMARY

TLB access time = t

Main memory access = m

TLB miss rate = p ⇒ TLB Hit rate = (1-p)

EMAT = $p(t+m+m) + (1-p)(t+m)$

$= p(t+2m) + (1-p)(t+m)$

$= pt + 2pm + t + m - pt - pm$

$\boxed{EMAT = t+m+p(m)}$ ⇒ let EMAT = y

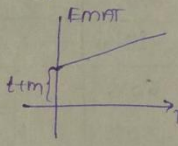
t+m = Constant = C

p(m) = p × m

↓ ↓ Represented by.

x a

⇒ $y = C + x(m)$ ⇒ $y = mx + C$



If TLB miss↑ EAT ↑

If no of levels = K then

EMAT = $p(t + Km + m) + (1-p)(t+m)$

$= p(t + (1+K)m) + (1-p)(t+m)$

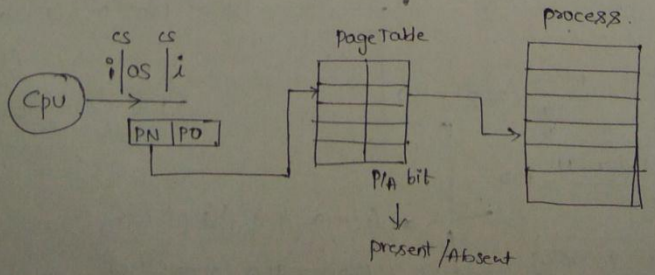
$= pt + (1+K)pm + t + m - pt - pm$

$\boxed{EMAT = t+m+p(Km)}$ p = miss rate not Hit Rate.

48. PAGE FAULT

⇒ Refering to a page that is not present in the memory is called PF.

⇒ Demand paging = Don't load any page until it is required



⇒ Now, the cpu generates the logical Address and with that address we goto 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 swith (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.

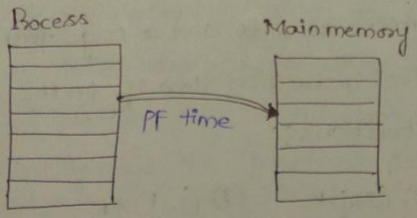
⇒ EMAT in case of page fault is. = p*(service time/ page fault time) + (1-p)(memory Access time)

page fault rate

EMAT = p*(service time) + (1-p)(memory Access time)

Also called page fault time

(with concept of DP)



⇒ Now, initially the process is going to start without loading any page in MM then cpu whenever it generates 'LA' it will findout that the 'pageno' 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. ∴

EMAT = p*(Service time + ma) + (1-p)(ma)

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) 23ns D) 35ns

EMAT = $p * (\text{servicetime} + mA) + (1-p)(mA)$

$= p * (\text{servicetime}) + [p*(mA)] + (mA) - [p*(mA)]$

$= p * (\text{service time}) + (mA)$

$= \frac{1}{10^6} * (10\,ms) + (20\,ns)$ → Access time without page faults.

$= 10\,msec + (20ns)$

EMAT = 30ns

Given

$p = \frac{1}{10^6}$, MA = 20ns

service time = 10ms

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) $(i+j) * K$

EMAT = $p * (\text{Service time}) + mA$

EIT = $i + \frac{1}{k}(j)$ = instruction access time + $p * (\text{service time})$

EIT = $i + (j/k)$ = $i + p * (j)$

$= i + \frac{1}{k} * j = \left(i + \frac{j}{k}\right)$.

51. GATE 2000 QUESTION ON PAGE FAULT

Suppose the time to service a page fault is on average 10msec, while a memory access takes 1μsec. Then a 99.99% hit ratio results in avg. memory access time of?

A) 1.9999 msec B) 1 msec C)

Given MA

Se

H

∴ E

52. GATE

A demand

and 300 t

The ploty

dirty is a

then valu

9) 0.194

⇒ Here the

Check

then

there

⇒ Servic

"

MA =

EMAT = 1 μs

3 =

⇒ 1 + 100

= 200p

memory acess

ory acess,

MA = 20ns

me = 10ms

es an

e a

, while

Given MA = 1μsec

Service time = 10msec

Hit ratio = 99.99% ⇒ miss rate = 0.01% ⇒ 0.0001

∴ EMAT = p * (service time) + MA = (1μsec) + (0.0001)(10 × 10^-3)

= 1μsec + (10^-4 × 10^-2)

= 1μsec + 1μsec

EMAT = 2μsec

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 pbty of page fault is 'p'. In case of page fault the pbty 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) 0.981

⇒ 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 write back the page in the process and then replace it so 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

pbty of not being dirty

PS = (1-p)(100) + p(300)

PFST = 100 + 200p

EMAT = 1 unit m + (p)(PS)

3 = 1 + (p)(100 + 200p)

⇒ 1 + 100p + 200p^2 = 3.

200p^2 + 100p - 2 = 0 ⇒ p = 0.0196

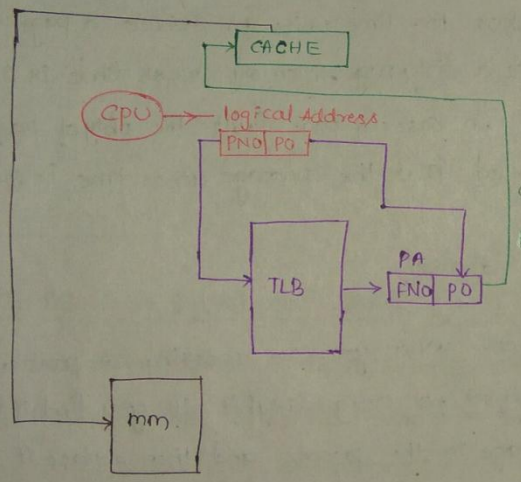
53. GATE 2003 QUESTION ON TLB AND PAGING

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 = Avg time taken to access Virtual Address = (VA → PA) + (fetch the word from process)

(VA → PA): Take the help of TLB

(fetch the word from process): First check in cache if not present then go to MM.

$$\Rightarrow \text{Avg time} = \underbrace{t + (1-p_t)(K*m)}_{\text{conversion from (VA} \rightarrow \text{PA)}} + \underbrace{c + (1-p_c)(m)}_{\text{fetching}}$$

t: Hit rate of TLB; K: No. of levels; c: Hit rate of Cache

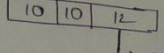
$$= 1\text{ns} + \frac{4}{100}(2*10) + 1\text{ns} + \frac{10}{100}(10)$$

Avg time = 3.8 ns

8) Suppose a process has only the following pages in virtual address space two contigious code pages starting at virtual address 0x00000000, two contiguous datapage starting at VA 0x00400000 and a stack page at virtual address 0xFFFFF000. The amount of memory required for storing page tables of this process (87)

a) 8KB b) 12KB c) 16KB d) 20KB

Now, Address split



↳ Represent page offset ⇒ size of page = 2^{12}B.

= 4KB.

⇒ PTE = PTS = No of pages * size of each PTE

PTS = 4MB.

⇒ 2 pages = 0x00000000 to 0x00000FFF (page offset) and 0x00001000 to 0x00001FFF

0000000000

2 page: 0x00400000 to 0x00400FFF and 0x00401000 to 0x00401FFF

004 = 0000000001 00

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 b) 1230 nsec

EAIT = cpu time + 2* (EMAT)

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

= t + (1 − P_f)(2 * MAT) + m + P_f * ps

= 0 + (0.1)(2 * 150 n) + 150 n + $\left(\frac{1}{10,000}\right)$(8m)

= (30 + 150 + 800) μsec = 980 μsec ⇒ EAIT = 100 ns + 2*(980) = 2060 ns.

55. GATE 2014 QUESTION ON TLB

Consider a paging h/w with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 msec 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 ___ ?

$$EMAT = p(t+m) + (1-p)(t+2m)$$

$$= 0.6(10+80) + (0.4)(10+160)$$

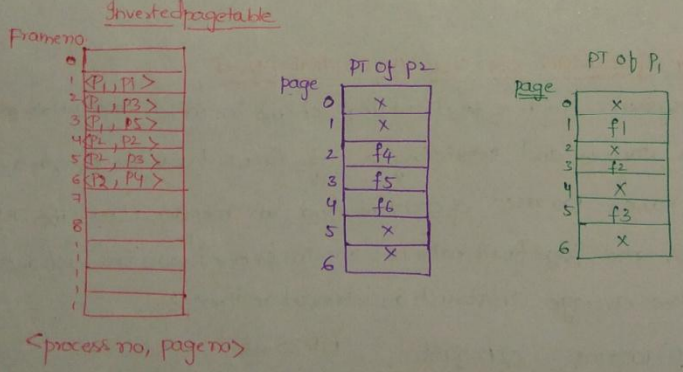
$$= \frac{60}{100} \times 90 + \frac{40}{100} \times 170$$

$$= 54 + 68$$

$$= 122 ms.$$

56. INVERTED PAGE TABLE

Now, Assume there are 10 process running, now the page tables of all the process should be loaded now, the main memory will be occupied by 10 page tables 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.



If PA = 32
PS = 4KB
IPTE = 4B.
} what is the size of IPT ?

IPTS = (No. of frames) * IPTE

$$= \frac{2^{32}}{2^{12}} B * 4B$$

$$= 2^{20} * 4B.$$

$$= 4MB.$$

57. IMPORTAN

Fram

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

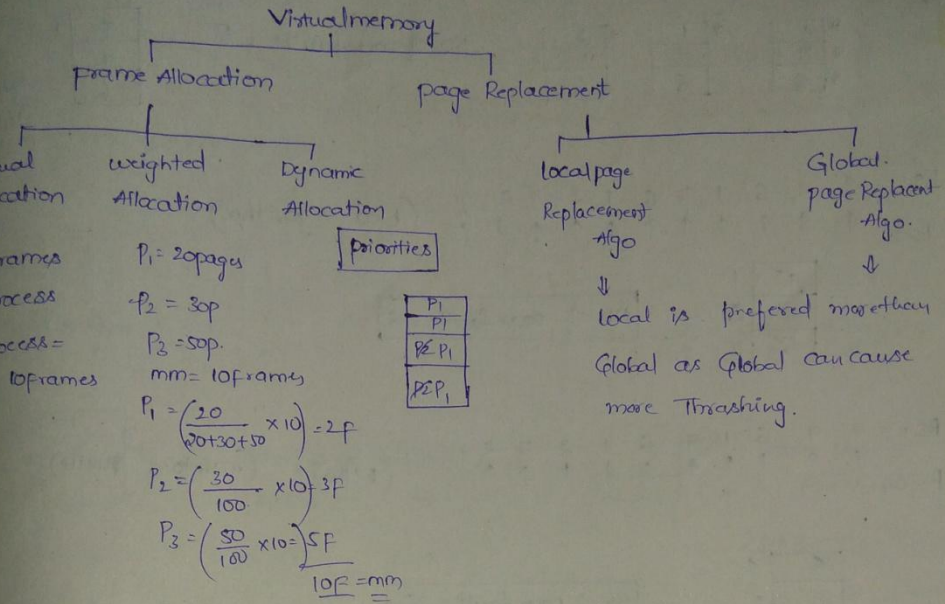
59. QUESTION O

Demand paging

GATE- 2014

Reference string =
Frames = 3.

57. IMPORTANCE OF FRAME ALLOCATION AND PAGE REPLACEMENT



⇒ 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 refered 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

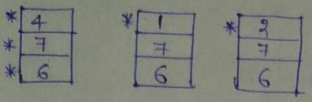
GATE- 2014

Reference String = 4,7,6,1,7,6,1,2,7,2

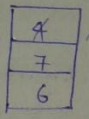
Frames = 3.

RS= 4 7 6 1 7 6 1 2 7 2 (optimal page Replacement Algo) (9c)

F=3 ⇒



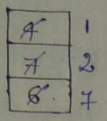
No. of page faults = 5



= No. of letters
= 4,1,2,7,6
= 5 PF

RS = 4 7 6 1 7 6 1 2 7 2

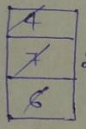
F=3 (LRU Algorithm)



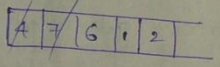
No. of page faults = 6

RS = 4 7 6 1 7 6 1 2 7 2

F=3. (FIFO) (maintain Queue)



No. of page fault = 6

Queue 

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. ⇒ now we need to convert them into page numbers.

0100 = 01 → 0200 = 02 → 0430 = 04

0499 = 04
0510 = 05
0530 = 05
0560 = 05
0120 = 01
0220 = 02
0240 = 02
0260 = 02
0320 = 03
0370 = 03

∴ RS = 01, 02, 04, 04, 05, 05, 05, 01, 02, 02, 02, 03, 03

= 1, 2, 4, 4, 5, 5, 5, 1, 2, 2, 2, 3, 3.

F=1

⇒ 1 ✗ ✗ ✗ ✗ 3

⇒ No. of page faults = 7 = No. of miss

⇒ Miss Rate = 7/13 = PF Rate.

⇒ No. of references made = 13. (length of RS)

61. GAT

RF = 1, 2,

F=3

Optimal

RF = 1, 2,

F = 3

FIFO

64. BEL

RF : 0

F : 3.

Optimal

F=3.

F=4.

PF =

FIFO

F:3

Hit rate = 1 − miss rate

= 1 − 7/13 = 6/13.

61. GATE QUESTION ON PAGE REPLACEMENT ALGORITHMS

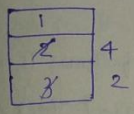
RF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1

F = 3

Optimal

RF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1

F = 3



No. of page faults = 5

PFR = 5/10

LRU

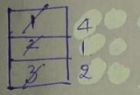
RF = 1, 2, 3, 2, 4, 1, 3, 2, 4, 1

F = 3

No. of page faults = 6.9

PFR = 9/10

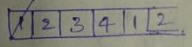
FIFO



No. of page faults = 10

PFR = miss rate = 6/10 = 60%

Hit rate = 1 − 6/10 = 4/10 = 40%.

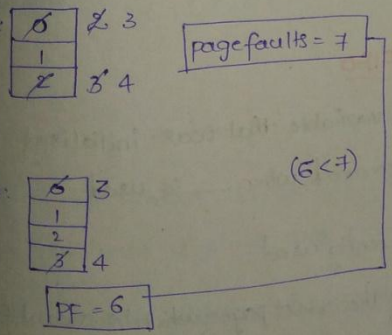


64. BELADY ANAMOLY

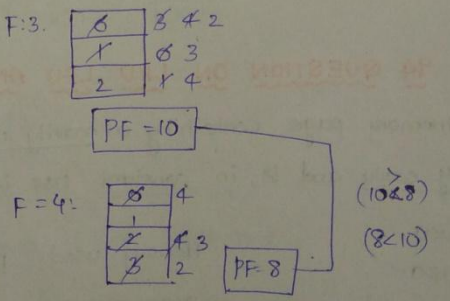
RF: 0 1 2 3 0 1 4 0 1 2 3 4

F: 3.

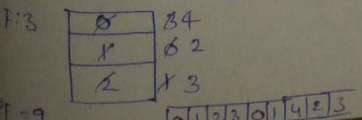
Optimal

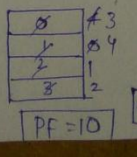


LRU



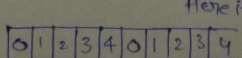
FIFO





Belady Anamoly.

Here it is different



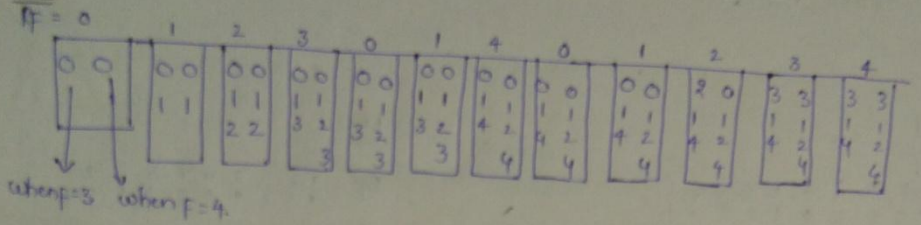
65. STACK ALGORITHMS

⇒ The main reason for Belady Anamoly is "stack properity."

⇒ optimal properity is a stack Algorithm

⇒ whenever a Algo is a stack Algorithm it does not follow Belady Anamoly

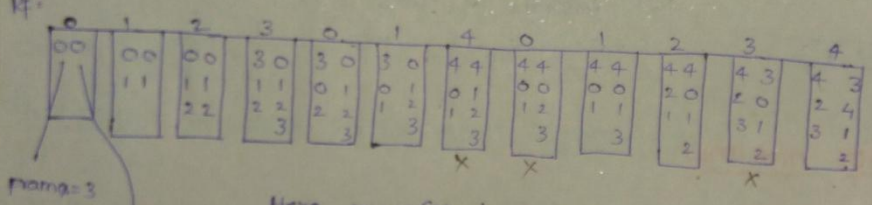
optimal



pages(3) ⊆ pages(4)

pages(m) ⊆ pages(m+1) → If an Algo follows this properity then it is called stack Algorithm and it does not follow Belady Anamoly

FIFO Algorithm



Here pages(3) ⊄ pages(4)

pages(m) ⊄ pages(m+1) ⇒ It is not stack Algorithm and so it follows Belady Anamoly

66. 94 QUESTION ON LRU, LFU AND FIFO

A memory page containing a heavily used variable that was initialized very easily and is in constant use is removed when ____ is used

a) LRU
b) FIFO ✓
c) LFU
d) None

Heavily used = frequently used
very easily = one of the oldest pages which is available
constant use = Recently used page

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 donot occur for these 4 pages because they are already present in frames

96 PF

∴ Total PF = 100 + 96

= 196 PF's.

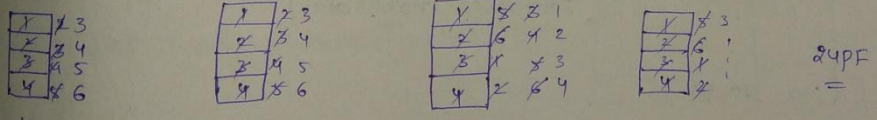
69. SOME INTRESTING BEHAVIOUR OF OPTIMAL PAGE REPLACEMENT

⇒ 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 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6

optimal = 12 PF MRU = 12 PF LRU = 24 PF FIFO = 24 PF



12 PF

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

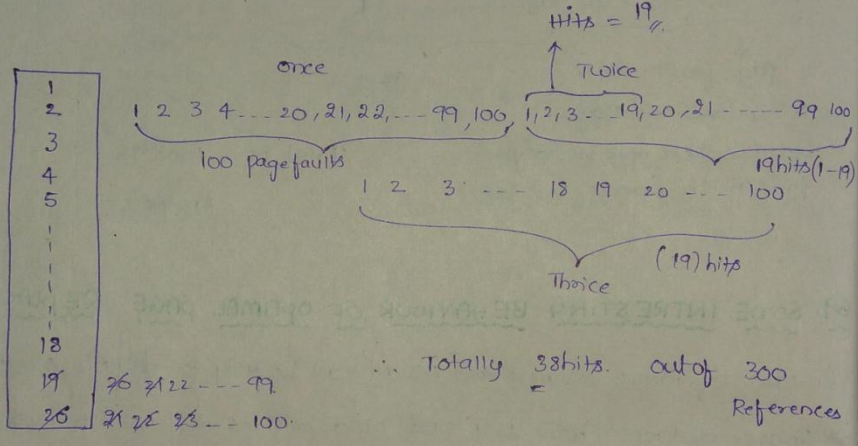
⇒ MRU uses a single page frame and it gives worst case

⇒ It is very difficult to implement optimal practically.

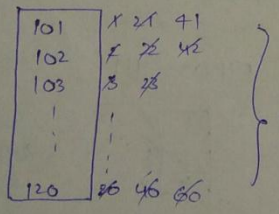
70. GATE 2014 QUESTION ON LRU, MRU, LIFO, FIFO AND OPTIMAL

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,d) 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

bered
00 in
e following
optimal

99 100
19hits(1-19)
100
its

300
References

the following
ns

11. WORKING SET ALGORITHM

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 ≤ window size

$w \leq A$

w = {1,2,3} A = [2 3 1 2]

w = {1,2,3} 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:

a d e | 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 = $\frac{4+3+3+3+3+4+3+2+3+4}{10}$

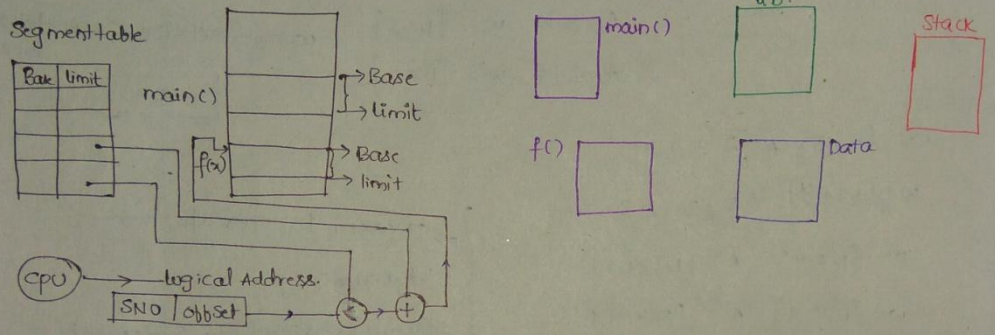
= 3.2

75. 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

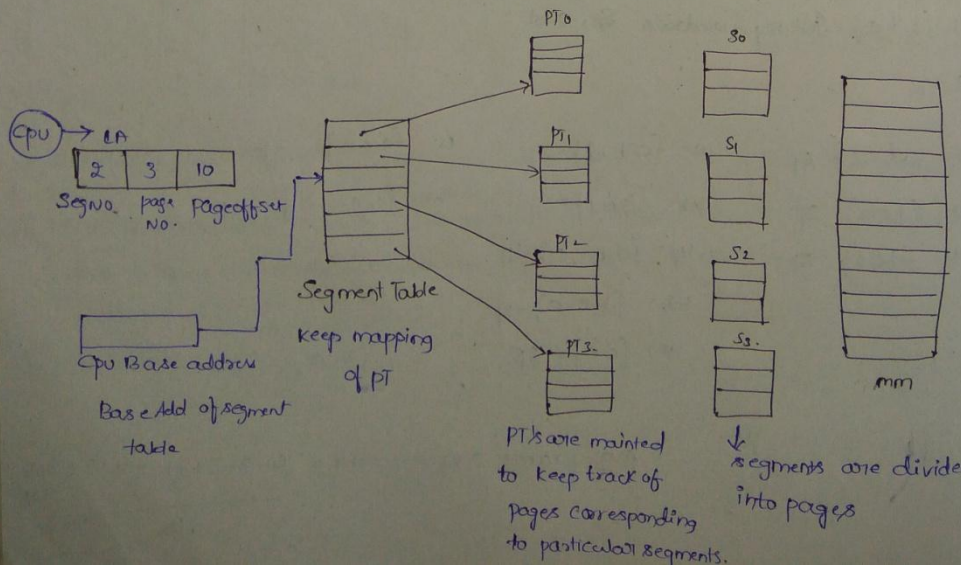
⇒ Segmentation gives the impact of division as how we assume the division should be

⇒ The program is divided into no of segments and each segment is loaded in main memory. It have 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
↓ y x
3 bits becua

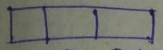
Now, PTS

2^x

⇒ 2

17. GATE 99 QUESTION ON SEGMENTATION PAGING

LA = 32 bits

⇒ 

= 16 bits

↓ 3 y x

3 bits because 8 segments are present ⇒ $3+x+y = 16$

⇒ $\boxed{x+y = 13}$

Now, PTS = No. of pages * (PTE)

$2^x = 2^y * 2$

⇒ $2^x = 2^{y+1}$ ⇒ $x = y+1$

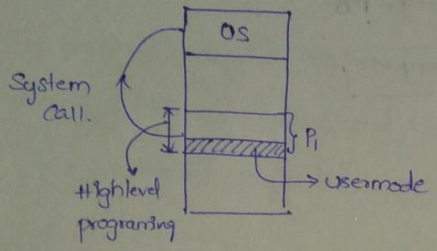
⇒ $x+y = 13$

⇒ $y+1+y = 13$ ⇒ $\boxed{y=6 \quad x=7}$ ∴ 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.

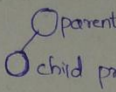
3. FILE MANIPULATION

- → 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 SYSTEM CALL

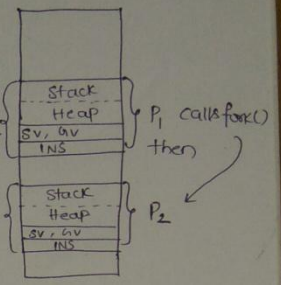
⇒ fork() / execute() is the systemcall

⇒ when any process calles fork() system call it will create a child process



⇒ when fork() is called there should be context switch and we should shift from usermode → Systemmode

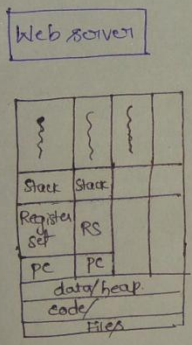
Overhead.



⇒ fork returns i=0 for child process.

fork returns (pid of child) for parent process } fork() creates two process (one is already existing one and new one)

8. PROCESS VS THREADS



process (fork())	Thread (user level)
→ System calls are involved	→ NO systemcall
→ context switch	→ Register set switch
→ diff copies of code and data	→ same copies of code and data.

Thread: New, Ready, Running, blocked, Terminated

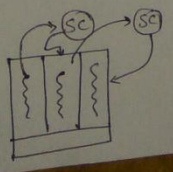
9. USER LEVEL V-S KERNEL LEVEL THREADS

→ The disadvantages of user level threads is

→ Blocking systemcall will block the whole task.

→ unfair scheduling

Kernel level threads



⇒ parallelism is guarenteed.

⇒ DisAdv: Expensive compared to ULT

⇒ switching requires more time (requires systemcall for switching)