

82

26th June, 2010

Saturday

Operating Systems

Topics :

- (1) Background
- (2) Process management
 - * Process concepts
 - * CPU scheduling
 - * IPC & synchronisation
 - * Concurrency
 - * Deadlocks
 - * Threads
- (3) Memory management
 - * Concepts
 - * RAM chips
 - * Techniques
 - * Virtual memory
- (4) File system & I/O management
 - * Interface
 - * Device characteristics
 - * Implementation issues.
- (5) Protection mechanisms

(Q). What is an operating system ?

Ans: Interface between user and Hardware

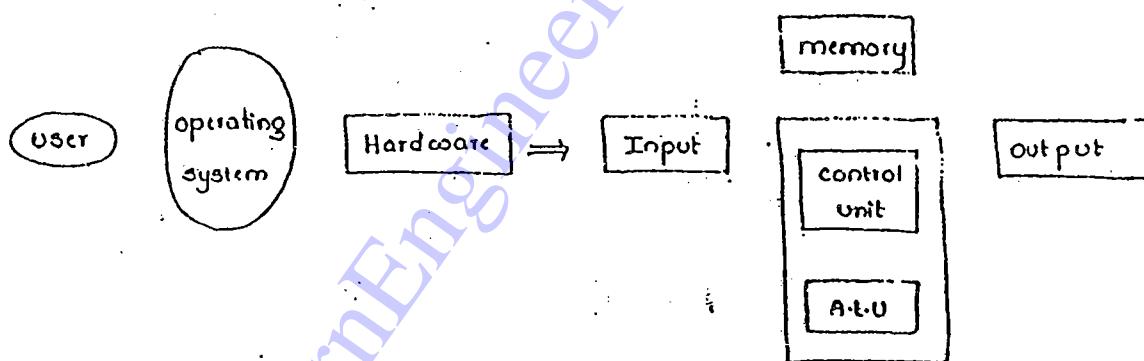
* Resource manager

* control program(s)

* A set of utilities to simplify application development.

* Acts like a government.

Von-Neumann Architecture :



i) Control signals

ii) Mathematical operations (Data, Register)

clock cycles.

$$c = a + b$$

Load R₁ m[a]

Load R₂ m[b]

Memory :

* Primary (main memory, physical \Rightarrow) RAM, ROM, cache, Registers

* Secondary (Auxiliary \Rightarrow) HD / DVD / Pendrives

* Processor cannot fetch data directly from secondary memory. Instead Instruction Register (IR) is used, from where instructions are fetched and executed sequentially.

* If IR requires any operands, then it fetches the operands and executes. as :

- * Fetch cycle
 - * Decode cycle
 - * Execute cycle
 - * Interrupt cycle
- Instruction cycle:

* The communication among user and CPU can be represented by Von-Neumann architecture.

Main Objective of operating System:

* It takes the complete control of Hardware, and create a platform, which is easy to user to write different applications.

Command Interpreter:

* A program that interprets the commands of the user.

* It acts as Interface between the user and Kernel.

* These are of two types :

* Text based O.S (Shell) : DOS, UNIX, NETWARE

* GUI based O.S (Desktop) : WINDOWS, LINUX.

* Operating system is a set of control program(s).

* Operating System is visualised as resources.

Hardware

- * CPU
- * memory
- * I/O

Software

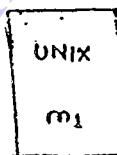
- * Device drivers
- * Semaphores
- * messages
- * monitors

Functions & Goals :

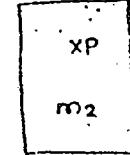
Goals :

- (1) convenience (user friendly)
- (2) Efficiency (resource utility)
- (3) Reliability & Robustness
- (4) Scalability (Ability to evolve)
- (5) portability (different platforms)

Consider an example :



Execution time: 50 sec



75 sec

To find a particular content in "UNIX" O.S., we must write its syntax.

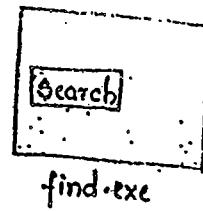
Eg:- \$find

./name "text.c"

so, that it requires exact syntax, but it takes less time to execute the given statement.

To find the same in "XP" O.S. It is an easy process that any user can

approach through (i.e., clicking "start" button & then "Search")



- * It takes more time than UNIX to execute, but it is more familiar to all kinds of user to operate.
- * So, "convenience" is more important than "efficiency".
- * In some real time systems, "Efficiency" is more important than "convenience".

First Generation

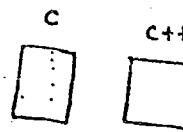
No operating system

Card Readers, Punch card

Second Generation

manual Intervention.

Magnetic Tapes



Batch processing
↓
Clubbing similar program
at one place & processed
one at a time.

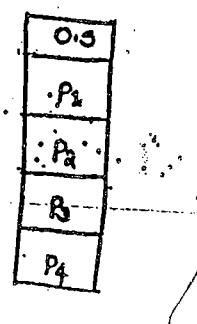
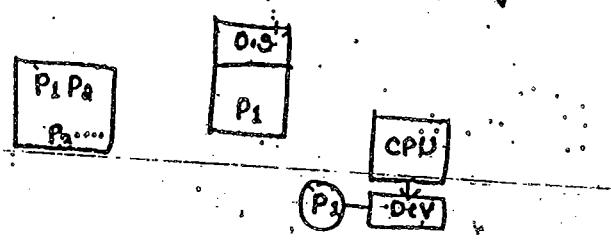
Third Generation

Hard Disk Technology

*(wastage of
idleness of CPU)

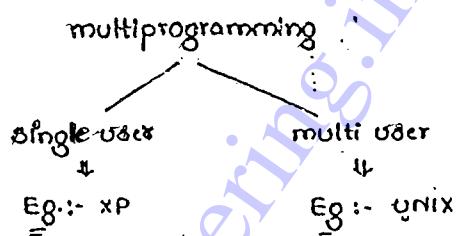
Uniprogramming
(DOS)
main memory

multiprogramming
(UNIX, XP)



Multiprogramming:

- * Ability of operating system to hold multiple ready to run programs in the main memory so that if the running program requires I/O, the CPU can be switched to another ready program.
- * Multiplexing of CPU among ready programs in memory.



Non-Pre-emptive:

- * Running process is released voluntarily (on its own) as follows:
 - * Completion of process
 - * I/O request

Pre-emptive:

- * Running process can be forced to release based on:
 - * Priority
 - * Time sharing

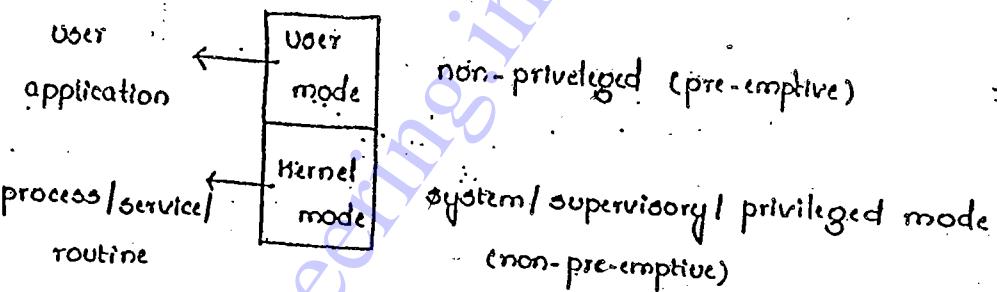
Time sharing Systems (pre-emptive):

Eg.: - windows xp

first version of windows : 3.0
3.1 } non-preemptive
3.11

Architecture of multiprocess support

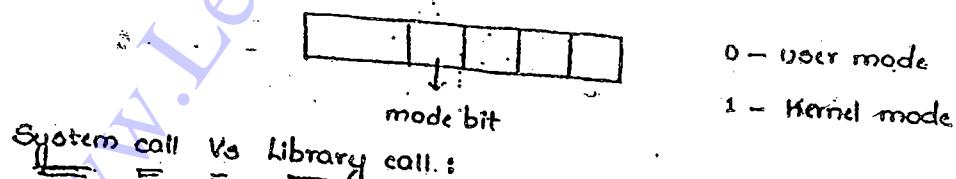
- (1) Direct memory Access (DMA) \Rightarrow Since computation & I/O required, we need DMA.
- (2) Address Translation
- (3) Atleast two modes of CPU execution.



- * In Kernel mode, the process have atomic execution (without pre-emption)
- * In user mode, the process have non-atomic execution (with pre-emption)

Program status word (PSW):

It represents the mode, in which the user operates in.



System call Vs Library call.:

main()

{
 int a,b,c;

 scanf("%d %d", &a, &b); \rightarrow (library call) \Rightarrow (BSA) \Rightarrow Branch & save

 In std. C = a+b;

 Library fork(); \rightarrow (system call) (sysc)

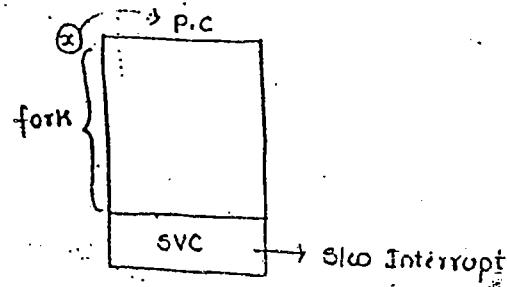
 printf("Hello"); \rightarrow (Supertlibrary call)

- * fork() creates child process and the execution of fork() is in kernel mode, because it is a part of O.S. and the conversion takes place (from user mode to system mode)
- * Execution of supervisory call (SVC) involves the software interrupt, which handles Interrupt Service Routine (ISR).
- * When an SVC is generated, ISR changes to 1 (non-preemptive mode). Then after the completion of all the functions in Kernel mode, again an SVC is generated, where ISR value is changed from 1 to 0 (pre-emptive mode).

- System call Interface (SCI) \Rightarrow UNIX
- In windows O.S., SCI is considered as API.

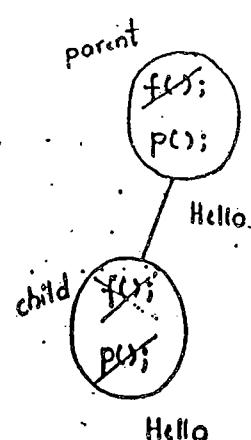
ISR \Rightarrow changes
mode bit
&
Dispatch the
table

fork	(x)



```
2) main():
{
    fork();
    printf("Hello");
}
```

Output : Hello
Hello



(a) main()

```

    {
        fork();
        fork();
        printf("Hello");
    }
  
```

- * If 1 fork \Rightarrow 2 prints
- 2 forks \Rightarrow 4 prints
- 3 forks \Rightarrow 8 prints

$$n \text{ forks} \Rightarrow 2^n = \text{Total}$$

$$\text{ncwo} = 2^{n-1}, \text{ parent} = 1$$

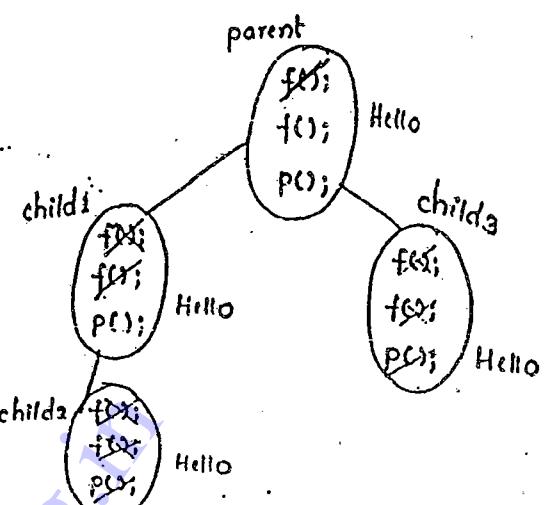
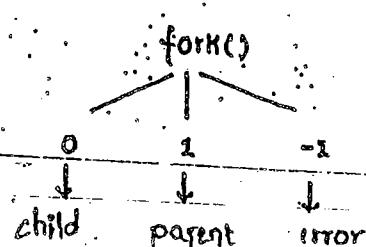
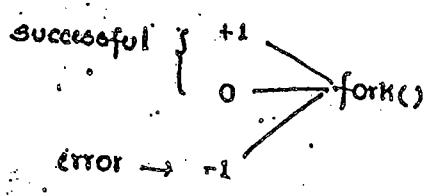
(b) main()

```

    {
        int i, n;
        for(i=1; i<=n; ++i)
            fork();
    }
  
```

$$(a) n-1 \quad (b) n! = 1 \quad (c) n^2-1 \quad (d) 2^n-1 \quad (\text{ncwo forks})$$

- * Any System call may return either +1, 0, -1 values.

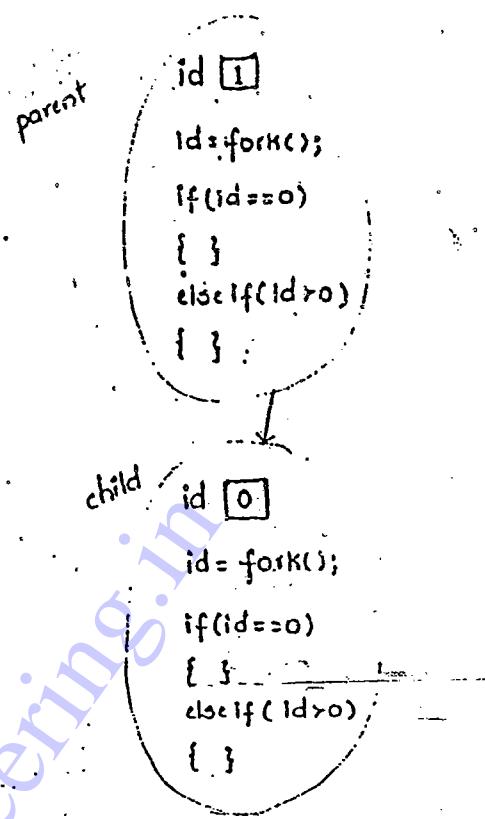


Output :
 Hello
 Hello
 Hello
 Hello.

```

main()
{
    int id;
    id = fork();
    if(id == 0) // child
    {
        // child //
    }
    else if(id > 0)
    {
        // parent //
    }
    else printf("Error in creation");
}

```



Difference between Kernel mode & user mode:

Compilers, editors and similar application-independent programs are not part of the O.S. even though they are typically supplied by computer architectures. This is crucial, but subtle point. The O.S system is that portion of the software that runs in Kernel mode. It is protected from user tampering by hardware.

- compilers & editors runs in user mode. If a user doesn't like a particular compiler, he/she is free to write their own clock interrupt handler, which is a part of O.S.

01st July, 2010

Thursday

Process concepts

Program Vs Process :

* Program under execution.

* Unit of execution

* Schedulable / Dispatchable unit

Process : Program

* Instance of program

1 : 1

* Central locus of control (COC)

1 : multiple

* Animated spirit

* Executing program resides in secondary memory:

Program	Process
* In secondary memory	* In main memory
* Without resources	* Utilization of resources
* Passive	* Active (Alive)

Formal definition of process :

* Developer / Designer views process as a simple Datastructure.

* Every Datastructure is always defined with four parameters:

* Definition

* Representation (process structure)

Implementation

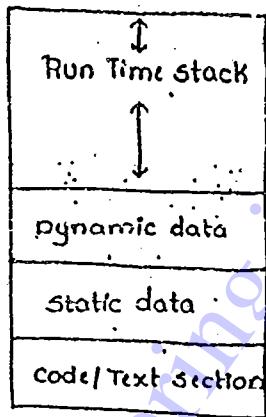
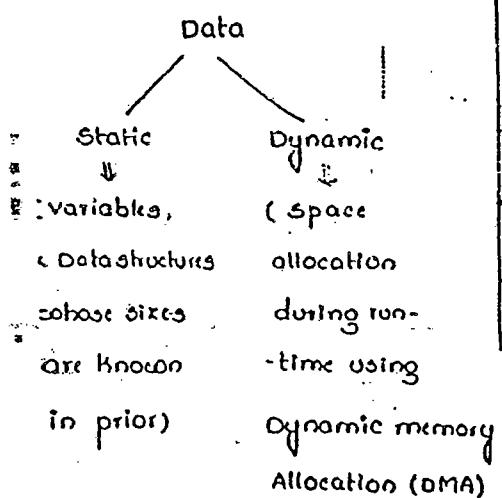
* Operations

* Attributes (push(), pop())

Process Structure (Representation) Implementation) in main memory :

Abstract view of process:

- * program consists of data and instructions.



Run Time stack \Rightarrow Activation records are maintained wherever function calls are generated.

Every process maintains this stack for storing addresses (recursive) i.e. use this stack.

Operations :-

* Essential resources required for stopping the execution of process, then some resource utilities are used.

* When a program is loaded in main memory, resources are allocated and then CPU schedules all the process.

* Create (resource allocation)

* Scheduted (CPU)

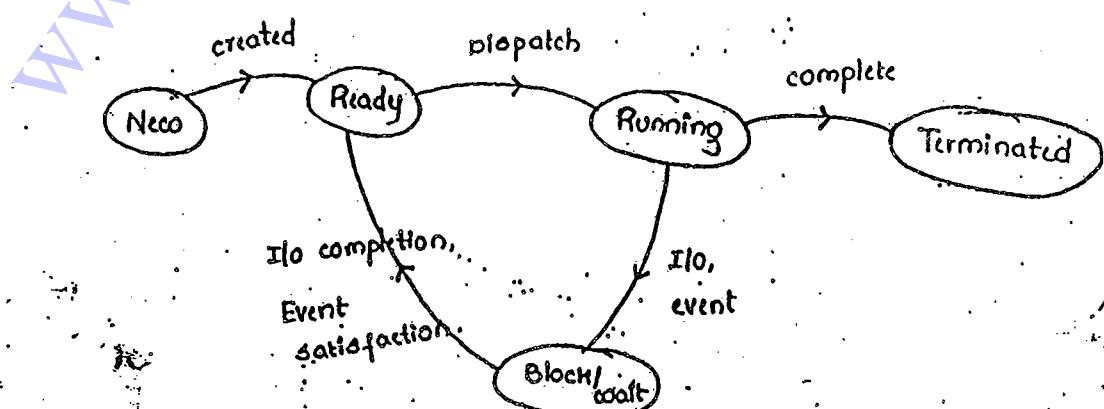
- * Execution (CPU)
 - * Blocked (I/O)
 - * Resume
 - * Suspend()

Attributes :-

 - * Process Id (pid)
 - * Priorities
 - * Process state
 - * Program counter
 - * memory limits
 - * list of files
 - * list of devices
 - * Type & size (fn b)
 - * Protection.

- * All the process attributes are stored within the process control block (PCB)

* * Process State :-



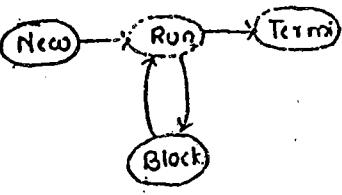
Process control Block (PCB)

Pid	State	Priority
PC	CPR	
mtr limits	files	
Devices	i	

process context / process environment.

Above diagram represents :-

(a) Uniprogramming (no "ready" is present)

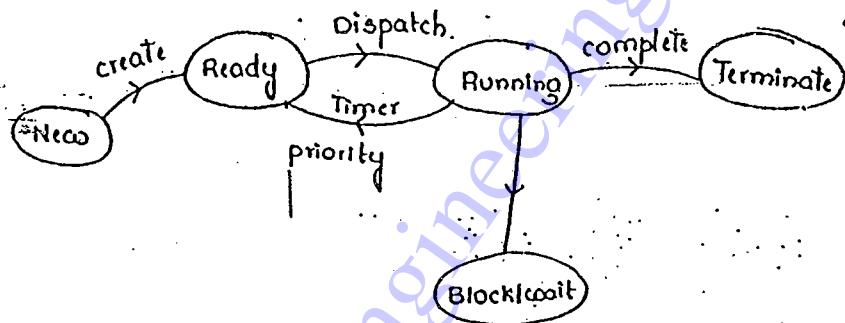


(b) pre-emptive multiprogramming

(c) non-preemptive multiprogramming

(d) multiple CPU based OS

Pre-emptive multiprogramming :-



Degree of multiprogramming = no. of processes

Degree ↑, when process ↑, so, we suspend process whenever required.

process suspension => swapped out on temporary basis and later on resumed.

There is no suspension for the following process states : ↑ performance

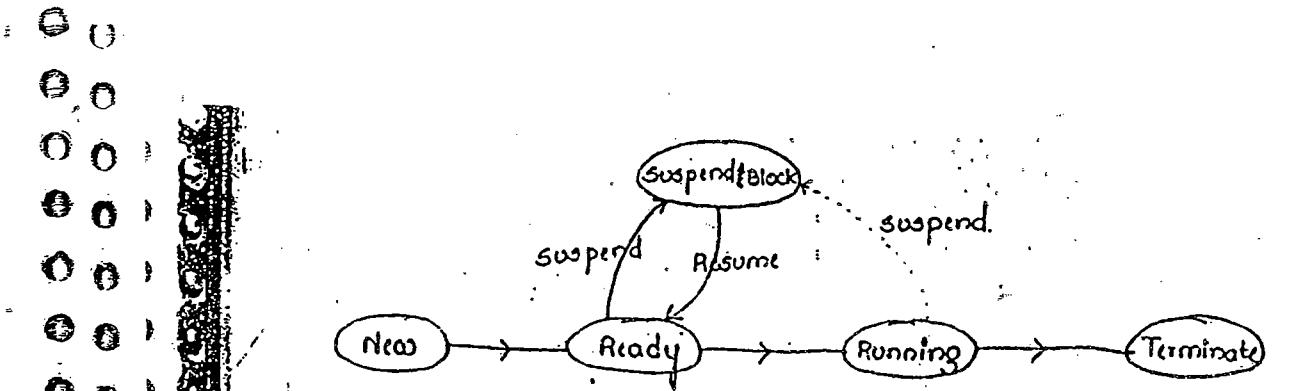
* New

* Terminate

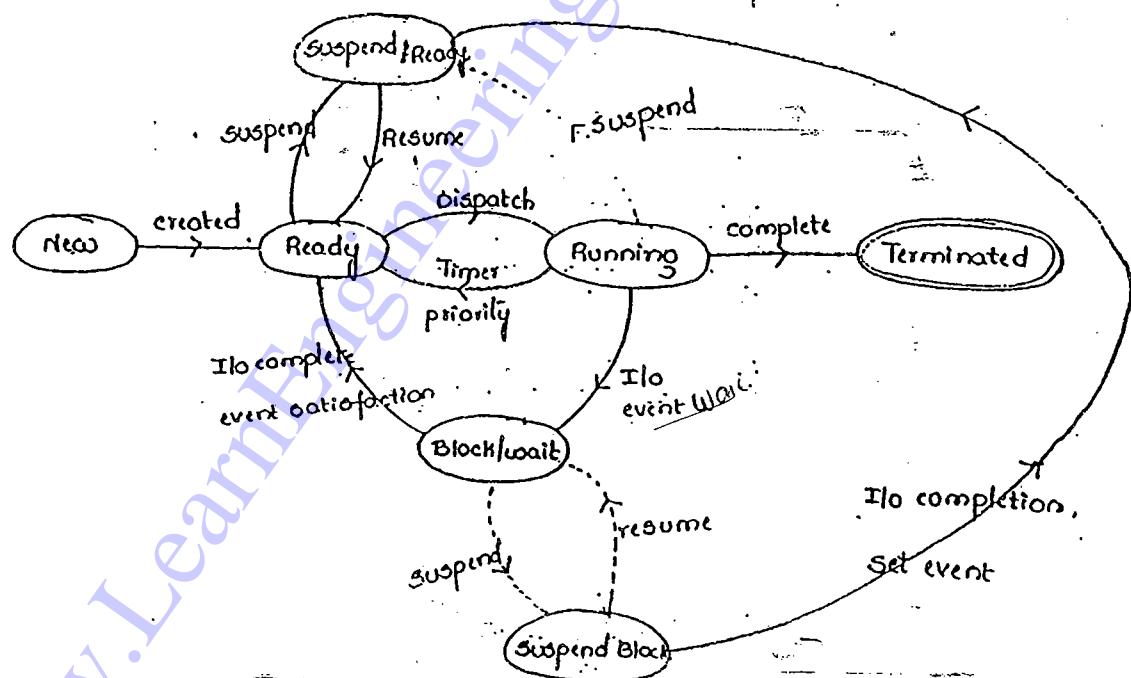
Most desirable state for suspension is : "Ready" state.

Block state Because, it is waiting for I/O or other events, so, when suspended it gets confused.

If "Ready" process is suspended, then it moves to "suspend" state.



Block/wait
When "Ready" state is suspended.



- * Only Ready, running, Block states \Rightarrow can be suspended
- * processes waiting for I/O operations are not suspended (dangerous). If g suspended, it moves to suspend block state.
- * Suspension of running processes are not preferred always. If suspended, it releases the resources pre-emptively.
- * So, "Ready" state is the safe to suspend any process, they are being temporarily stored in secondary memory.

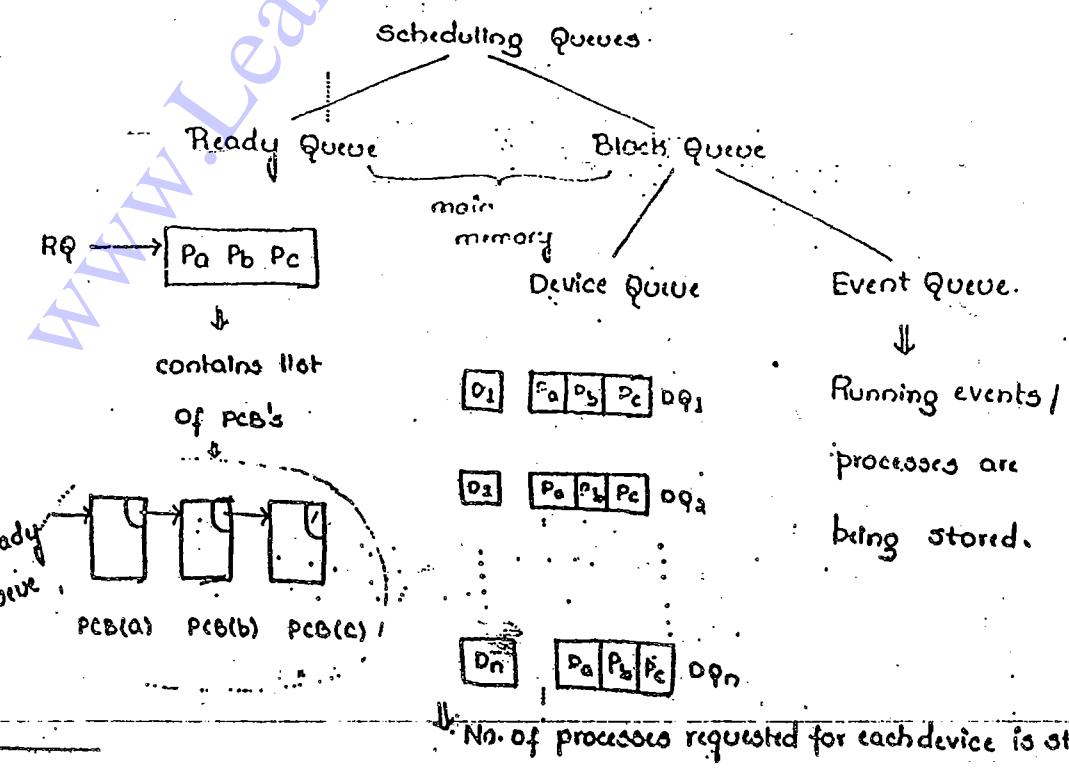
(Q) consider a system with 'n' CPU's and 'm' processes where $n \geq 1$ and $m \geq n$. calculate lower bound and upper bound on the process states.

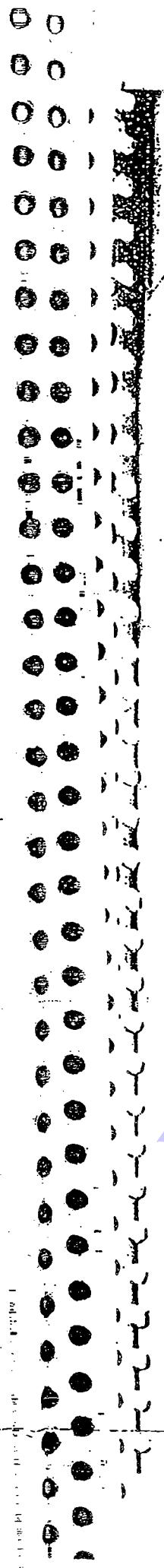
- * Ready
- * Running &
- * Block states.

Ans :- maximum Ready state processes are = m
 maximum running state processes are = n .

	Min LB	Max UB
Ready	0	m
Running	0	n
Block	0	m

Scheduling Queues & State-Queuing Diagram :-



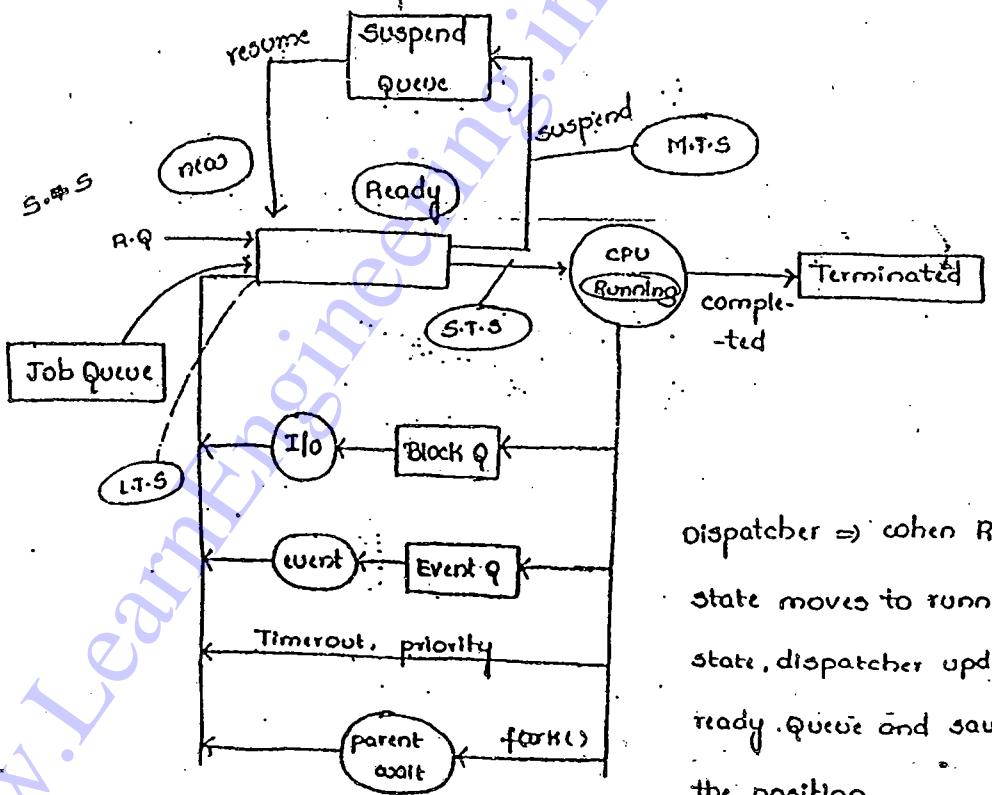


Scheduling Queues

Suspend Queue \Rightarrow It contains suspended processes.

Job Queue \Rightarrow It contains PCB's of Job Queue.

Process State Queue diagram :-



Dispatcher \Rightarrow when Ready state moves to running state, dispatcher updates ready queue and saves the position.

Long term Scheduler - loading new programs from Secondary to main memory
Job Queue

Short term scheduler(CPU Scheduler) :-

It decides that which process (ready process) should run next.

Middle term scheduler (Suspended Queue),

(It decides that which process should be suspended and resumed)

- A process is said to be CPU Bound if it generates minimal or very few I/O request and does more computation.
- A process is said to be I/O bound if it does most of I/O than computation.
 - * Switching a process to another on CPU is called "process switching".
 - * Saving and loading the PCB's from one process to another is called "context switching".
 - * "context switching" has direct impact on volume of information in PCB.

multiprogramming \Rightarrow long term scheduler.

CPU Scheduling

CPU Scheduler :

- * Decision of which process to run is taken.

Goals :-

- * Maximize CPU utility
- * Minimize Response time, waiting time.

Process Times :

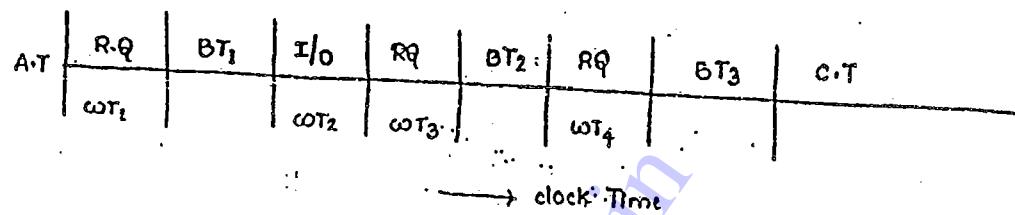
- * Arrival Time (A.T) - Submission time.
- * Waiting Time (Ready Queue / BQ)
- * Burst Time (CPU) - Service Time (B.T)
- * Completion Time (C.T)
- * Turn Around Time (TAT) = completion Time (C.T) - Arrival Time (A.T)

Response Time (R.T) :

- * Time of submission of request by a process, to obtain a result / response (first response).

Deadline (D) :

* No. of processes completed within the deadline.



Notation :

(i) n-processes (P_1, \dots, P_n)

(a) A.T (P_i) = A_i

(b) B.T (P_i) = x_i

(4) C.T (P_i) = c_i

(5) Deadline (P_i) = D_i

-or-

Formulae :

(a) TAT (P_i) = $c_i - A_i$

(b) WT : TAT (P_i) = $\frac{c_i - A_i}{x_i}$

(c) Avg. TAT (P_i) = $\frac{1}{n} \sum_{i=1}^n (c_i - A_i)$

(d) WT (P_i) = TAT - BT

$$= c_i - A_i - x_i$$

(e) Avg. WT (P_i) = $\frac{1}{n} \sum_{i=1}^n (c_i - A_i - x_i)$

(f) Schedule length, (l) = $\max(c_i) - \min(A_i)$

(g) Throughput (μ) = $\frac{n}{l}$

b) Deadline overrun = $C_i - D_i$

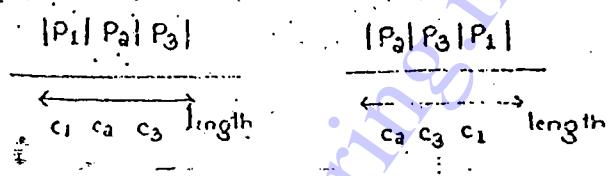
If $(C_i - D_i) < 0$ under run

If $(C_i - D_i) = 0$ on deadline

If $(C_i - D_i) > 0$ overrun.

Schedule :-

n-processes. n=3. (P_1, P_2, P_3)



04/07/2010

Wednesday

CPU Scheduling Techniques

Non-preemptive

pre-emptive

(1) First come first serve (FCFS) :

Criteria : Arrival Time (process)

mode : non-preemptive.

$$TAT = CT - AT$$

$$WT = TAT - BT$$

P.NO	A.T	B.T (CPU)	CT	TAT	WT
1	0	4	4	4	0
2	1	5	9	8	3
3	2	6	15	13	7
4	3	8	23	16	12
5	4	9	30	16	14
6	5	4	24	19	15

Gantt chart :

CPU	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	
	0	4	9	15	18	20	24

Arrival Time of P₁ = 0

$$P_3 \text{ C.T} = P_1 \text{ C.T} + P_3 \text{ B.T.}$$

i.e., at '0' clock time, P₁ arrives

$$\begin{aligned} \text{completion Time (P}_n\text{)} &= \text{completion Time (P}_{n-1}\text{)} \\ &+ \\ &\quad \text{Burst Time (P}_n\text{)} \end{aligned}$$

Eg. :-

PNO	AT	BT	CT	TAT	WT
1	8	4	20	12	8
2	5	5	13	8	3
3	7	2	16	9	7
4	2	1	3	1	0
5	4	3	8	4	1
6	6	1	14	8	7
7	3	2	6	2	0
			18		

$$TAT = CT - AT$$

$$WT = TAT - BT$$

Gantt chart :

	P ₄	P ₇	P ₅	P ₂	P ₆	P ₃	P ₁		
	0	2	3	5	8	13	14	16	20

$$L = 20 - 2 = 18$$

$$L = \max. \text{ of } C_i - \min. \text{ of } A_i$$

$$= 20 - 2$$

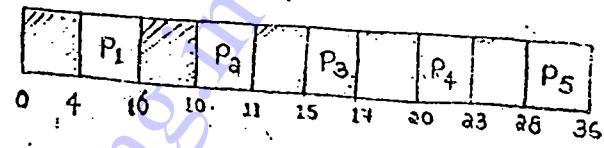
$$= 18$$

Since, there are no processes arriving at clock time = '0'. The process P₄ arrives at '2' clk time. So, the time (0-2) is subtracted from the total completion time. And, since P₄ arrives earlier than that of process P₁.

Process 'P₄ is considered first and then the other processes arrived within the particular Burst time.

Eg 3 :

PNO	AT	BT
1	4	2
2	10	1
3	15	3
4	20	3
5	28	8
<u>16</u>		



$$WT(P) = 0$$

$$L = 36 - 4 = 32$$

Shortest Job First (SJF):

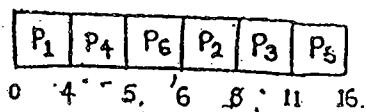
* It is also called as Shortest process next (SPN).

Criteria : Burst Time

Mode of working : Non-preemptive (default).

It also works under pre-emptive mode.

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



Initially, Process P₁ is ready for execution, since it has arrival time = 0. So, it has been processed.

for processing 'P₁', it requires 4 clock cycles ($B.T=4$), within the time, processes P₂, P₃, P₄, P₅ are also ready for execution.

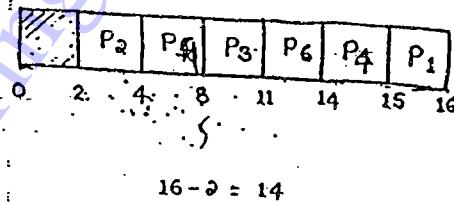
Within those processes, select the process which have less

Burst time and process it. Here, P_4 has a $B.T = 1$. So, it is selected.

\therefore Total clock cycles = $4+1=5$. At the time of 6th clock cycle, process, P_5 (new) is available on ready queue. Since, the $B.T=1$ of process P_5 , after completion of P_4 , P_5 is to get executed, even though it added recently in queue.

Eg:-

PNO	AT	BT
1	5	1
2	2	2
3	3	3
4	2	4
5	4	1
6	3	3
		14



(3). Shortest Remaining Time first (SRTF):-

In pre-emptive mode, SJF is known as shortest Remaining Time first.

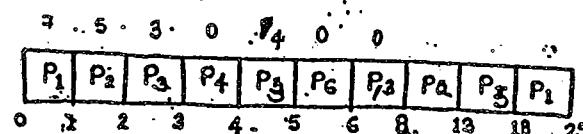
"Pre-emption of running process is based on the arrival of a new shorter process".

Criteria :- Burst Time

mode :- pre-emptive.

Eg:-

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

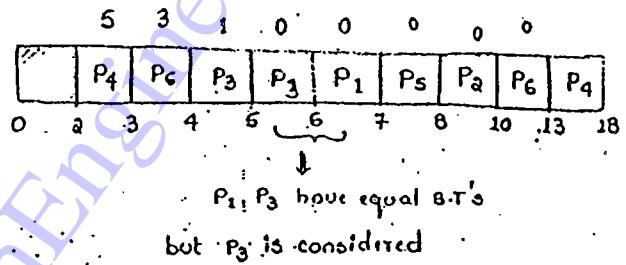


$$TAT(P_3) = 8 - 2 = 6$$

Initially, process 'P₁' has arrived, and within 1 clock cycle time arrives, then P₃ & soon. So, only 1 clock cycle is considered for P₁ within B.T = 8, so the remaining B.T = 7 remains. In P₂, B.T = 5, P₃ \Rightarrow B.T = 3, P₄ \Rightarrow B.T = 0,

- At clock cycle = 4, process 'P₅' also arrives, but if we consider it, it has more burst time than that of Process 'P₃'. So, we consider 'P₃' as it is SRTF. Next, within 5 clock cycles 'P₆' also arrives, which have B.T = 1, so we consider 'P₆' after 'P₃', and so on.... upto the end.

PNO	AT	BT
1	5	12
2	7	9
3	4	9
4	2	6
5	6	1
6	3	4



Performance of Shortest Job First (SJF):

Burst Time
Advantages:

- * Enhances Throughput
- * Average W.T & TAT reduces.
- * It is implementable, with predicted/ Estimated Burst times.

Disadvantages:

- * Starvation to longest job.
- * It is non-implementable, because burst times of processes are not mentioned.

Prediction Techniques for Burst Time :

(1) Process Size : (Bytes)

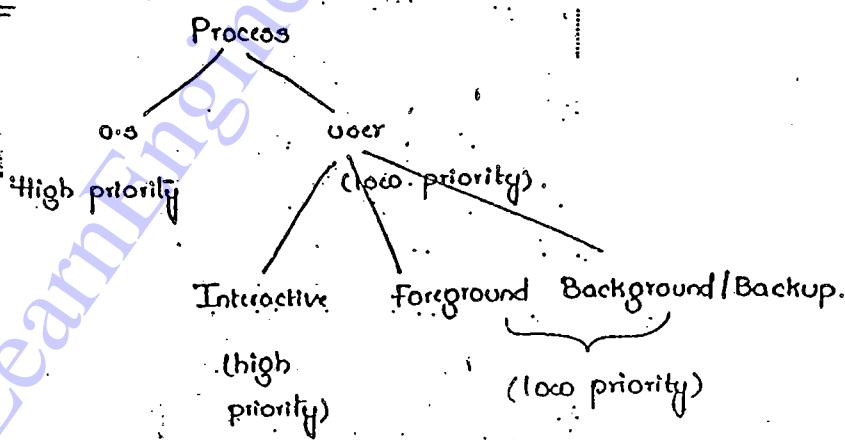
Based on past records, burst times; New burst time's are predicted for new records. functional change may present.

Assume $P = 102 \text{ KB}$

$$P_{old} = 102 \text{ KB} \quad (Q)$$

$$P_{new} = 100 \text{ KB} \quad (Q)$$

(2) Process Type :



(3) Next CPU burst is the average of its previous Bursts :-

Let P_i be the process. Let t_i be the real known Burst Time.

Let \bar{t}_i be the predicted Burst time.

Then $\bar{t}_{n+1} = \frac{1}{n} \sum_{i=1}^n t_i$

$$\bar{t}_i = c \text{ (given)}$$

P_i	RQ	CPU	I/O	RQ	CPU	RQ	CPU	RQ	...
	wt_1	t_1	wt_2	wt_3	t_2	wt_4	t_3	wt_5	

(4) Prediction based on Exponential Averaging / Aging Algorithm :-

Let $T_i \rightarrow$ predicted B.T

$t_i \rightarrow$ Real B.T

completed $\rightarrow n$ -CPU Bursts.

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n \quad \rightarrow (1)$$

$$0 \leq \alpha \leq 1$$

$$\text{If } \alpha = 0 ; \quad T_{n+1} = T_n.$$

$$\text{If } \alpha = 1 ; \quad T_{n+1} = t_n$$

$$\text{If } \alpha = 0.5 ; \quad T_{n+1} = \frac{1}{2}(t_n + T_n)$$

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n \quad \rightarrow (1)$$

$$T_n = \alpha t_{n-1} + (1-\alpha) T_{n-1}$$

$$T_{n+1} = \alpha t_n + (1-\alpha) [\alpha t_{n-1} + (1-\alpha) T_{n-1}]$$

$$= \alpha t_n + \alpha(1-\alpha)t_{n-1} + (1-\alpha)^2 T_{n-1} \quad \rightarrow (2)$$

$$= \alpha t_n + \alpha(1-\alpha)t_{n-1} + \alpha(1-\alpha)^2 t_{n-2} + (1-\alpha)^3 T_{n-2} \quad \rightarrow (3).$$

(4) Highest Response Ratio Next (HRRN):

$$\text{Criteria : Response Ratio (RR)} = \frac{\infty + s}{s}$$

where $\infty \rightarrow$ waiting time of process so far.

$$s = \frac{\text{Service time}}{\text{Burst time}}$$

$$\text{when } \infty = 0, \text{ RR} = \infty = 1,$$

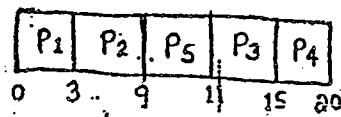
All processes must have $\text{RR} \geq 1..$

mode of working : non-preemptive.

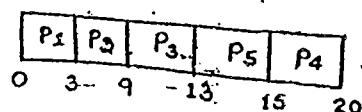
Non-preemptive SJF :-

P-NO	AT	BT
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

NP-SJF :



HRRN :-



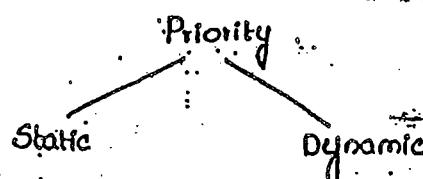
* process having high R.R. it is first considered.

* At clock cycle 9, Process P₅ is in Ready Queue, which have very little B.T. So Process P₃ executed instead of P₅ because of long time waiting. This scheduling algorithm gives importance to long waiting processes as well as B.T. processes.

(5) Priority based Scheduling :

Criteria : priority

Mode : non-preemptive



(not changed
throughout its execution)

Increasing the priority level regularly at the runtime is called as "Aging Algorithm".

Drawback :

Starvation.

Eg:-

Priority	PNO	AT	BT
4	1	0	4
5	2	1	5
6	3	2	8
8	4	3	6
2	5	4	3
12	6	5	5

* Higher Integer
high priority

NP - priority :-

P ₁	P ₄	P ₆	P ₃	P ₂	P ₅
0	4	10	15	23	28

preemptive priority :-

CPU	3	4	7	5	2	0
	P ₁	P ₂	P ₃	P ₄	P ₄	P ₆

a:

Prio.	PNO	AT	BT
12	1	5	4
9	2	6	5
7	3	2	4
6	4	1	6
4	5	4	9
8	6	3	3

	5	3	2	1
	P ₄	P ₃	P ₆	P ₆

Round-Robin (m.p / Time sharing) :-

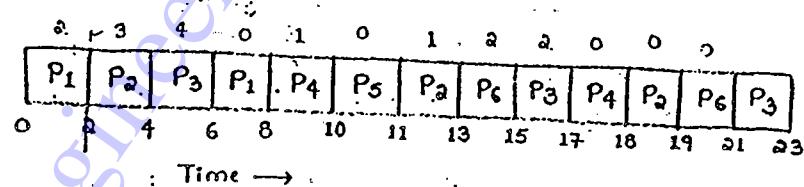
Criteria : $A.T + \underbrace{\text{Time Quantum (TQ)}}_{\text{Slice}}$

mode : pre-emptive

$$TQ = \theta.$$

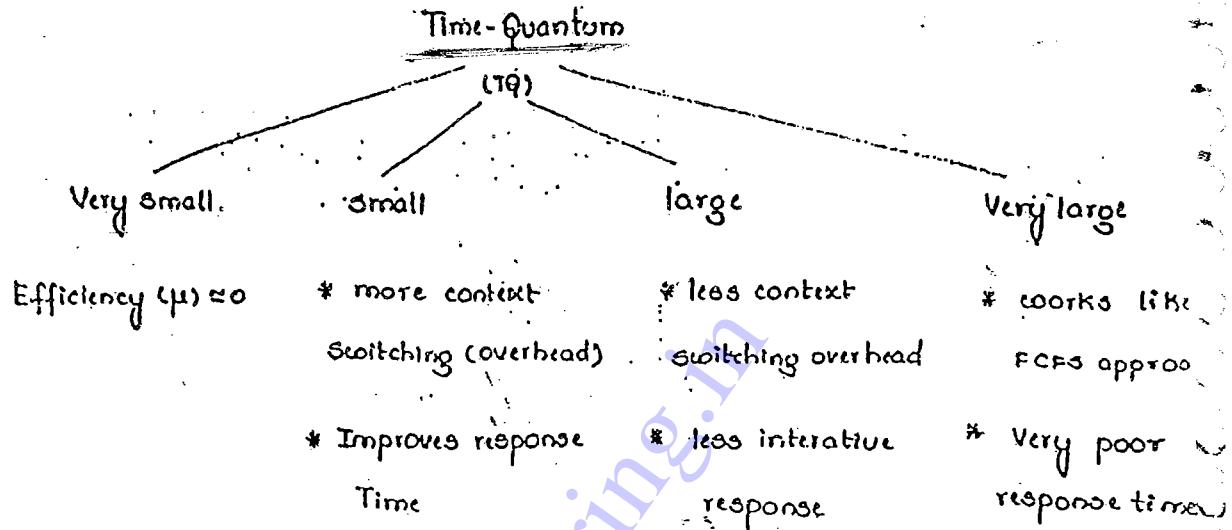
PNO	AT	BT
1	0	4
2	1	5
3	a	6
4	3	3
5	4	1
6	5	4

R.Q : $P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_3 P_4 P_2 P_6 P_3$



- * Initially, P_1 arrives and executes till $B.T = a$. (Since Time slice = a). Then, within a clock cycles, processes P_2, P_3 also arrives, then considered their $B.T$ at $TQ = a$ and hence P_1 requires another a clk cycles it must again be maintained in Ready Queue.
- * At time = 8, all the processes arrives, based on their $B.T$'s, ready Queue is maintained.
- * If TQ is very small, efficiency (μ) = 0.
- * If TQ is low \Rightarrow context switching occurs. { so TQ value must be chosen moderately. }
- * If TQ is high \Rightarrow FCFS approach is followed

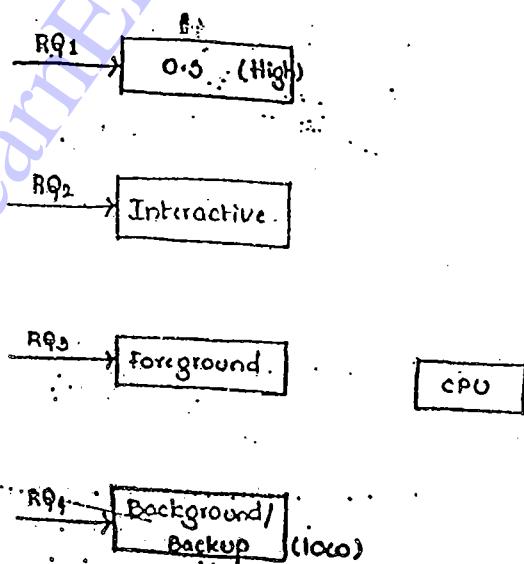
Performance of Round Robin:



08/07/2020

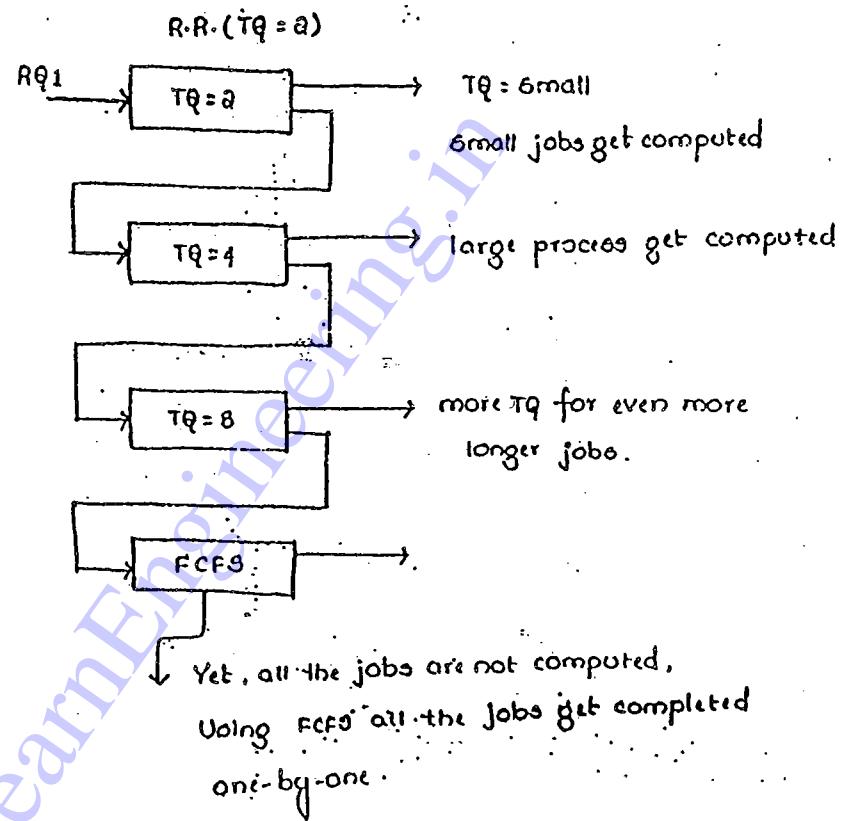
Thursday

Multi-level Queue Scheduling



- * In multi-level Queue, the drawback is that, the second queue is serviced only when first queue is empty / complete.

- * Here, the feedback is pre-empted to the same Queue.
- * So, we have multi-level feedback Queue to overcome this drawback.

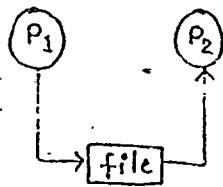
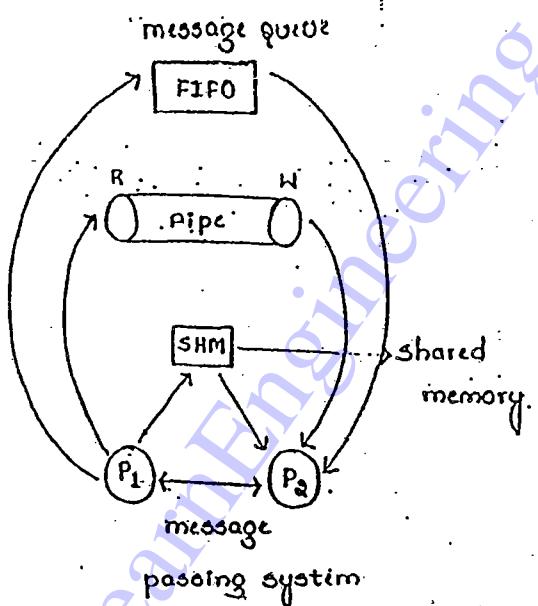


Inter Process communication & Synchronization:

Inter process communication (I.P.C):-

* For the communication among any two process, there must be a media to communicate.

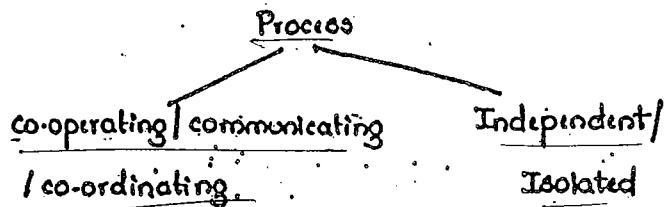
Eg. :- file (It is not always used)



Problems due to lack of synchronisation :-

- * Loss of data
- * Inconsistency (wrong results)
- * Dead locks.

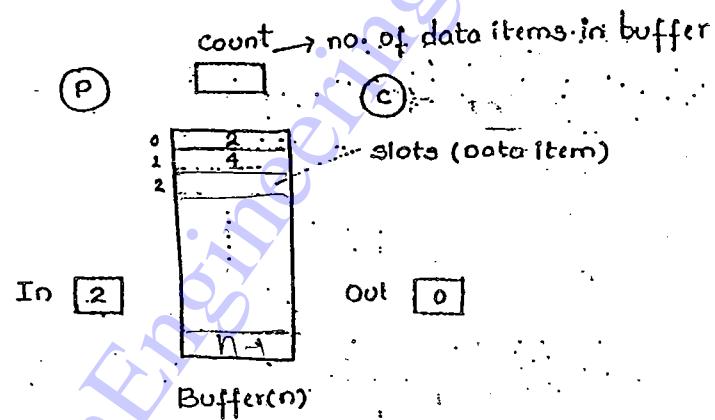
How to achieve Synchronisation?



Co-operating process :-

- * Two (or) more processes are said to be co-operative, if they get affected (or) affects the execution of other process.
- * Otherwise, they are said to be Independent.

Producer-Consumer problem :-



- * If Buffer is full, producer get affected.
- * If Buffer is empty, consumer get affected.

#define n 100

int Buffer[n];

```

void producer(void)
{
    int itemp, in=0;
    while(1)
    {
        ProduceItem(itemp);
        while(count==n); // Busy wait;
        Buffer[in] = itemp;
    }
}

```

Ram
Uma

```
In = (In+1) mod n;  
count = count + 1;  
}  
  
void consumer(void)  
{  
    int itemc, out=0;  
    while(1)  
    {  
        while(count == 0);  
        itemc = Buffer[out];  
        out = (out+1) mod n;  
        count = count - 1;  
        processitem(itemc);  
    }  
}
```

Lack of synchronization in Producer-consumer problem :-

Inconsistency :-

In producer,

- count = count + 1; // This step needs to fetch.
the variable "count" and
then performs the operation.
- (1) Load Rp, m[count]
- (2) Inc Rp
- (3) Store m[count], Rp

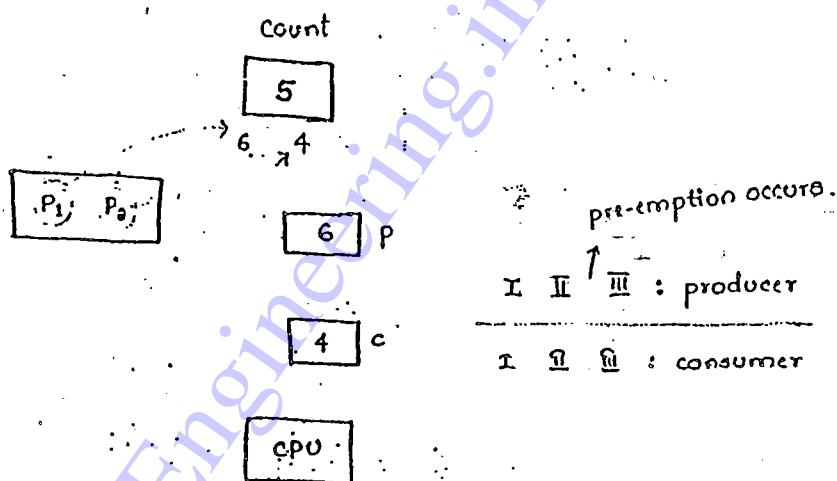
In consumer,

$$\underline{\text{count} = \text{count} - 1;}$$

(1) Load R_c, m[count]

(2) Dec R_c

(3) Store m[count], R_c



* In producer,

while first two steps get executed, pre-emption occurs. i.e.,

consider the count = 5 (initially). In step 1, the value is loaded into register R_p. and then in step 2, the value in the register get incremented. so, the value of count = 6.

* Before storing the "count" value from the register, it got pre-empted. so, the producer have the count value = 6..

* In consumer,

count value = 5 is stored in a register R_c, and then decremented. Before trans-

-ferring the value from R_c it get pre-empted. so it have count = 4. so, there is inconsistency, because count value is never equal to 5.

13/07/2010

Tuesday
=

Synchronization Mechanisms.

Producer - Consumer :

Lack of Synchronization :

- * Inconsistency
- * Loss of data
- * Deadlocks

Terminology :

1) Critical Section(s) & non-critical section(s) :-

- * Critical section is that part of a program where shared resources are accessed.
- * Non-critical section is that part of a program where no shared resources are accessed.

CG = shared resource (any variable/device)

(a) Race conditions (concurrency) :

processes racing to get into critical section.

(B) pre-emption :

A process gets executed in the middle of other process.

count = count + 1

(1) load

(2) Inc:

(3) store.

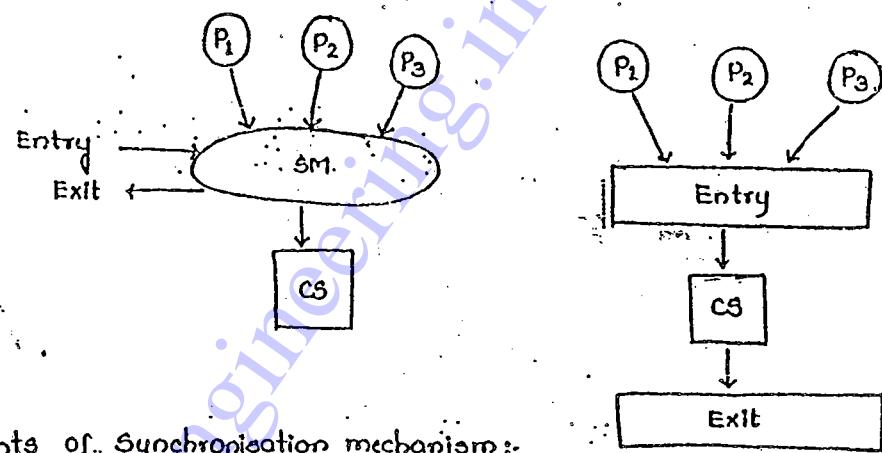
(4) mutual exclusion:

No two processes may be present in the critical section at same time.

Architecture / model of Synchronisation mechanism :

Synchronisation mechanism :

- * To ensure mutual exclusion so that there is no problem of Race condition among processes.

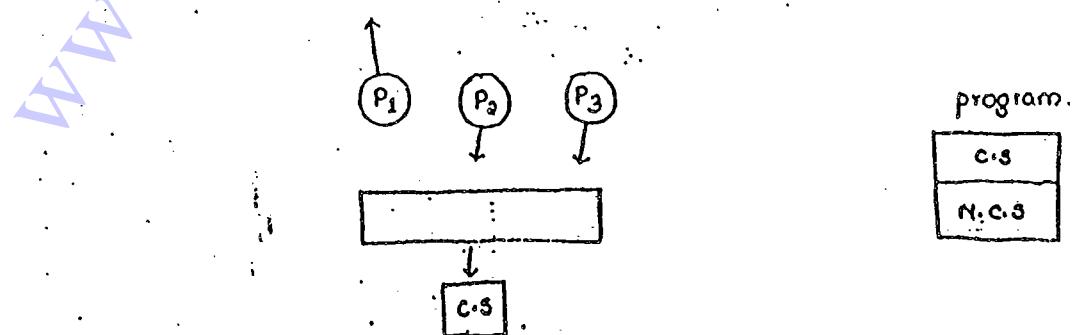


Requirements of Synchronization mechanism :

(1) mutual exclusion must always be guaranteed.

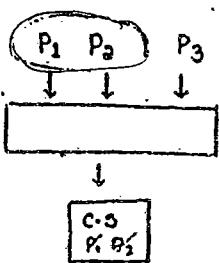
(2) progress :

process running outside critical section should not block (the process of the other interested process from accessing / entering the critical section).



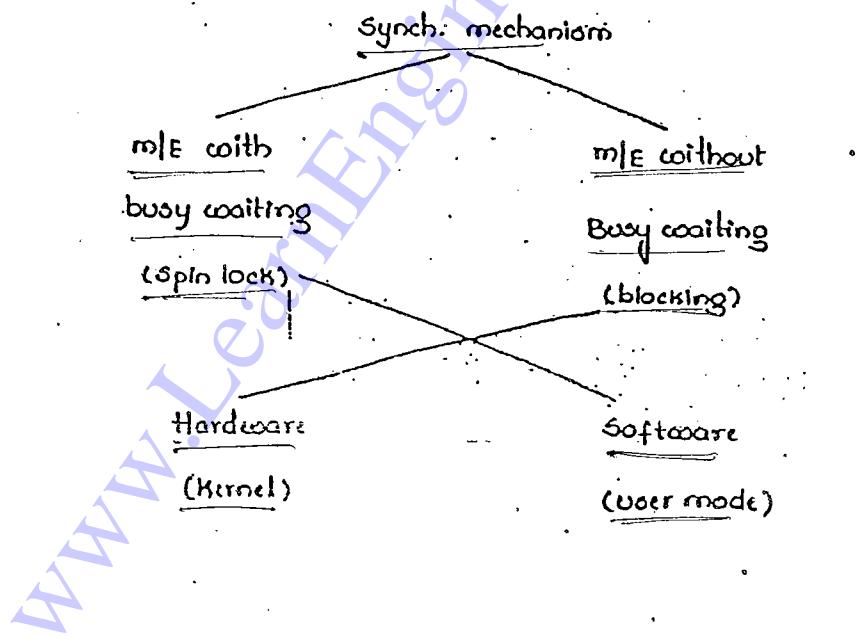
(3) Bounded waiting :

* No process has to wait forever to access its critical section.



If P_1 enters C.S. and after leaving, if P_2 wants to enter C.S., then it can enter and after performing operations, it leaves. Then, if again P_1 wants to enter along with P_3 , where P_1 has high priority than P_3 , then P_1 enters & then P_3 and so on, where there is no chance for P_2 to enter C.S. (i.e., P_2 remains idle forever).

- (4) No Assumptions about the speed (or no. of CPU's) may be assumed.

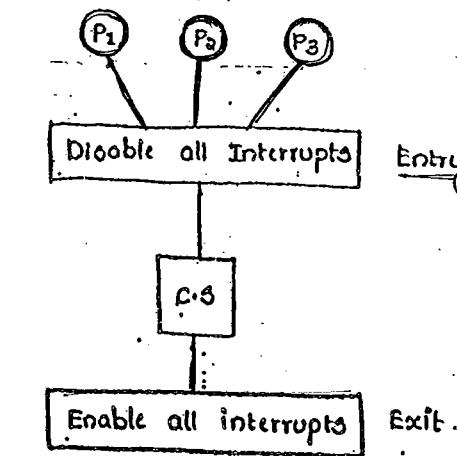


Disabling Interrupts :-

- * no E without busy waiting.
- * multi-process
- * Kernel mode.

- * The main cause of pre-emption is "interrupt".
- * Disable the source of pre-emption in the critical section at entry level and enable the interrupts at exit level.
- * It guarantees m/E because if P₁ running in c.s., other interrupts are disabled.
- * Enabling/ disabling of interrupts done only in kernel mode.
So, O.S process can only enable/disable the interrupts and user process cannot enable/disable \Rightarrow Restriction to O.S process.

* critical section
* race condition
* pre-emption.

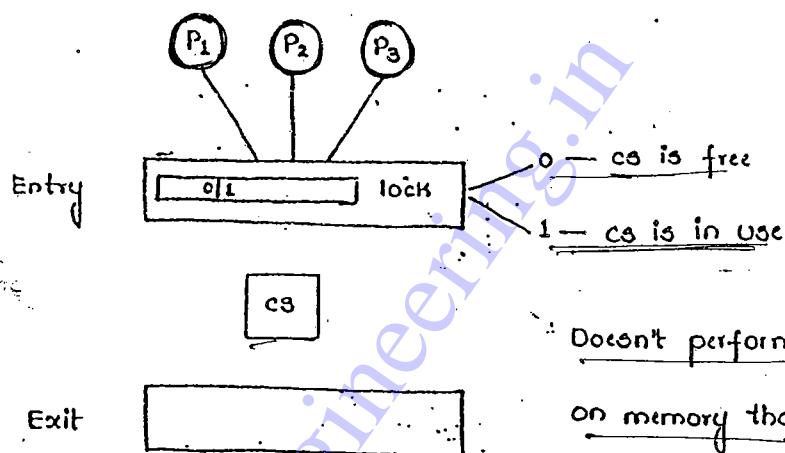


(a) Lock variable:

* Slooo mechanism implemented in user mode.

* MIE with busy waiting.

* multiprocess.



Doesn't perform direct manipulation
on memory that load into ALU
reg.

```
void Entry_section(int process)
{
    while(lock!=0); // Busy waiting.
    lock=1;
}
```

Here, initially the lock value is '0' (lock=0)

when, P₁ needs to enter c.s, it follows
all the steps.

Step 1: The lock value is stored in the register
'R_i'.

Step 2: It is compared whether lock is '0' or not

Step 3: If lock value is not zero (i.e., lock=1), then again process same steps (1&2)

Step 4: If lock=0, then the process enters c.s and while, entered then it changes

lock = 1.

Assembly language :-

Entry Section :

1. load R_i, m[lock]

2. cmp R_i, #0 ;

3. JNZ steps

4. store m[lock], #1 ;

c.s



lock

P1

Entry Section

R1 0

R2 0 CPU

C.S P1

C.S P2

P1 : 1 2 3 preempt

P2 : 1 2 3 4

5

P1 : 4 5

Exit Section :

Store m[lock], #0 ;

- * It fails to guarantee mutual exclusion.
- * Busy waiting results in wastage of CPU cycles / CPU time.

Hint for testing either supports to guarantee m/E :-

(1) Identify the shared variable (lock)

(2) Read the value of shared variable (load)

(3) compare (optional)

(4) Update / action (based on result of comparison)

→ If process 'P₁' entered c.s, and before changing lock=1, it get pre-empted.

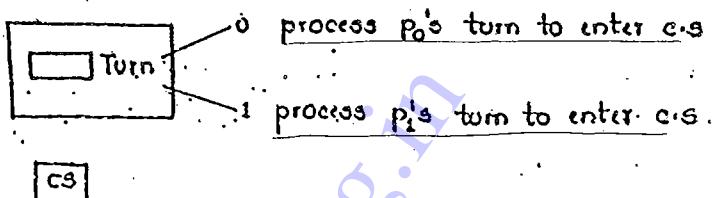
Since the lock=0, another process 'P₂' wants to enter c.s and get resides in CPU, and gets into c.s.

* Since P₁ and P₂ gets into c.s simultaneously, it fails to guarantee mutual exclusion.

30-100P 20

(3) Strict Alternation :

- * Slow at user mode: $(P_0, P_i) \text{ or } (P_i, P_j)$
- * a-process
- * mle with busy waiting.



- * Strictly on alternate basis, processes (P_0 & P_1) takes turn to enter c.s.

void $P_0(\text{void})$

{

while(1)

{

Non-cs()

Entry

while(turn!=0); // busy waitingcs

Exit

turn=1;

Because "turn" value is
not immediately updated,
but updated at exit section,
so, it guarantees mle.

Void $P_1(\text{void})$

{

while(1)

{

Non-cs()while(turn!=1); busywait P_1 untiltwin become 1csturn=0;

- * Guarantees mle.

- * Not Guarantees progress

Direct

progress

Indirect

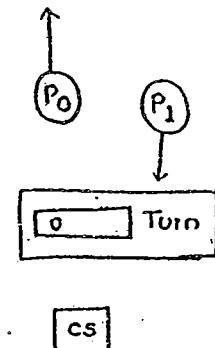
progress

direct Progressdirect progress :-

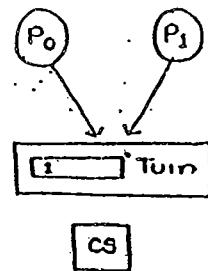
Initially, the turn=0, and if P_0 is not interested in CS, and P_1 is interested, then the turn must be equal to '1' to enter into CS. But, unless ' P_0 ' enters, "turn" can't be changed, and ' P_0 ' not interested to do so. Hence, progress is not guaranteed.

Indirect progress :-

If ' P_0 ' enters CS and while exiting turn changed to '1', so that ' P_1 ' need to enter. If again ' P_0 ' wants to enter (i.e., interested) and P_1 is not interested. Then, P_0 is discarded to enter CS because turn=1 and it needs P_1 to change turn=0. Hence, progress is not guaranteed.



' P_0 , which is outside block
' P_1 directly not to enter CS.



P_1 is blocking P_0
indirectly.

(4) Peterson's Solution (Dekker's Algorithm):-

- * m/E with busy waiting

- * slow at user mode.

- * 2-process solutions (P_0 & P_1)

- ↓
 - disadvantage

```
#define N 2
#define TRUE 1
#define FALSE 0

int interested[N] = {false};
int turn;
```

```
void entry-section(int process)
```

- {
- (1) int other;
- (2) other = 1 - process;
- (3) interested[process] = TRUE;
- (4) turn = process;
- (5) while(interested[other] == TRUE && turn == process); // busy waiting

```
void exit-section(int process)
```

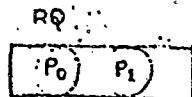
- {
- interested[process] = false;
- }

* progress is guaranteed, m/e guaranteed.

* Bounded waiting is guaranteed (Since, it is not guaranteed for more than 2 processes).

Drawback:

* wastage of CPU time.



interested[0] = F

int'd[0] = F T

interested

turn



P₀ : 1, 2, 3

P₁ : 1, 2, 3, 4, 5

P₀ : 4, 5

P₁ : CS

HARWARI P SOLUTION

5. Test & Set Lock instruction (TSL) :-

(privileged instruction \Rightarrow atomically)

- * Slow at user mode
- * mLE with busy waiting
- * multi-process (advantage over peterson)

Eg :- TSL register, flag ;

0 - CS is free
1 - CS is busy

** "copies the value of flag into register and sets the value of flag to one (always)".

Entry-section :

- (1) TSL regi, m[flag];
- (2) cmp regi, #0;
- (3) JNZ step1
- (4) [CS]
- (5) store m[flag], #0;

Priority Inversion (P/TSL) :

- * Preemptive priority based CPU scheduling is used.

If $H > L \Rightarrow$ Deadlock

when a high priority process enters the CPU. And P_L requires CPU to perform its instructions, inorder P_H to enter into e.g. so, P_H enters in CPU and needs to enter CS & P_L is in CS and needs CPU to exit. Hence, Deadlock occurs.

RQ :

P_H

CPU
PH PL

P/TSL

Entry
Sectio

CS
PL

Producer & consumer :

* M/E without Busy waiting (Blocking)

(i) Sleep() and wakeup() :-

void producer(void)

{

int itemp, in = 0;

while(1)

{

produce item(itemp);

if (count == N) sleep(); \Rightarrow If Buffer = full

Buffer[in] = itemp; $\quad \quad \quad$ producer \Rightarrow sleep()

in = (in + 1) mod N;

count = count + 1;

if (count == 1) wakeup(consumer); \Rightarrow when item is placed in

Buffer consumer \Rightarrow wakeup.

}

void consumer(void)

{

int itemc, out = 0;

while(1)

{

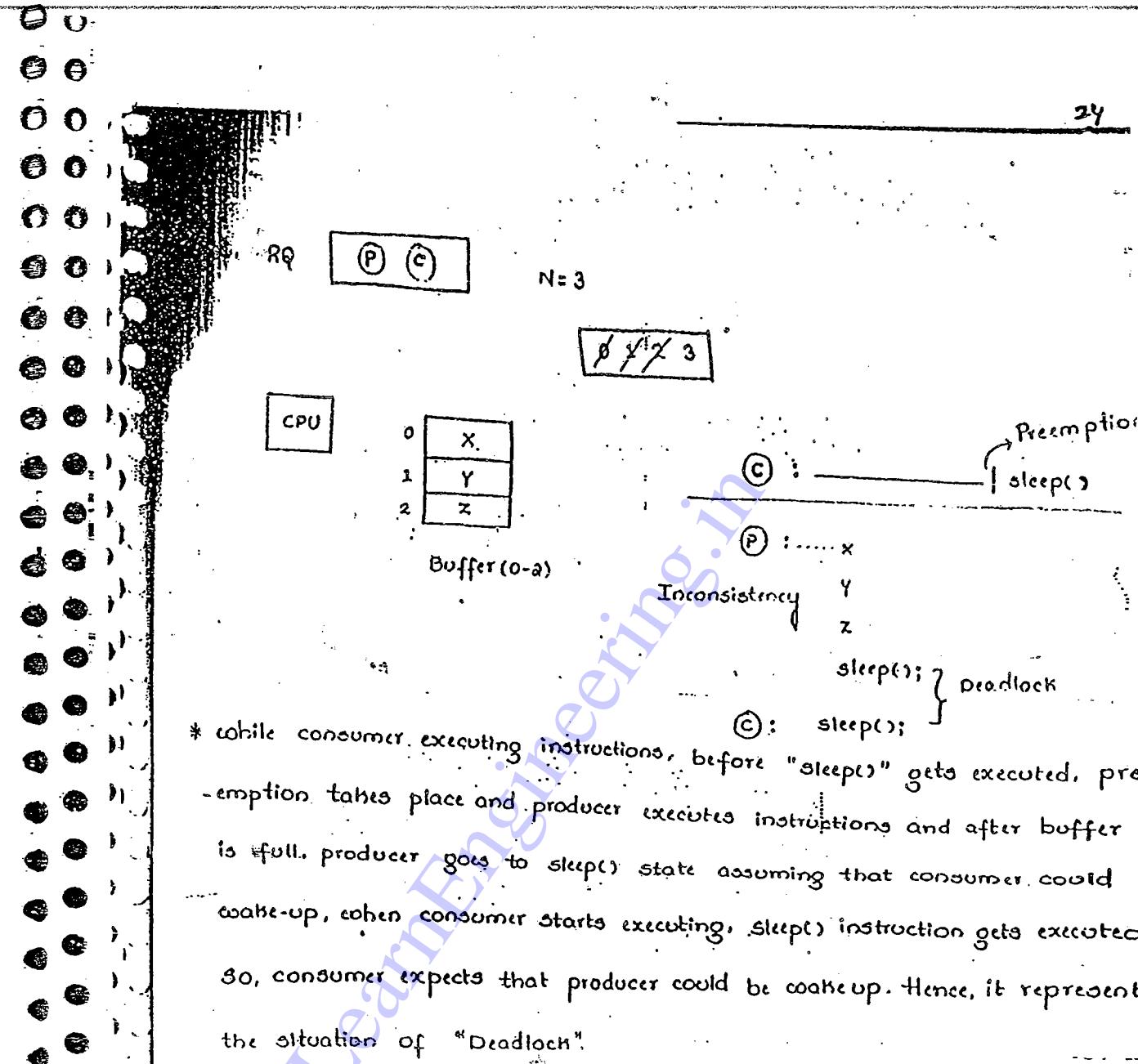
if (count == 0) sleep();

itemc = Buffer[out];

out = (out + 1) mod N;

count = count - 1;

If (count == N-1) wakeup(producer);

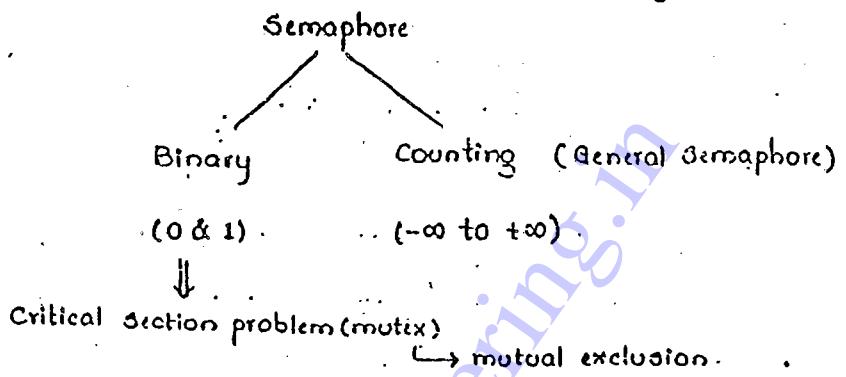


17/07/2010

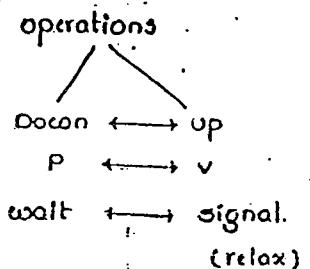
SaturdaySemaphores

- * A variable (Semaphore) that takes on integrated values.

(Integers)



- * It is an operating system resource.
- * Semaphore operations executed in kernel mode (automatically)
- * Implementation by Dijkstra (variable)



struct SEMAPHORE

{
 int value;

QueueType L;

}; // List of PCB's of those processes that get blocked while performing "down" operation unsuccessfully.

```

    SEMAPHORE s;
    {
        value = 4;
        ...
        DOWN(s);
        <c.s>
    }

```

"Semaphore" gets executed in kernel mode.

* If the value after decrementing is negative, then it is unsuccessful.

* If value is zero (or) positive then it is successful.

(+) value → Successful Down processes.

(-) value → Blocked processes.

Eg :-

s.value = 4	Down.
4 - 1 = 3	1
3 - 1 = 2	1
2 - 1 = 1	1
1 - 1 = 0	1
0 - 1 = -1	-1
-1 - 1 = -2	-2

{ 4 successful processes [(+)ve. value of counting available in stack Semaphore always indicate no. of down operations successful].
 a blocked process(es) [(unsuccessful process(es)) ⇒ waiting for semaphore].

DOWN(SEMAPHORE s)

```

    {
        s.value = s.value - 1;
        if(s.value < 0) //Unsuccessful.
    }

```

put this process (PCB).in
SL() & Block it (sleep());

}

After Blocking the process,

just it moves off (i.e., no

down operations are performed)

UP(SEMAPHORE s)

```

    {
        s.value = s.value + 1;
    }

```

if(s.value < 0)

{ Select a process from SL()
and wakeup();

} wakeup process gets into c.s but not

If s = -a perform "down" operation

After $P_2 = -2 + 1 = -1$

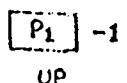
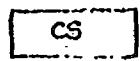
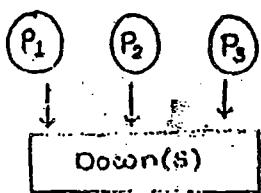
$$P_2 = -1 + 1 = 0$$

$$P_3 = 0$$

$$= 0 + 1$$

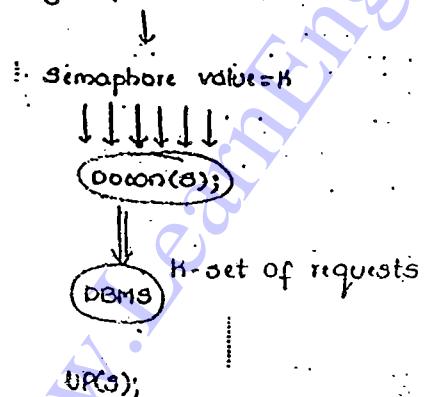
$$= 1$$

- * When we have 3 processes, semaphore value starts with 1 \Rightarrow performing down operation on it, the value becomes '0'. (i.e., the down operation is successful, & it enters into critical section). After P_1 enters into c.s, all the other processes perform down operation.
 $(P_1 = 0, P_2 = -1, P_3 = -2)$



- * Process never get blocked while performing "UP" operation and it gets blocked while performing "Down" operation.

Allowing K processes into c.s



S.Q() \rightarrow Semaphore Queue consists
list of processes

After process wakes up, don't perform any down operations, simply execute the critical section:

Eg:- $S = 10$.

16 p, av, 8p, 1v, 4p, 3v, 6p, av

$$\begin{aligned}
 S &= 10 & p \rightarrow \text{down} \\
 &= -4 & v \rightarrow \text{up} \\
 &= -9 & \\
 &= -10 & \\
 &= -9 & \\
 &= -10 & \\
 &= -16 & \\
 &= 14 &
 \end{aligned}$$

Binary (mutex) Semaphore (0/1):

```
: struct BSEMAPHORE
```

```
{
```

```
    enum value {0,1};
```

```
    Queue Type L;
```

```
}
```

```
BSEMAPHORE S;
```

```
S.value = 1;
```

```
DOWN(S);
```

```
<c.o>
```

```
DOWN (BSEMAPHORE S)
```

```
{
```

```
if (S.value == 1) // successful.
```

```
S.value = 0;
```

```
else // unsuccessful.
```

```
{ put this process (PCB)
```

```
in B.L() & Block it
```

```
Sleep();
```

```
}
```

Condition-I : If there are Blocked processes.

(or)

Condition-II : If there is a process in critical section

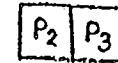
Then, value of Binary semaphore must be zero..

After performing down operation
on Bsemaphore,

if value = 0 => unsuccessful

value = 1 => successful.

(1-2)
If S=1, => P₁ = 0 (After down)



operation, semaphore
must be 1).

P₂ checks semaphore value. Since, the
value = 0, no down's operation is perf-
ormed. Simply it moves to Block state.
Similarly, by having semaphore
value, P₃ also gets blocked.

After completion of all processes, semaphore value gets incremented.

(Because chance given to newly entered process)

UP(BSEMAPHORE s)

{

If (s.L() is not empty)

{

Select a process from

s.L() and wake up()

}

{

s.value = 1;

}

}

User doesn't know no. of processes present,

in the Queue by just having semaphore

value, one can't decide no. of processes.

So, we perform operation on blocked Queue

Semaphore value = 0 \Rightarrow It represents that there is no compilation to present blocked processes.

After completion of processing all the jobs in blocked queue, increment

Semaphore value. It represents that don't release semaphore until all the

jobs are completed.

Initial semaphore = 0, no blocked processes

s = 1

ss> 10 p, 3v, 15p, 7v, 8p, av, 3p, 2v s=0

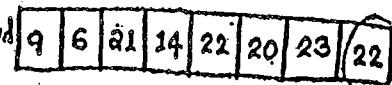
22

0	1	0	1	0	1	0	1
9	9	6	21	14	22	30	23

p = down

v = up

Blocked
process



Cases of Binary Semaphore:-

Case (i) :-

S = 1 s value becomes = 0
P(s); status = successfully.

case (ii) :

S = 1 S = 1
V(s); status = successful.

case (iii) :

S=0 S=0
P(s); status = unsuccessful.

case (iv) :

S=0 first value (initial value).
+ V(s); a = 1
 (up operation) status = successful.

Classical Interprocess Communication problems.

(1) producer-consumer :-

```
#define N 100
```

```
int Buffer[N];
```

Semaphore empty = N; // no. of empty slots in Buffer

Semaphore full = 0; // no. of full slots

Besmaphore mutex = 1; // used to ensure r/w between P & C on buffer

void producer(void)

{

int itemp, in = 0;

while(1)

{

producer_item(itemp);

|

(Blocking producer)

Down(empty); // check for Buffer is full or not.

Down(mutex)

Buffer[in] = itemp;

in = (in+1) mod n;

If Empty = 0 ; down(empty) = 0-1
= 0

Buffer = full, then block producer

Exit \Leftarrow { up(mutex); // release the critical section.
 up(full);
 consumer_item(itemp);

```

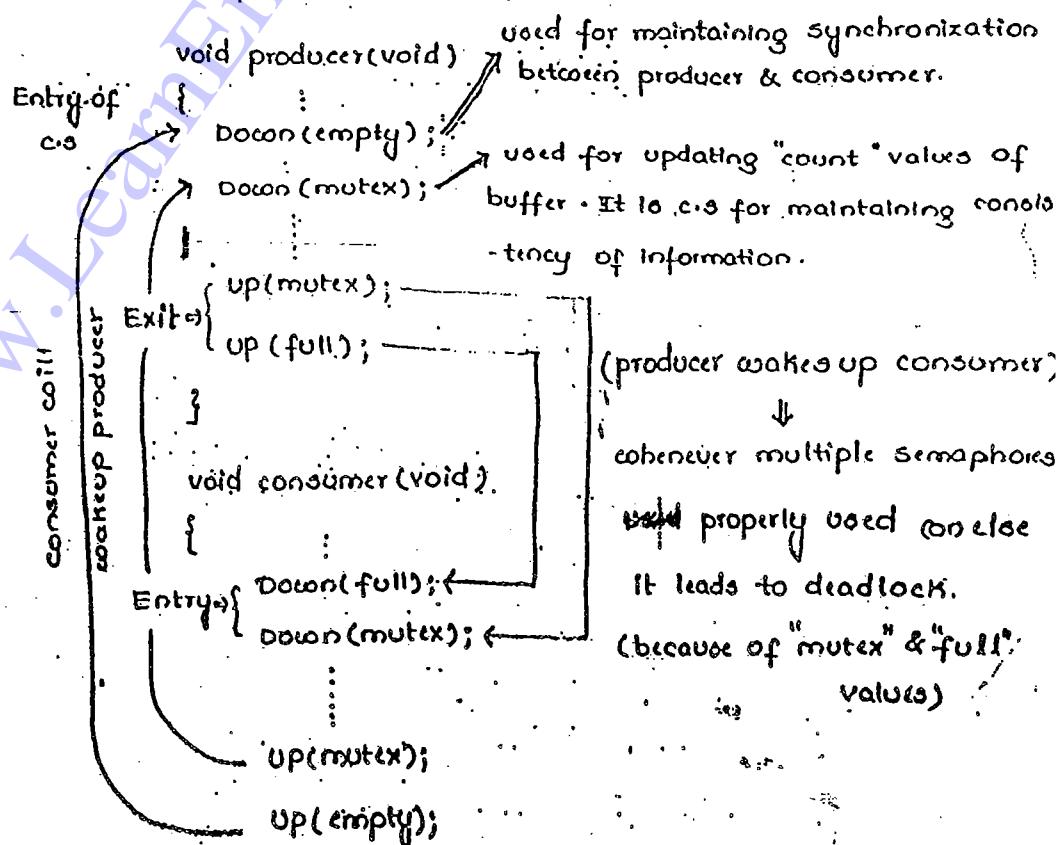
void consumer(void)
{
    int itemc, out=0;
    while(1)
    {
        consumer-item(itemc);
        Doon(full); } Interchange If
        Down(mutex); deadlock occurs to
        itemc = Buffer[out];
        out = (out+1) mod N;
        Up(mutex);
        Up(empty);
        producer-item(itemc);
    }
}

```

Enter { }

itemc = Buffer[out]; Up(full)

out = (out+1) mod N; Up(mutex).



(a) Reader-writer problem :-

(1) 1 Reader - 1 writer

(2) Multiple Readers - 1 writer

(3) Multiple Readers - Multiple writers

* we can perform multiple "Read" operations at a time.

Database enters into critical section, when there is a writer. so, at a time, only one write operation can be performed.

```
int RC=0; // Readers count
```

```
Semaphore mutex=1; // Used for mutual exclusion b/w readers
```

RC must be in c.s.

```
Semaphore db=1; // used to ensure m/e between readers & writers  
on DBMS
```

```
Void Reader(void):
```

```
{
```

```
critical
```

```
{
```

```
DOWN(mutex); // To increment readers count, check "mutex" value
```

```
RC = RC + 1;
```

```
if(RC==1)
```

```
DOWN(db);
```

```
UP(mutex);
```

// If first reader enters into DBMS, It's responsibility
to check whether any writer present in c.s
(or not) using (RC==1) condition.

```
CS < READ+DBMS >
```

```
DOWN(mutex);
```

```
RC = RC - 1;
```

if (RC == 0) :
 UP(db); ⇒ release db
 UP(mutex); ⇒ release mutex

Supposedly have counter, it's val becomes ≈ 0 , so, no reader enters the first stmt.

void writer(void)

If first writer is successful to enter into critical section, it releases the mutex b/w the readers for allowing many no. of readers.

< update - DBMS >

Again decrementing the 'Rc' count, use critical section of Readers.

(3) Dining philosopher's problem:-

Normal case :

```
void philosopher(int i)
```

sehile(s)

`thinks(f)`

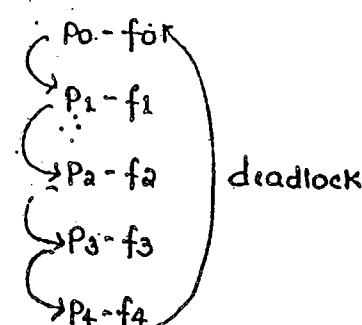
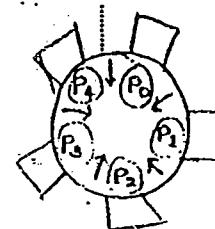
take-fork(1);

Take-fork($c_{i+2} \cdot \%N$):

`eat(i);`

put-fork(i);

put_fork((i+1) % N);



- To avoid the problem of deadlock, in this, philosopher breaks the rule :-
 - i.e., Even positions member follow one approach & odd position members follow another approach.

Actual problem :-

- Philosophers take the position on a table, they first try to take left hand side fork, if they succeed, then they take right fork. If both the forks are available, then they start eating & then put off the forks.
- But, at a time, all philosophers can't have both forks, where deadlock situation occurs.

Eg:- If P_0 first takes f_0 , P_1 takes f_1 , P_2 takes f_2 , P_3 takes f_3 , P_4 can't have its left fork because P_0 's right is P_4 's left, hence, deadlock occurs. So, no one wants to release their left forks.

State[N] // stores state of philosopher either thinking
(or) eating

Semaphore mutex = // used for releasing & taking the fork \Rightarrow it is C.S.

Semaphore s[N] = {0};

```

#define N 5

#define THINKING 0
#define HUNGRY 1
#define EATING 2

#define LEFT (i+N-1) % N
#define RIGHT (i+1) % N

int state[N] = {0}; // Either thinking (0), hungry (1) or eating.

Semaphore mutex = 1; // To handle forks either by taking (0) or releasing

```

```
void philosopher(int i)
```

```
{ cobile(i)
```

```
{ Think(i); // initial state
```

```
take-forks(i);
```

```
eat();
```

```
put-forks(i);
```

```
void take-forks(int i)
```

```
Down(mutex); // taking forks.
```

```
state[i] = HUNGRY;
```

```
test[i];
```

```
UPC(mutex);
```

```
DOWN(sem); // fails to take forks is so blocked.
```

```
void put-forks(int i)
{
    DOWN(mutex);
    State[i] = THINKING;
    test(LEFT); } state again changed.
    test(RIGHT); } to THINKING.
    UP(mutex);
}

void test(int i) // testing forks are available or not
{
    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] == EATING)
    {
        state[i] = EATING;
        UP(s[i]); // if Eating is completed & set s[i]=1
    }
}
```

Concurrency & Concurrent programming

Concurrency

Real

physical

concurrency

Pseudo

concurrency

(achieved by processor)

(achieved through multi processor,
array processor, vector processor)

Precedence Graph ~~matrix~~ :-

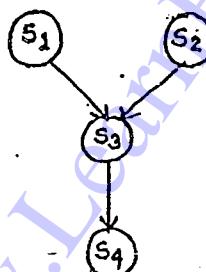
- * It indicates which statements executes concurrently.

$$S_1 : a = b + c;$$

$$S_2 : d = e + f;$$

$$S_3 : k = a + d;$$

$$S_4 : l = k + m;$$



There is no precedence.

Any statement can be executed first.

Concurrency conditions :-

$$S_1 : a = b + c;$$

$$S_2 : d = e + f;$$

$$S_3 : k = a + d;$$

$$S_4 : l = k + m;$$

Read set \rightarrow The values read into equation

$$R(S_2) = \{b, c\}.$$

Write set \rightarrow update value in equation

$$W(S_1) \in \{a\}$$

$a + \dots + b + c \dots$ all a, b, c updated so,

$$R(S) = W(S) = \{a, b, c\}.$$

- (1) $R(s_i) \cap W(s_j) = \emptyset$ if Reading & writing set of two different processes equal, then we can't perform
- (2) $W(s_i) \cap R(s_j) = \emptyset$ (concurrent execution) at a time.

$$a = c * d$$

$$a = l * m \text{ (not supported)}$$

If (count ≠ 0) → all processes are not completed.

fork & Join :

fork 1;

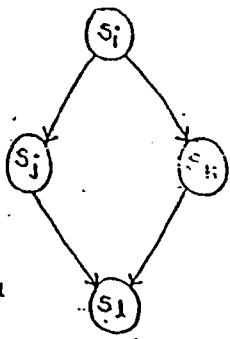
s_j ; // after completion of s_j go to s_i .

go to t_1

$t_1: s_k$ // it executes immediate instructions t_1

$t_1: \text{Join count};$ // in

s_1



Join (int count)

{

count = count - 1;

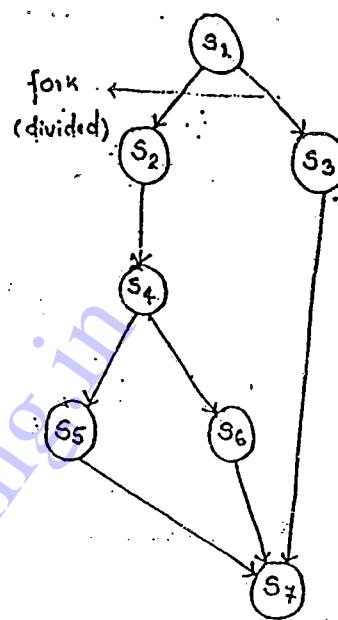
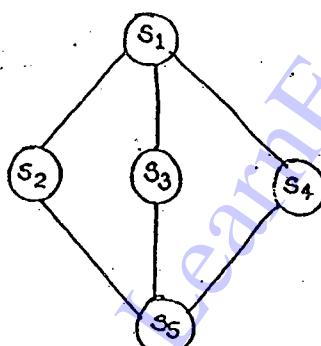
if (count ≠ 0) // all the processes are not

exit; completed. So, there is

}

no execution of s_1 .

Count = 3

S₁;fork L₁ (S₁ followed by fork)S₂;S₄;fork L₂S₅;L₁: S₃go to L₃L₂: S₆L₂: Join countS₇:

Count = 3

S₁;fork L₁S₂;go to L₂go to k₁L₂: fork L₂S₃;go to L₂L₂: S₄;go to L₃L₃: Join count (check for
all processes
either comple-
ted or not).

S₂, S₃, S₄ are independent processes so, they can be accessed in any way. But S₅ is purely dependent on S₂, S₃ and S₄.

Eg:- If S₂, S₃ accessed & completed, and S₄ needed to be accessed now, even though there is S₅ to be accessed, we won't access. After the completion of all the processes at the upper level (S₂, S₃, S₄), then only the last level process (S₅) must be computed. For such checking, "Join".

condition is used which counts the completed functions that are needed, used for completion of s_5 .

Parbegin - Parend & co-begin - coind:

$\text{begin} \{$

$s_1;$

$s_2;$

$s_3;$

$\text{end} \}$

\Rightarrow



$s_0;$
 $\text{parbegin} \{$

$s_1;$

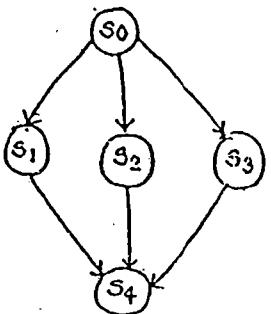
$s_2;$

$s_3;$

parend

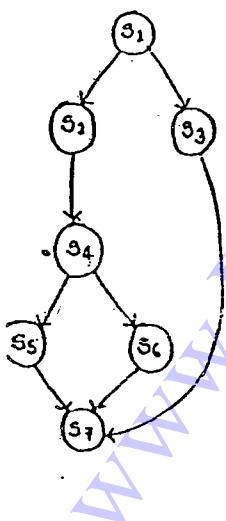
$s_4;$

\Rightarrow They support
parallelism

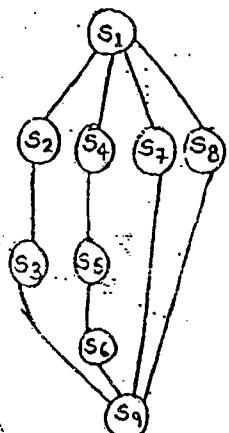


$\text{Begin-end} \Rightarrow$ represents sequential actions

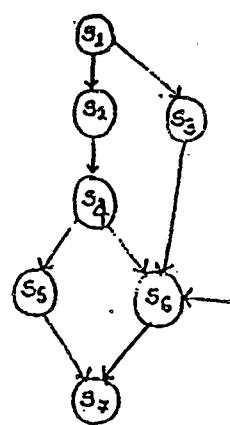
$\text{Parbegin - Parend} \Rightarrow$ represents parallel actions.



$s_1;$
 Parbegin
 begin
 $s_2;$
 $s_4;$
 Parbegin
 $s_5;$
 $s_6;$
 Parend
 end
 $s_3;$
 Parend
 $s_7;$



$s_1;$
 Parbegin
 begin
 $s_2;$
 $s_3;$
 end
 begin
 $s_4;$
 $s_5;$
 $s_6;$
 end
 $s_7;$
 $s_8;$
 Parend
 $s_9;$



For this graph, we cannot write a code because for execution of s_6 , we need s_4 and s_3 . With the help of code, we cannot impose such condition. We can overcome this problem by using "Semaphore".

Parbegin - Parend with semaphore :-

Semaphore a,b,c,d,e,f,g = {0};

Parbegin

begin s_1 ;

Parbegin

$v(a)$;

$v(b)$;

Parend

end

begin $p(a); s_2; s_4;$

Parbegin

$v(c)$;

$v(d)$;

Parend

end

begin $p(b); s_3;$

$v(e)$;

end

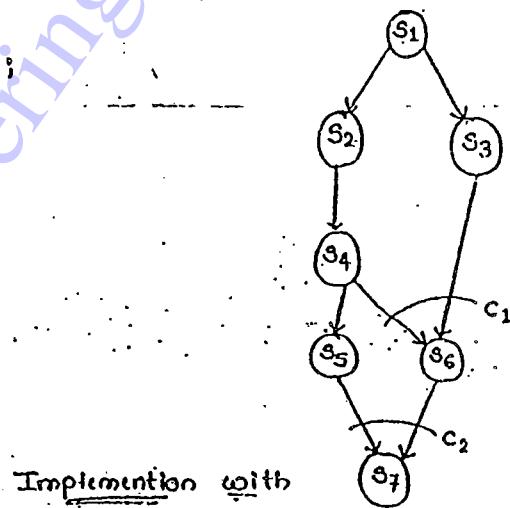
begin $p(c); s_5;$

$v(f)$;

end

begin $p(d); p(e); s_6; v(g);$ begin $p(f); p(g); s_7;$ end.

Parend



Implementation with fork & Join :-

$c_1 = 2; c_2 = 2; l_1: s_3;$

$s_1;$ $l_2: \text{Join } c_1;$

$\text{fork } l_1;$ $s_6;$

$s_2;$ $l_3: \text{Join } c_2;$

$s_4;$ $s_7;$

$\text{fork } l_2;$ s_5

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$s_7;$

$g \text{o to } l_3;$

$l_1: s_3;$

$l_2: \text{Join } c_1;$

$l_3: \text{Join } c_2;$

$s_7;$

$g \text{o to } l_3;$

$s_7;$

```
Eg:- void P(void)
{
    A;
    B;
    C;
}
void Q(void)
{
    D;
    E;
}
main()
{
    Parbegin()
    P();
    Q();
    Parenend()
}
```

What are the possible outputs for the above code:

(A) A,B,C,D,E

(B) D,E,A,B,C

(C) A,B,C,E

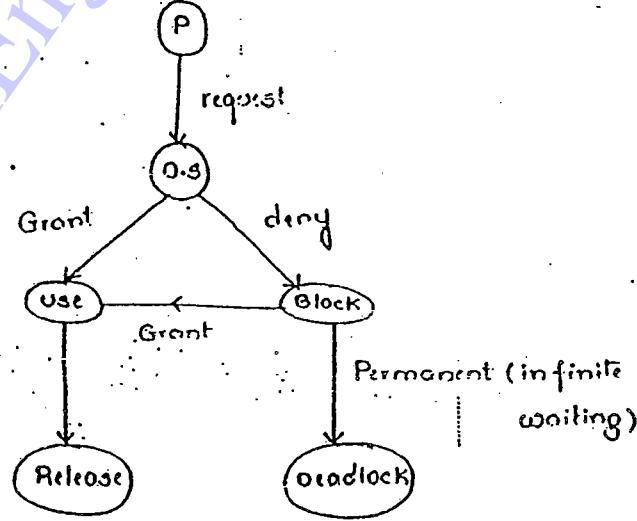
(D) D,A,C,E,B

(E) E,C,B,D,A

A,B,C and D,E are sequential actions. C,B cannot be happen so, only first three are possible outputs.

DEADLOCK (Lockingup)

- * Two (or) more processes are said to be involved in deadlock, iff they wait for the happening of an event, which would never happen.
- * System model (for avoiding deadlocks)
 - * 'n' processes (P_1, \dots, P_n)
 - * 'm' resources
- * Starvation \Rightarrow long waiting.
- * Deadlock \Rightarrow Infinite waiting.



- * Processes make a request for resources. If the resources are available, then it is granted by O.S. If it takes some time for resource allocation, then process gets blocked. Blocked processes requests for the resources, after certain time, and if again not granted, then the process moves to deadlock state.

A deadlock situation can arise if the following four conditions :-

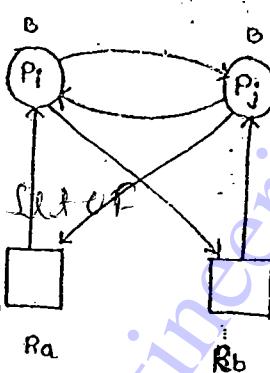
- (1) Failure of mutual exclusion. Hold simultaneously in the system.
- (2) Hold & wait.
- (3) No-preemption (process won't release their resources).

Deadlock Condition

- (4) circular wait.

① If a process type only one instance, then a cycle then must be deadlock.

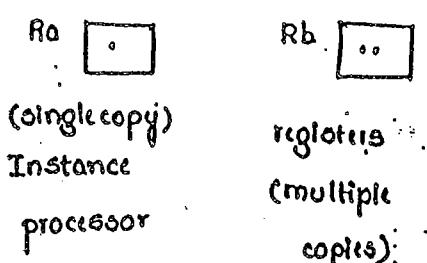
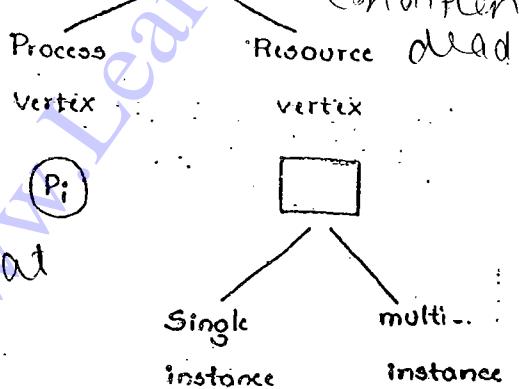
② If cycle involve only a set of outflow type and each has single instance then deadlock occurred.

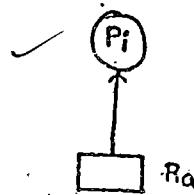
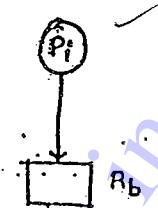


③ Each process involved in the cycle is deadlock.

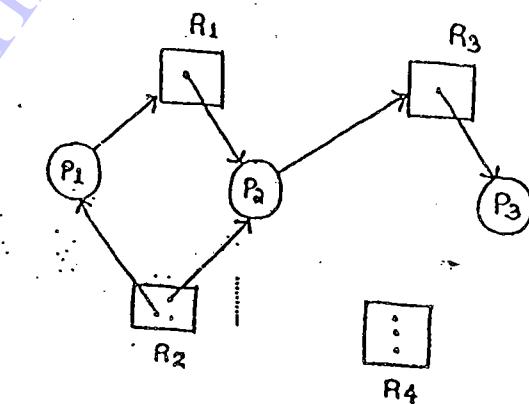
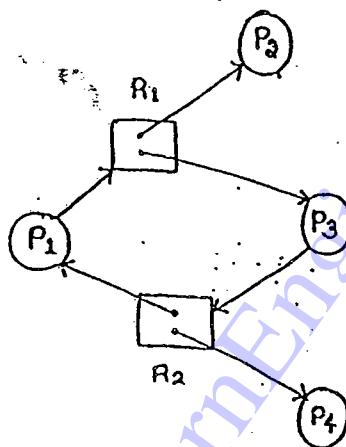
④ Cycle in the graph RAG both are necessary and sufficient condition for the existence of deadlock

If the process has multiple instances then the cycle is necessary but not sufficient



Edges :Assign edgeRequest edge

$R_a \rightarrow P_i$ allocating
a instances of
 R_a .



P_1 gets blocked, since it requests ' R_1 ',
which inturn R_1 gets accessed by
 P_2 . Similarly, P_3 also gets blocked.

- * If RAG has single instance and multiple instance.
 - * In multiple instances, there is a chance of occurring deadlock, because they form cycles (some times).
 - * Single instances may also cause deadlock, if the processes accessed in the cycle form.
- If the graph contains no cycle then no process in the system is deadlock. If the graph contains a cycle then a deadlock may exist.

method for handling deadlock! how way

For More Visit : www.LearnEngineering.in

① we can use a protocol to prevent or avoid deadlock ensuring that the system will never arrive in the deadlock.

② Strategies for deadlock handling :-

we can allow a system to enter a deadlock state detect it and recover.

③ we can ignore the problem altogether. And prevent Type-1 the deadlock never occurs in system.

Type-3

This solution

is used by most operating systems

* Deadlock prevention

* Deadlock avoidance

(Banker's algorithm)

No deadlock happens
occurs.

* Deadlock

Detection and

Deadlock

Recovery

Deadlock happens

* Deadlock Ignorance

(Ostrich Algorithm)

↓
(Deserts)

Deadlocks prevention :-

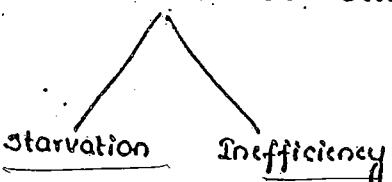
* Try to dissatisfy one / more of the four necessary conditions.

(1) mutual exclusion :- Not required for shared resource may hold for non sharable resource

(2) ! (Hold & wait) :-

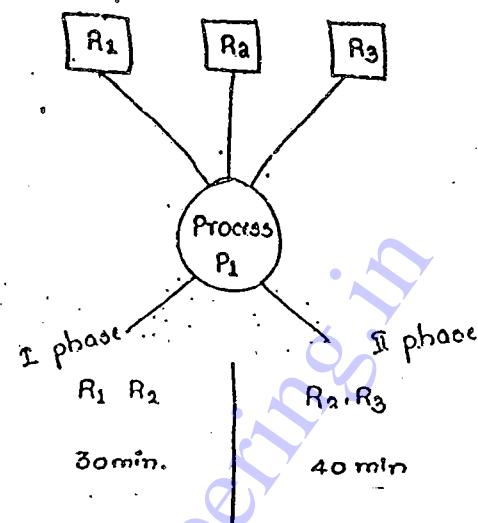
Either "hold" (or) "wait" (but not both)

* Requests & deallocation of all the resources done between the process commencements :-



mutual Exclusion :-

* Holding only one process within the critical section.



Consider,

- * A process ' p_1 ' which needs to access the resources ' R_1 ', ' R_2 ' and ' R_3 '. If a resource ' R_1 ' and ' R_2 ' are accessed and another time, again ' R_2 ' needs to be accessed. So, the following situation holds :-
Hold & wait :-
- * Process ' p_1 ' in first phase, requires R_1 and R_2 resources and the time span allotted to phase-I is 30min. to complete.
- * After the completion of phase-I, process ' p_1 ' releases the resource ' R_1 ' but holds the resource ' R_2 ' (because it again requires the same resource at phase-II also), and requests for the resource " R_3 ". This situation is considered as "Hold & wait".
- * Inorder to avoid this problem, we have two types of solutions :-

1st Approach :-

- * Process, in phase-I, requests all the resources, whatever it needs.

$$\text{Eg.:- } P_1(R_1, R_2, R_3)$$

- * They doesn't wait for all the resources to be accessed.

2nd Approach :-

- * Release all the resources before making a new request.

$$\text{Phase - I} \rightarrow P_1(R_1, R_2)$$

$$\text{Phase - II} \rightarrow P_1(R_0, R_3)$$

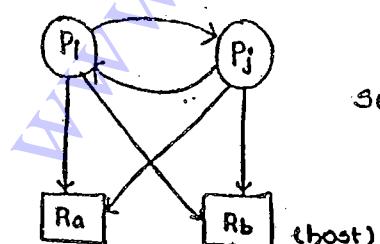
- * After the completion of phase-I, release all the resources even though, it needs the same resource at phase-II.

- * Then, again in phase-II, the resources are requested newly (if already used in phase-I).

(c) No pre-emption :-

- * Allocates the processes to get preempted.

Self forcefully :



Self release \Rightarrow It requests the resources that are not available, so that process releases other resources that are already available

- Consider, P_j , which already accessed ' R_a ', it requests for R_b , but R_b is not available, at that time. So, ' P_i ' releases ' R_a ' resource (in the mean, it may be useful for others) after completion of other process, it will execute.

22/07/2010
Thurday

Prevention (restrictive):

(c) Circular wait :

"Total order relation on all process for requesting of resources."

Incl dec.

P₁-8

P₂-4

P₃-12

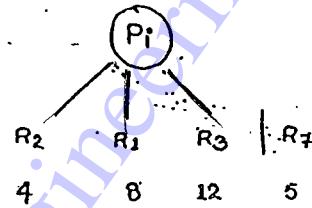
P₄-18

P₅-25

P₆-9.

P₇-5

f(R) → I'



Starvation occurs when there is a release on "

p_i requests in any order

(i.e., either in incl dec).

If no order is followed,

any sequence is followed, p_i must release all resources.

Avoidance :-

(Banker's Algorithm):

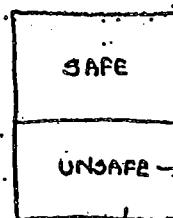
* Safety Algorithm

* Resource - Request.

Objective :-

* Obtain a system always in a safe mode.

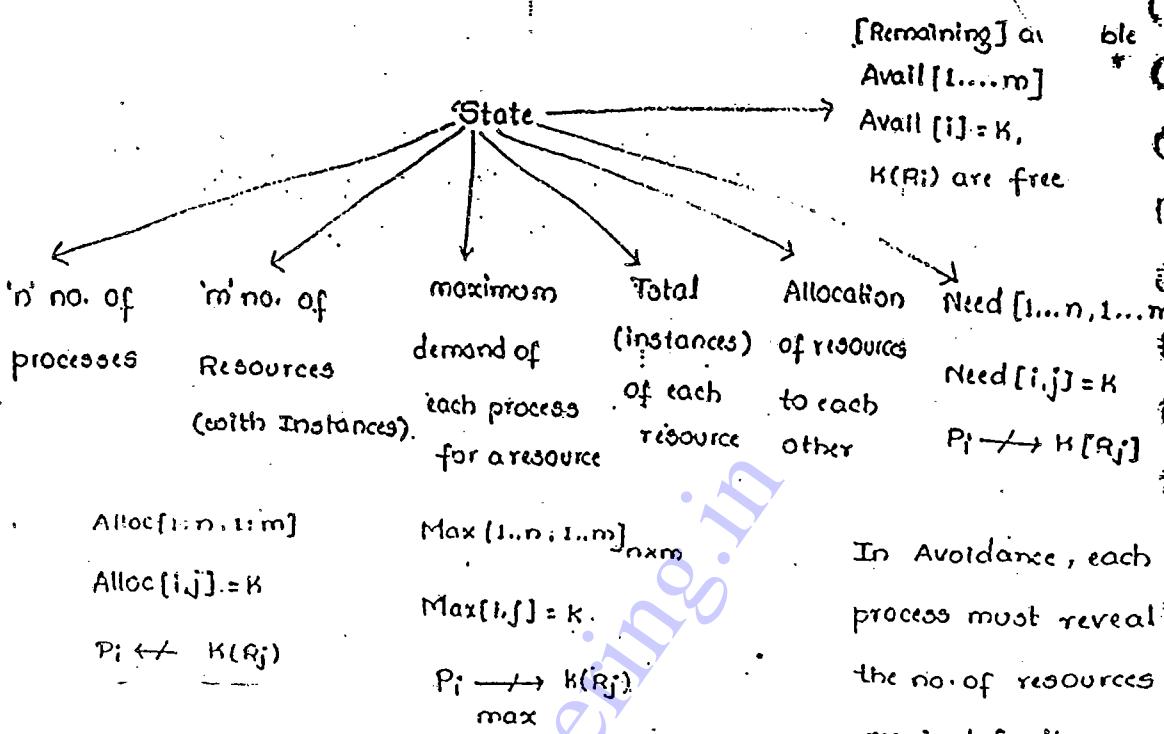
* It is an "A priori Info" algorithm. It is based on state of the system



no deadlock

occurs.

→ Banker algo is more efficient than the
Jelgwile allocation graph



Eg : $n=5$ (P_1 to P_5).

$$m=1, R = 28 \text{ (Total)} \quad Need = max - Alloc.$$

n	R		(Need)		(Avail)	
	(max)	(Alloc)	R _i	R _j	R _i	R _j
P_1	10	5	5		3	
P_2	20	10	10	4	5	
P_3	8	1	2		6	
P_4	8	4	4		13	
P_5	12	5	7		18	
						28

$T_0 : \langle P_2, P_4, P_3, P_1, P_2 \rangle$

safe-sequence

Avail = current avail + allocation.

* System is said to be **SAFE**, iff the needs of all the processes satisfied with the available resources in some (on the order of) (is called **SAFE sequence**) otherwise, the system is said to be **UNSAFE**.

But not

really a deadlock

to indicate

warning

- ① mxn^2 operation to check if state is safe or not
- ② detecting a cycle in the REG sequence n^2 operation

$$n=5, \quad m=3 \quad (A,B,C) = (10,5,7)$$

P	max			Alloc			need			Avail			Alloc (a+b+c)
	A	B	C	A	B	C	A	B	C	A	B	C	
P ₀	7	5	3	0	1	0	7	4	3	3	2	2	10 - 7 = 3
P ₁	3	2	2	2	0	0	1	2	0	5	3	2	5 - 2 = 3
P ₂	9	0	2	0	0	2	6	0	0	7	4	3	7 - 6 = 1
P ₃	2	2	2	2	1	1	0	1	1	7	4	5	
P ₄	4	0	3	0	0	2	4	3	1	7	5	6	
										10	5	7	

T₀ : $\langle P_1, P_3, P_4, P_0, P_2 \rangle \Rightarrow \text{safe.}$

Initial availability = (5,3,2), O.S have enough resources to grant to P₂. but it first calculates any problem if user gives this resource, It runs the "Safety algorithm" in this situation.

T₁ : Resource Request algorithm :-

"Request by the resources are granted or not" is being identified in this algorithm.

T₁ : P₂ : $\rightarrow (1,0,2) \text{ (Req 2)} \rightarrow \langle P_1, P_3, P_4, P_0, P_2 \rangle \Rightarrow \text{safe.}$

T₁ : P₄ : $\rightarrow (3,3,0) \text{ (Req 4)} \rightarrow \text{Blocked.}$

It is blocked because it doesn't satisfy the Availability.

$T_1 : \langle 0,0,0 \rangle \rightarrow P_0$

n	max			Alloc	need	Avail
	A	B	C			
P ₀	7	6	3	0 1 0	7 4 9	2 8 0
P ₁	3	2	2	3 0 2	0 2 0	
P ₂	9	0	2	3 0 2	6 0 0	
P ₃	2	2	2	0 1 1	0 1 1	
P ₄	4	3	3	0 0 2	4 3 1	

Resource Request Algorithm :-

1. Request \leq Available

P₁ $\rightarrow \langle 1,0,2 \rangle$ request

2. Request \leq need

P₄ $\rightarrow \langle 3,3,0 \rangle$

3. (Assume request is satisfied)

P₀ $\rightarrow \langle 0,0,0 \rangle$

$$\text{Available} = \text{Available} - \text{request}$$

$$\text{Allocation} = \text{Allocation} + \text{request}$$

$$\text{Need} = \text{Need} - \text{Request}$$

4. If result is safe, allocate resource.

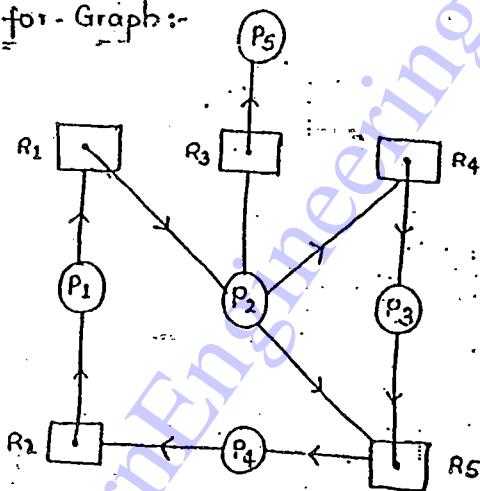
III Deadlock Detection & Recovery :

occurs when system performance is low.

If many blocked processes are present.

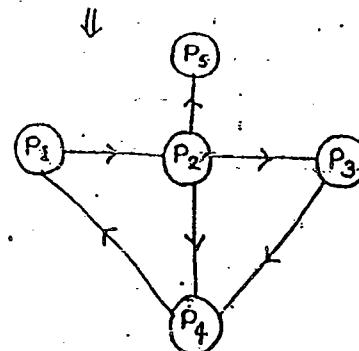
(1) Single Instance of a resource :-

wait-for-Graph :-



Among 5 processes, 4 are blocked. So, we apply deadlock detection algorithm.

Cycle
Detection
Algorithm
is used



$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$

cycle \Rightarrow Deadlock

II multi-Instance of a resource :-

$$n=5, m=3, (A, B, C) = (7, 8, 6)$$

T ₀	Alloc	Req	req is again processed after allocation		
			A B C	A B C	A B C
P ₀	0 1 0	0 0 0	0 0 0	7 - 7 = 0	
P ₁	2 0 0	2 0 2	0 1 0	8 - 8 = 0	
P ₂	3 0 3	0 0 0	3 1 3	6 - 6 = 0	Initially, resource availability
P ₃	2 1 1	1 0 0	5 2 4		is '000' but P ₂ requests 001.
P ₄	0 0 2	0 0 2	5 2 6		It can't satisfy O.S.
			7 2 6		P ₁ → P ₄ ⇒ Input to recovery module.

To check safe mode : {P₀, P₂, P₃, P₄, P₁}

T₅ : P₂ → <0, 0, 1>

<P₀, ..., X Deadlock

P₁ - P₄

↓ recovery

Recovery

Process

Resource

Termination

pre-emption

Kill-All

↓ with

(restart all the
processes)

Roll back

(disadv: starvation)

repeatedly run

the algorithm of

deadlock detection

after kill of each process

Disadvantages:

Kill-All :-

Adv :-

* No need for detection again.

* Again, it must be started from beginning

... To overcome, this, we have "Killing

graciously".

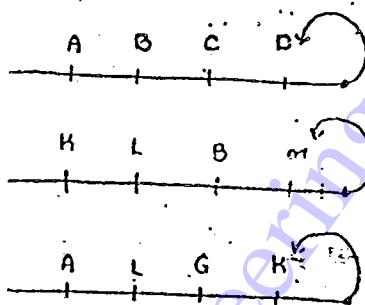
One-at-a-time :-

Adv :-

Disadv:-

* Save the other processes to resume. * Need for detection algorithm again

Resource pre-emption :-



Roll-back :

Repeatedly, a process is detected to have deadlock and re-start then the starvation occurs.

Deadlock handling strategies are not used on desktop systems and "Deadlock Ignorance" is considered :-

* Robustness: not important tasks

* follow-on strategic algorithm (in missile, real time, Industrial 0.9)

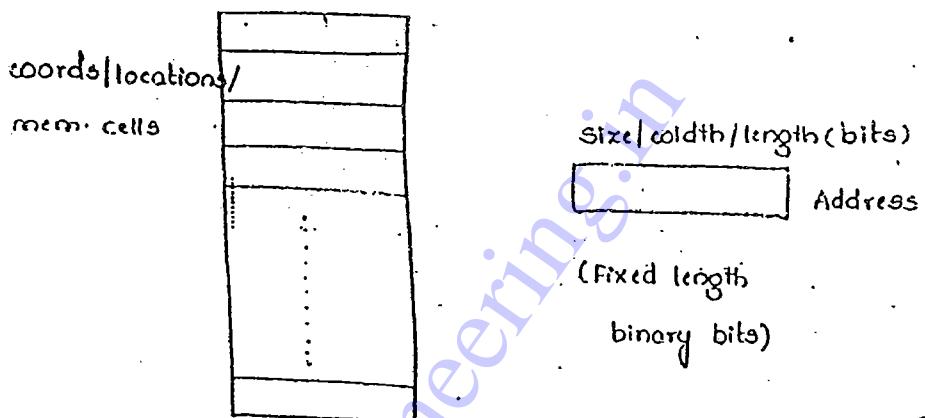
The strategies are used in real time systems where there is no single bit errors.

4.
follows Deadlock
prevention

Memory Management

Abstract view of memory

(1-D Array of words)

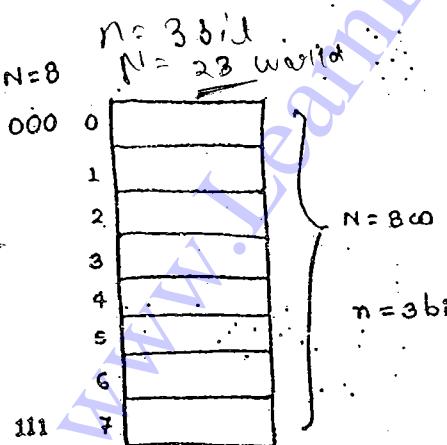


WORD

$N = \text{Total no. of words in memory}$ (Capacity of memory)

$n = \text{Address (in bits)}$

$m = \text{width / size (bits) of words}$



$$\begin{aligned} 2^{10} &= 1KB \\ 2^{20} &= 1MB \\ 2^{30} &= 1GB \\ 2^{40} &= 1TB \end{aligned}$$

$$N = 8 \text{ words} = 2^n = 2^6$$

$$\downarrow$$

$$n = 6 \text{ bits } (2^6)$$

$$N = 512 \text{ Mwords}$$

$$\downarrow$$

$$= 2^9 + 00$$

$$= 2^9 \text{ bits}$$

$$N = 4 \text{ GB}$$

$$\downarrow$$

$$n = 2^9 + 30$$

$$= 30 \text{ bits} \Rightarrow 2^{32} = 4 \text{ GB}$$

$$N = 2 \text{ GB}$$

$$n = 30 + 1 = 31 \text{ bits}$$

$$N = 512 \text{ GB}$$

$$n = 30 + 9$$

$$= 39 \text{ bits}$$

$$2^6 = 32 \text{ M}$$

$$2^7 = 64$$

$$2^8 = 128$$

$$2^9 = 256$$

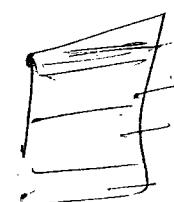
$$2^{10} = 512$$

$$2^{10} = 1024 \approx 10^3 = 1\text{K}$$

$$2^{20} = 1024 * 1024 \approx 10^6 = 1\text{M}$$

$$2^{30} \approx 10^9 = 1\text{G}$$

$$2^{40} \approx 10^{12} = 1\text{T}$$



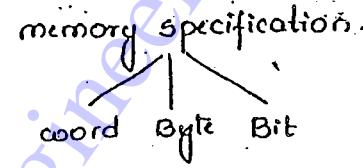
$$N = 2^n$$

$$512 \Rightarrow 2^9$$

$$n = \lceil \log_2 N \rceil$$

$$\boxed{N = 2^n \text{ words}}$$

$$\boxed{n = \lceil \log_2 N \rceil}$$



$$n = 23 \text{ bits.}$$

$$m = 16 \text{ bits } (1 \text{ word} = 2 \text{ bytes}) = \underline{\underline{16 \text{ bits}}}$$

	capacity	Address
words	8 MW	<u>$n = 23 \text{ bits}$</u>
Bytes	$8 \text{ M} \times 2 \text{B}$ = 16 MB	<u>$n = 24 \text{ bits}$</u>
Bits	$16 \text{ M} \times 8 \text{ bits}$ = 128 Mbits	<u>$n = 27 \text{ bits}$</u>

$$\begin{aligned}
 8 &= 2^3 \text{ MW} \\
 &\downarrow \\
 3 + 20 &\Rightarrow 23. \\
 16 &= 2^4 \text{ MB} \\
 &= 4 + 20 = 24 \\
 128 &= 2^7 \text{ mbit} \\
 &\approx 7 + 20 = 27
 \end{aligned}$$

$$N = 256 \text{ MH}$$

$$m = 32 \text{ bits } (1 \text{ word} = 4 \text{ bytes}) \Rightarrow 256 \text{ M} \times 4 \text{ B} \Rightarrow 1 \text{ G.}$$

	capacity	Address
words	256 MH	<u>$n = 28 \text{ bits}$</u>
Bytes	$256 \text{ M} \times 4 \text{ B}$ = 1G	<u>$n = 30 \text{ bits}$</u>
bits	8G bits	<u>$n = 33 \text{ bits}$</u>

$$\begin{aligned}
 256 &= 2^8 \text{ MH} \\
 &\downarrow \\
 8 + 20 &\Rightarrow 28. \\
 1 \text{ G} &= 2^{30} \text{ B} \\
 &\downarrow \\
 30 + 20 &\Rightarrow 50. \\
 8 \text{ G} &= 2^{33} \text{ bits}
 \end{aligned}$$

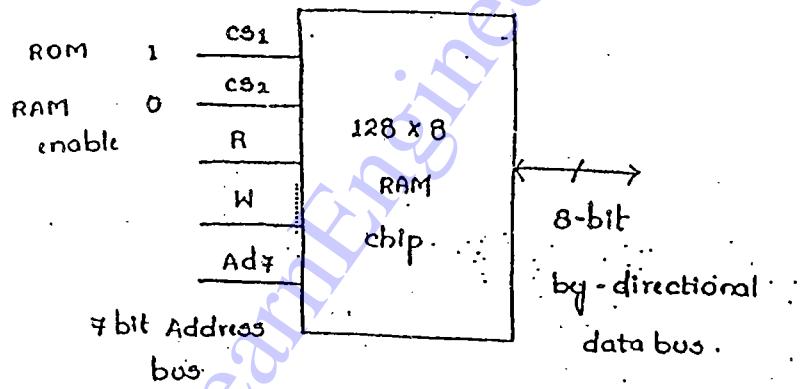
$$N = 256 \text{ Gbits}$$

$$m = 64 \text{ bits} \quad (\text{1 coord} = 8 \text{ bytes}) \Rightarrow 256 \text{ G} \times 8 \text{ B}$$

$$\begin{aligned} & 2^8 \cdot 2^{30} \times 2^3 \times 2^3 \\ & \frac{2^{38}}{2^6} = 2^{32} = 4 \text{ GH} \end{aligned}$$

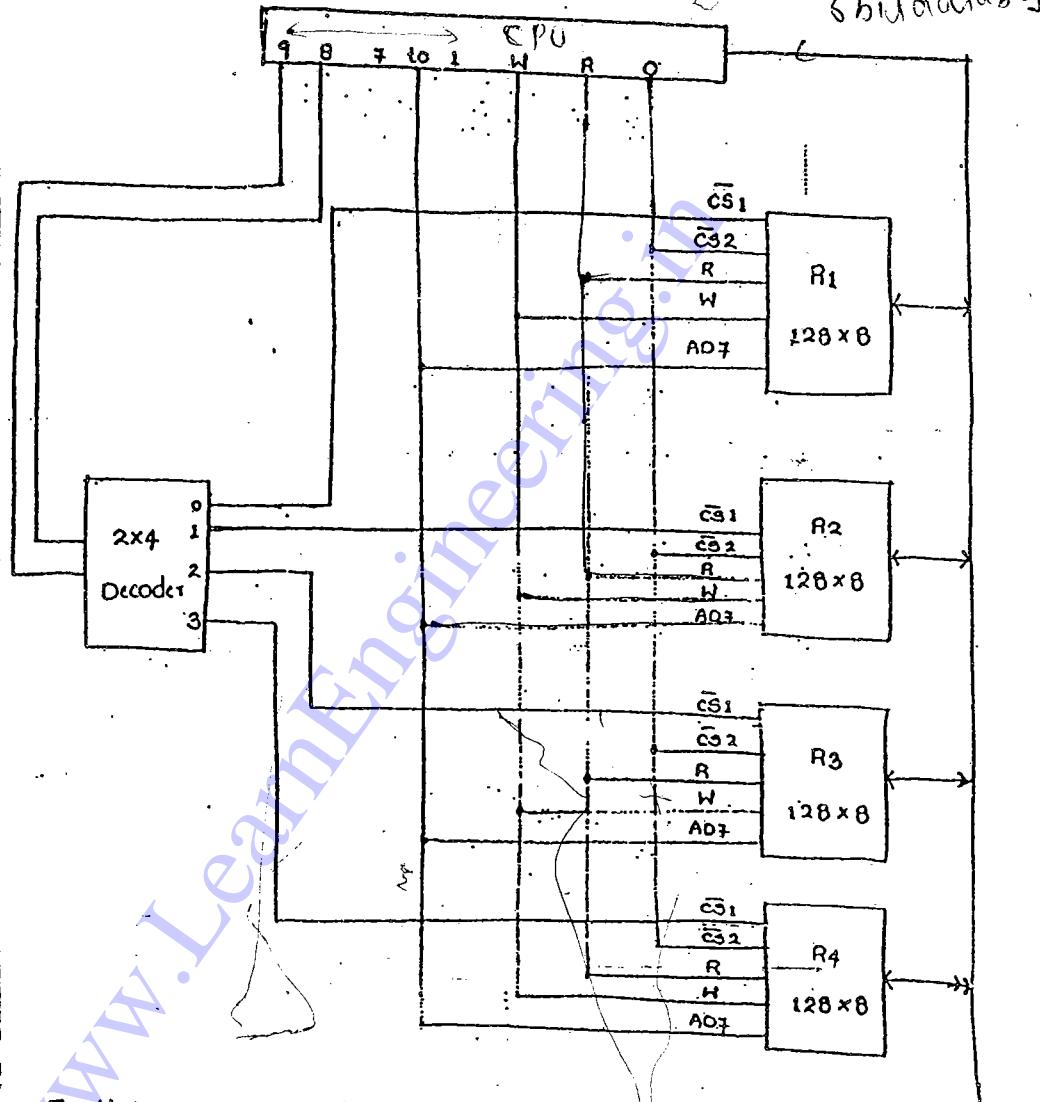
	capacity	Address
coords	4GW	32 bits
Bytes	$4G \times 8B$ $= 32 \text{ GB}$	36 bits
Bits	256 Gbits	38 bits

RAM chip organisation :-



$$\text{No. of chips} = \frac{\text{memory desired}}{\text{chip size}}$$

$$\frac{512 \text{ MB}}{128 \text{ MB}} = \frac{2^9}{2^7} = 2^2 = 4$$



I Using 128 x 8 RAM

$$16 \text{ KB} = ?$$

↓ no. of words
increasing

(i) chips = 128

$$\frac{16 \text{ KB}}{128} = \frac{2^{14}}{2^7} = 2^7 = 128$$

(ii) n = 14 bits

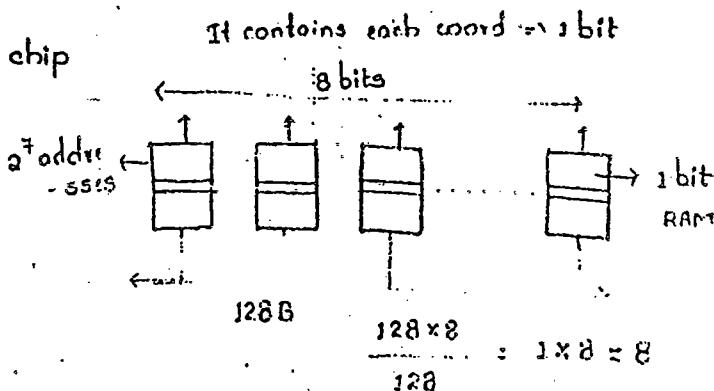
(iii) Decoder = 4×128

II Using 128×1 RAM chip

128 bytes

(i) chips = 8

(ii) $n = 7$ bits



since, memory and data size are same, there is no need of decoder.

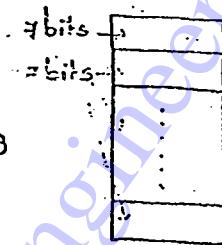
III Using 128×1 RAM chip

16 KB

(i) chips = 1024

(ii) $n = 14$ bits

(iii) Decoder = $2^7 \times 128$



128×1 RAM

$128 \times 8 \text{ rows} = 1024 \text{ bits}$

$$\frac{16 \text{ KB}}{128 \times 1} = \frac{2^4 \times 2^{10} \text{ B}}{2^7} = 1024.$$

$$= 2^7 = 128 \times 8$$

= 1024

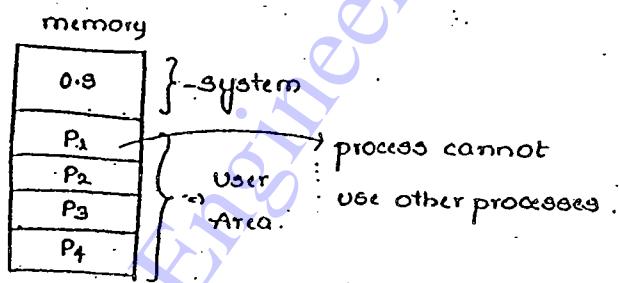
Memory Manager

Functions :-

- * Allocation
- * Protection
- * Free Space management
- * Deallocation

Goals :-

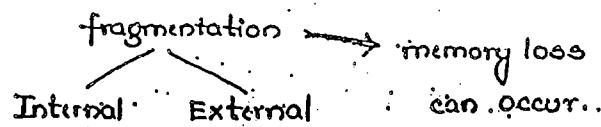
- * Efficient utilization of space.
- * Run larger program in smaller memory



Goals :-

(1) Space utilization:-

Keeping fragmentation at each level, so that there is a chance of effective utilization of space.



(2) Run larger program in smaller memory

Program = 100 KB (Disk)



Using virtual memory / overlays.

Ex:- Run the data within pendrive (3GB=60KB)

Memory management

Techniques

Contiguous (C)

* centralized

program, as a unit is completely stored

* Overlays

* partitioning

Non-contiguous (NC)

* Decentralized

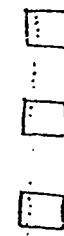
program portions are distributed in memory

* paging

* Segmentation

* Segmentation - paging

* virtual memory



24/07/2010

saturday

(1) Overlays (larger programs run in smaller space):

2-pass Assembler

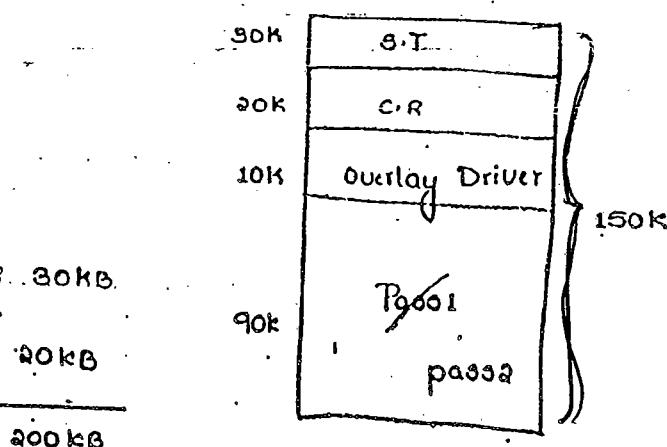
pass 1 : size - 90 KB

pass 2 : size - 80 KB

Symbol Table

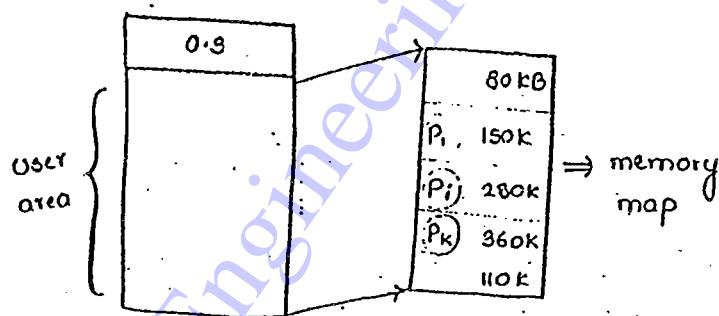
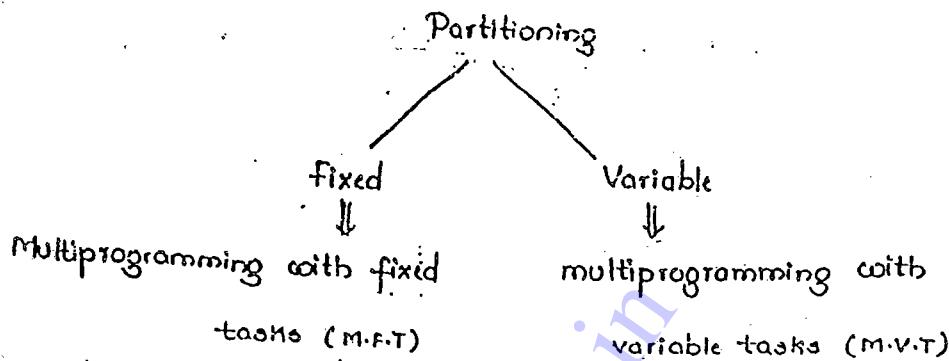
for construction of passes : 30KB

C.R (Common Ruptcs) : 20 KB



loading programs \Rightarrow overlay driver

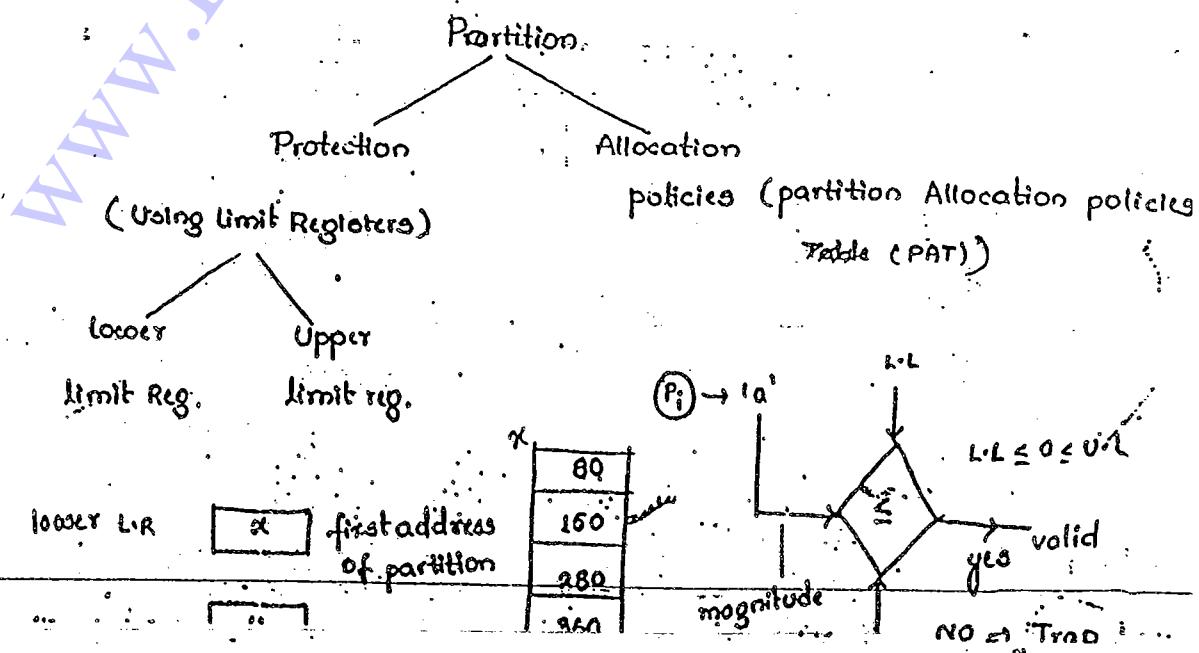
(2) partitioning :-



fixed (M.F.T) \rightarrow no. of partitions are fixed and of different sizes

Since

$1 \text{ partition} = 1 \text{ process}$



Eg: $P_{req} = 100 \text{ KB}$

80
150
280
360
50

Partition Allocation policies :-

(i) First fit (F.F) :-

- * Allocate the process in the first free begin of the partition.

To include $P_{req} = 100 \text{ KB}$, the partition of $P = 150 \text{ KB}$ is selected as it is the first partition of required size.

(ii) Best fit (B.F) :-

- * Smallest begin of free partition.
 - * Best available fit
 - * Best fit only.

Best available fit :-

Among available partitions, select the smallest best available.

Best fit only :-

It selects for best fit only, if it is not available, it waits for that partition, and after available it takes.

Next fit (N.F) :-

Same as first fit, but it doesn't start from first partition, but it starts from the place where the last partition ended up (i.e., last process's partition ended point).

Advantages :-

- * Searching becomes faster than first fit.

(d) coarct fit (C.F.):

Largest free partition \Rightarrow Selecting the largest available free partition and get placed in that partition.

Performance of MFT :-(1) fragmentation :-

- * Internal

- * External \Rightarrow when unable to accommodate a process of size even though we have sufficient size of memory.

1 partition \Rightarrow 1 process

(contiguous allocation)

Eg:- Process = 180 KB

80 KB
150 KB
*
280 K
*
350 K
110 KB

$$\begin{aligned} & 80 + 150 + 110 \\ & = 340 \text{ KB} \end{aligned}$$

If we have a new process of 180 KB to place into memory, but there is no certain partition to get fit into this proc even though it has total of 340 KB free; so, it is called as External frag.

(2) Maximum process size (or) maximum partition size \Rightarrow no flexibility.

~~(3) Degree of multiprogramming :-~~

Restricted to no. of partitions (can't be $> n$).

↓
no flexibility.

(4) Best fit is superior to all.

In order to overcome all these problems, we have m.v.t.

M.V.T (variable partitions) :-

⇒ Dynamic approach

⇒ No user area for partitioning

P ₁	80 K
P ₂	110 K
P ₃	75 K
P ₄	180 K
P ₅	260 K
P ₆	320 K
	300 K

PCM (Partition Control memory)

Used for storing the control information of particular partition.

* No use of limit registers in M.V.T for protection.

* limit value
size
status
L.L, U.L

Stored in PCM.

Eg.: If a new process of 70 KB is needed to fit into memory, then if we take first fit, the 80 KB partition is divided as 70 KB for the new process & rest of 10 KB. So, we overcome the fragmentation problem here.

Process (new) = 70 KB

Performance of M.V.T :-

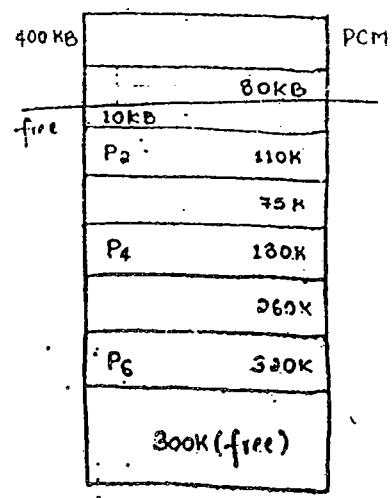
(1) fragmentation:-

* External

* No Internal.

(2) Max. process size ⇒ flexible

no max. part. size.

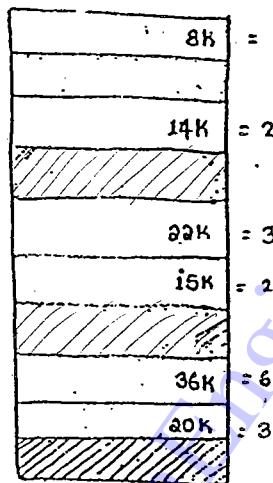


(8) Degree of multiprogramming :-

- * No restriction to no. of partitions.

- * flexible

(4) worst fit performs better, and best fit performs worst.



(i) 6K ?

First fit = 8K

Best fit = 8K

worst fit = 36K

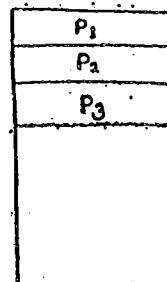
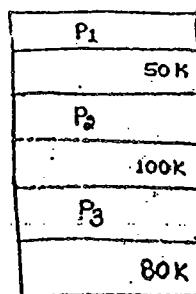
worst fit = 36K

(ii) How many successive requests
of size 6K can be satisfied.

Ans: 17

External fragmentation:-

(i) Compaction (dynamic relocation of processes):-



All the processes at different
memory locations are compacted
to one side and the free memo-
ry is now available.

* process should be dynamically relocatable and can be processed / continued.

If cannot be run in processes, then no. compaction is followed.

Disadvantages :-

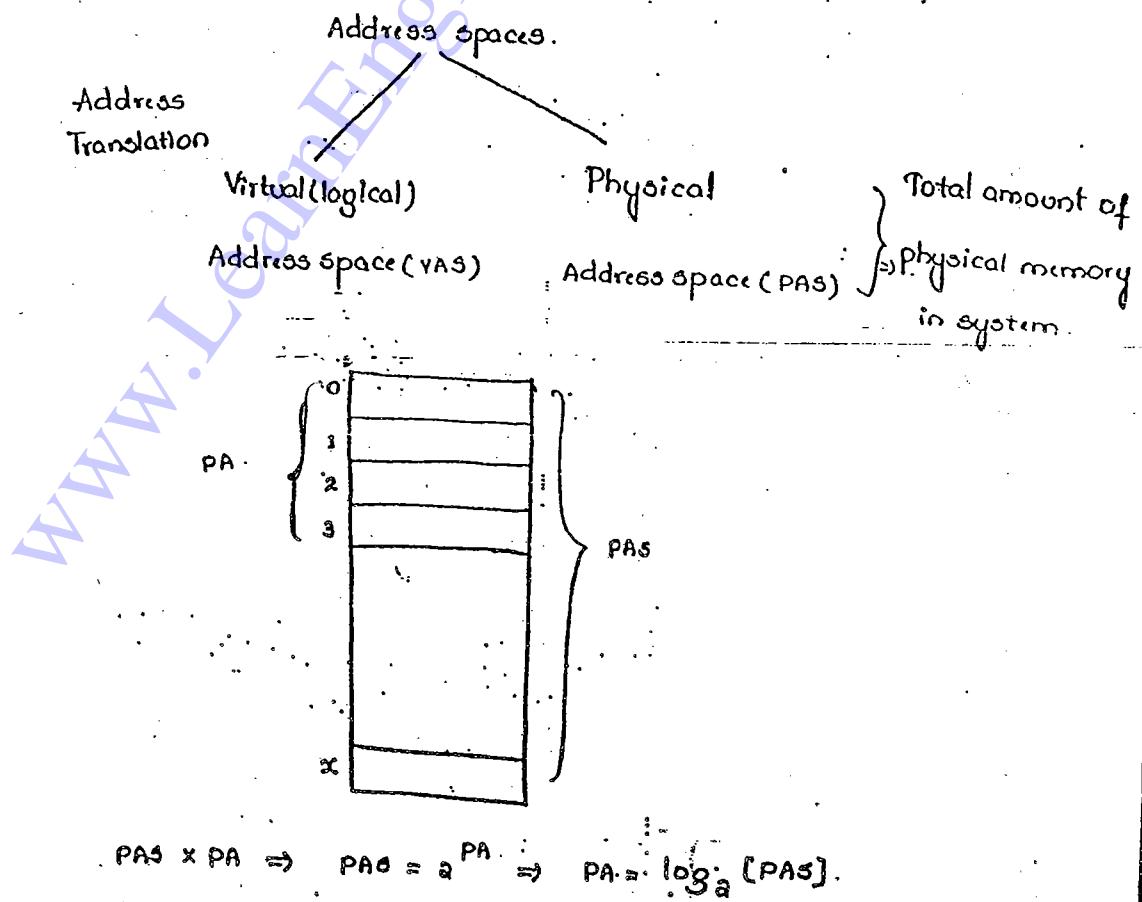
- * longer time (Since everytime relocation is done).
- * If done frequently, time is consumed.

(2) Non-contiguous Allocation :-

- * To overcome external fragmentation, we use non-contiguous allocation policies.

Address space :-

- * Space of coords / series of coords
(set)
- * A set of locations of memory is address space.



Synchronous

Logical Address Space :-

* Every process within the CPU has an assumption that it can access the whole memory available.

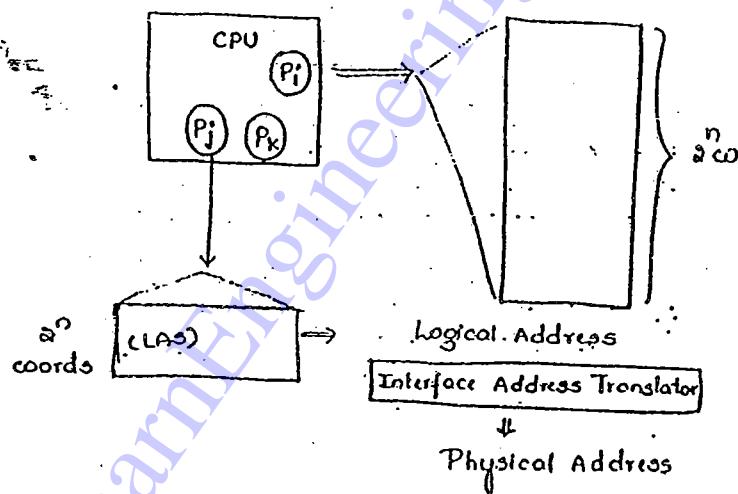
* LAS must be translated to physical for instruction/data fetch.

If $LA = 13$ bits (2^{13})

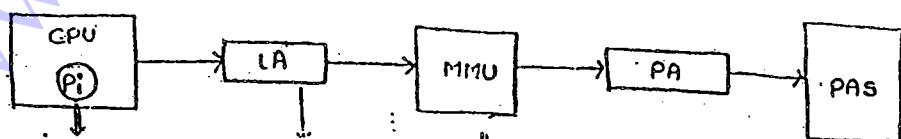
$$LA = \log_2(LAS)$$

Then, $LAS = 8 KB$

$$LAS = 2^LA$$



Architecture of non-contiguous Allocation policies :-



Always logical address is generated.

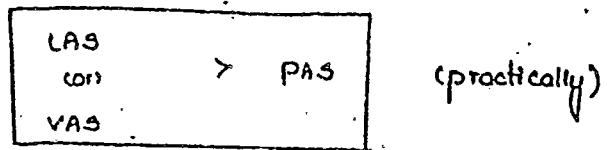
Acts as an interface between logical & physical.

(1) Organisation of LAS

(2) Organisation of PAS

(3) Organisation of MMU (mtr mgmt unit)

(4) Translation process



Because, it must have to support the concept of larger program storage in shorter memory.

Theoretically, all of them have different relation.

Paging:-

Simple paging:-

$$LAS = 8 \text{ KB}, \quad PAS = 4 \text{ KB}, \quad LA = 12 \text{ bits}, \quad PA = 12 \text{ bits}.$$

(1) organisation of LAS:-

(i) LAS is divided into equal size pages

(ii) Page size is a power of 2.

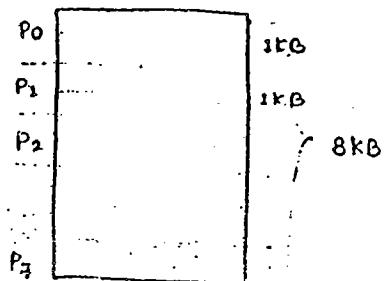
$$\text{(iii) No. of pages } (N) = \frac{\text{LAS}}{\text{Pagesize}}$$

$$\text{(iv) page no. } (P) = [\log_2 N]$$

$$N = 2^P$$

$$\text{(v) page offset } (d \text{ bits}) = [\log_2 P]$$

$$P = 2^d$$



(vi) Format of logical address:

$$LA = 12 \text{ bits}$$



pages.

$N = 2^P$
 $1024 = 2^{10}$
 $\log_2 1024 = 10$

(a) Organisation of PAS:

(i) PAS is divided into equal size frames (page frames)

(ii) Frame size = page size

frame size is same as page size

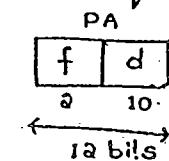
(iii) No. of frames (m) = $\frac{\text{PAS}}{\text{PS}}$

(iv) Frame no. ('f' bits) = $\lceil \log_2 m \rceil$
 $N = 2^f$

(v) frame offset = page offset

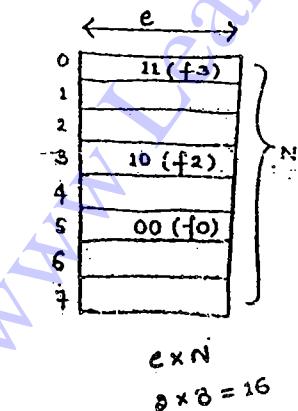
(vi) frame holds the page, therefore, frame offset is equal to Page offset.

PAS = 5 KB
fA



(b) Organisation of MMU (page table):

page map Table \Rightarrow Associated with processes.



LAS
P ₀
P ₁
P ₂
P ₃
P ₄
P ₅
P ₆
P ₇

PAS = 4 KB
f ₀ P ₅
f ₁ P ₃
f ₂ P ₃
f ₃ P ₀

* No. of entries in P.T (P.T size) = No. of pages in LAS.

* PT entries contain frame numbers.

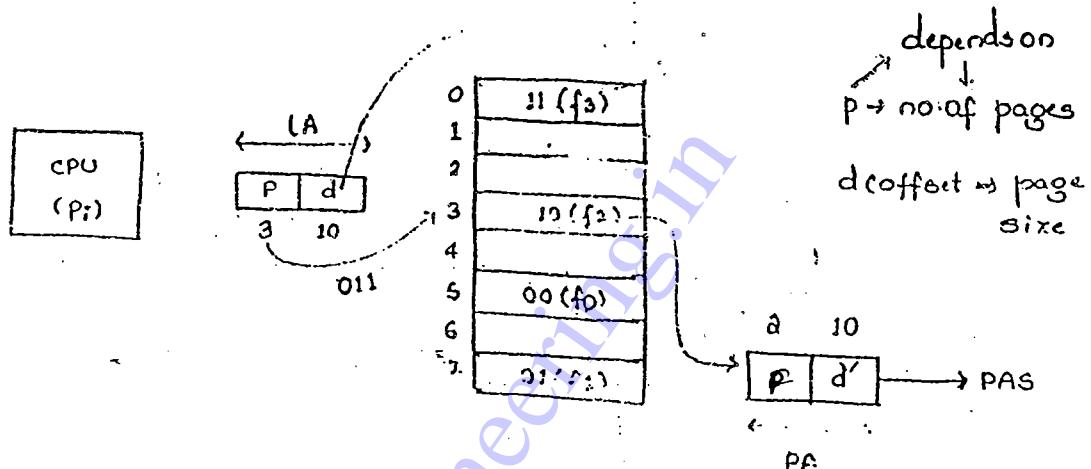
* Associated with processes (Each process has its own page table)

* page tables are stored in main memory.

* P.T size (in bytes) = $N * e$ bytes.

$e \rightarrow$ page table entry size in bytes

Address Translation:-



Eg :-

$$LAS = 32 \text{ MB}, \quad PAS = 256 \text{ KB} \quad N = ? \quad m = ? \quad p = ? \quad f = ? \quad d = ?$$

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

$$\text{P.T size} = ?$$

Sol:-

$$N = \text{no. of pages} = \frac{32 \text{ MB}}{4 \text{ KB}} = \frac{2^{25}}{2^{12}} = 2^3 = 8 \text{ K.}$$

$$M = \frac{256 \text{ KB}}{4 \text{ KB}} = \frac{2^{18}}{2^{12}} = 2^6 = 64 \text{ K}$$

$$p = \text{page no. (depends on no. of pages)} = 18 \text{ bits} \quad (N = 8 \text{ K}) \\ = 2^3 + 10 \\ = 13$$

$$f = \text{frame no. (depends on } M) = 6 \text{ bits.}$$

$$d(\text{offset}) \text{ depends on page size} = 12 \text{ bits} \quad (4 \text{ KB}) \\ = 2^2 + 10 \\ = 12$$

$$\text{P.T size} = \text{no. of pages} = 8 \text{ K}$$

$$\text{If } e = 4 \text{ B then P.T size} = e * N$$

$$= 8 \text{ K} * 4 \text{ B}$$

$$= 32 \text{ KB}$$

Eg:- 2K pages & 512 frames calculate

If PAS = 4MB, e = 4B, N, M, P, f, LA, PA, d, PTS.

Sol:-

$$N = 2K$$

$$M = 512$$

$$PAS = 4MB$$

$$P = 11 \text{ bits} \quad (\text{2K} \Rightarrow 2^10 + 10 = 11)$$

$$PA = (2^2 + 2^0)$$

$$f = 9 \text{ bits} \quad (\text{512 frames} \Rightarrow 2^9)$$

$$= 22 \text{ bits}$$

$$LA = 24 \text{ bits} = (16 MB)$$

$$\xleftarrow{8 \text{ bits}}$$

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

$$\begin{array}{|c|c|} \hline f & d \\ \hline \end{array}$$

$$d = 13 \text{ bits} \quad (PA = 22, f = 9)$$

$$\xleftarrow{14 \text{ bits}}$$

$$P.T.S (\text{Bytes}) = N * e$$

$$\begin{array}{|c|c|} \hline P & d \\ \hline \end{array}$$

$$= 2K * 4B = 8 KB$$

$$\begin{array}{cc} 11 & 13 \\ \hline \end{array}$$

Performance of paging:

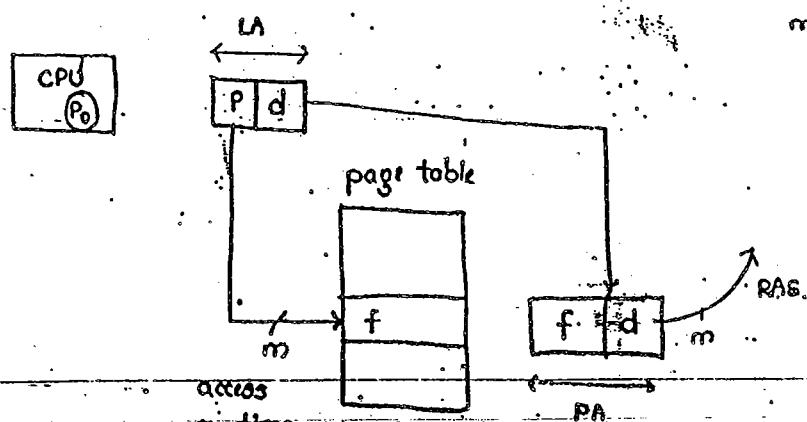
* main memory access time (mmAT) = 'm' ($\mu s/\text{no}$)

* Effective memory Access Time (EMAT)

Decrease / Reduce of EMAT from mm & bring closer to 'm'.

Access Time:

Amount of Time to read or write to memory.



(iii) paging with TLB (Translation look-a-side Buffer):

If generated LA is not found within the cache (on TLB), it is called "TLB miss". If found, it is called "TLB hit".

T.L.B Access time = c

cohere [c << m]

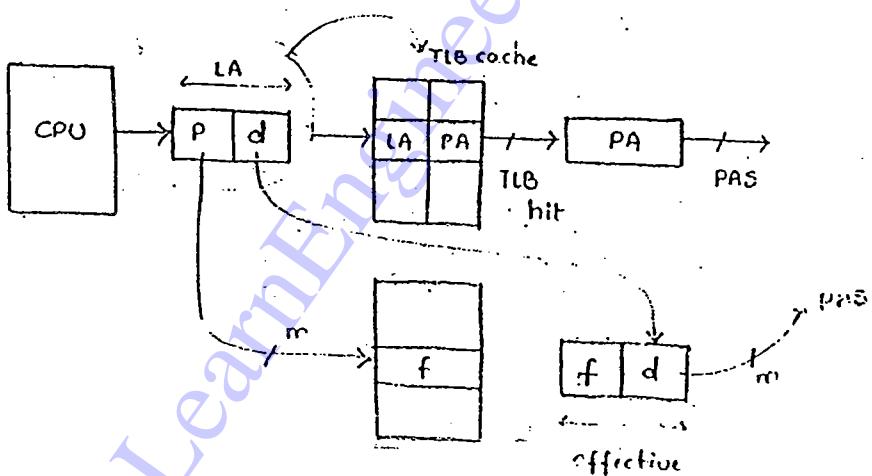
$$\text{T.L.B page hit ratio } (\alpha) = \frac{\text{No. of hits}}{\text{Total references}}$$

$$\text{T.L.B miss ratio } (\gamma) = 1 - \alpha$$

TLB is a cache contains

list of physical & logical address

If generated LA is present, then
it is TLB hit, (on it goes to page
table & generates physical address)



$$\text{EMAT}_{(\text{SP} + \text{TLB})} = \alpha(c + m) + (1 - \alpha)(c + \gamma m)$$

↓ ↓
Success failure.

Eg:- (i) $m = 100 \text{ ns}$ $c = 20 \text{ ns}$ $\alpha = 90\%$

$$\begin{aligned} \text{EMAT} &= 0.90(20 + 100) + 0.1(20 + 100) \\ &= 9(120) + 20 = 130 \text{ ns.} \end{aligned}$$

(ii) $\alpha = 10\%$

$$\begin{aligned} \text{EMAT} &= 0.10(20 + 100) + 0.9(20 + 100) \\ &= 12 + 9(120) = 810 \text{ ns} \end{aligned}$$

(additional time)
Effective memory overhead : $= m.$

(simple paging)

SP - simple
paging

$$\boxed{\text{EMOH} \quad (\text{s.p} + \text{TLB}) = x(c) + (1-x)(c+m)}$$

* By using page table, every time, we require "m" memory access time 'm' is overhead for minimising. This introduce TLB that access time is 'c' which is less than 'm'. overhead

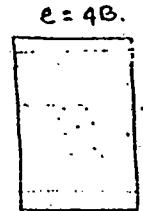
* In hit rate, Translation require "c+m", if miss occurs it takes $c+m+c$

* Multi-level paging :-

If $\text{LA/VA} = 32 \text{ bits}, \text{ P.S} = 4 \text{ KB}$.

$\text{PT Size} = 1 \text{ M}, \text{ B} = 4 \text{ MB}$.

$$N = \frac{2^{32}}{2^{12}} = 2^{20} \\ = 1 \text{ M}$$



Larger page tables
 2^m are not desired.

Reduction of page table

Increasing page size

Multi-level.

Drawback :-

Increase Internal fragmentation.

(wastage of space)

Large :- $\text{P.S} = 1 \text{ KB}$

$= 2 \text{ pages}$

Program 1024

$= 1024 \approx 1 \text{ page}$

Small :- $\text{P.S} = 8 \text{ B}$

$= 512$

$= 64$

Optimal page size = ?

Internal fragmentation

P.T overhead

must be

minimum

* LAS/VAS = 's' coords.

* page Table entry = 'e' bytes.

* page size 'p' = ?

$$\text{P.T overhead (bytes)} = \left(\frac{s}{p} \right) * e.$$

Internal fragmentation

$$= \frac{p}{a}$$

$$\text{Total overhead} = \left(\frac{p}{a} + \frac{s}{p} * e \right)$$

$$\text{To minimize/ optimize} \Rightarrow \frac{d}{dp} = \left(\frac{p}{a} + \frac{s}{p} * e \right)$$

$$(min.) = 0 \quad \left(\frac{1}{a} - \frac{1}{p^2} s * e \right)$$

$$p^2 = ase.$$

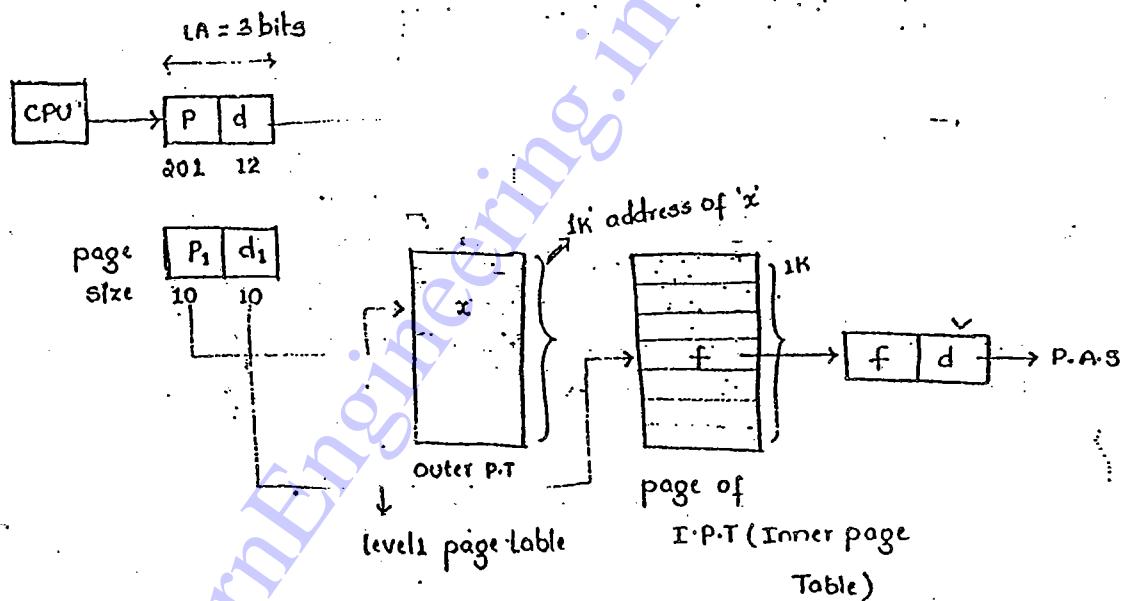
$$\boxed{p = \sqrt{ase}}$$

28/07/2010

wednesday
=

Paging :

* Dividing any address space into equal size units (pages)



Total virtual memory = 2^{32}

Each page size = 4 KB

Total pages are $\frac{2^{32}}{4 \text{ KB}} = 2^{20} \Rightarrow 1M$ pages.

Each page size is 4 KB. So, again apply paging on page table of each 1K.

Total no. of entries = $\frac{1M}{1K} = 1K$.

Two level addresses = 20 \rightarrow no. of pages
12 \rightarrow offset.

Divide again pages of each size = 1K \Rightarrow It stores address of outer page for storing address it require 10 bits

Performance :-

* Main memory access time = m

* Effective access time = $m + m + m = 3m$

↓ ↓
Inner Outer

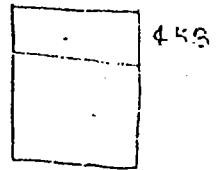
* Effective access time with n level paging = $(n+1)m$.

for reducing access time using TLB :

TLB hit ratio = α

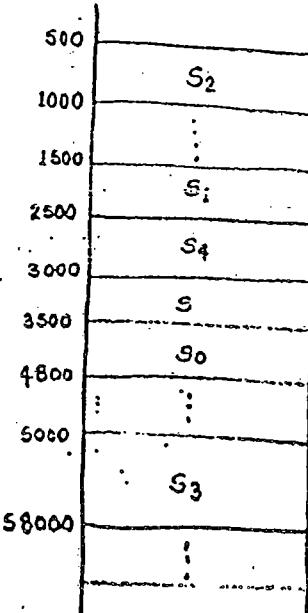
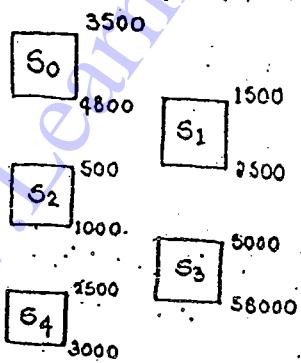
$$= \alpha(c+m) + (1-\alpha)(c+3m)$$

For n -level pages = $\alpha(c+m) + (1-\alpha)(c+(n+1)m)$



Segmentation

Paging doesn't preserve user's view of memory allocation.



Segment is a piece of code.

Program, divided into unequal sizes is called Segment (for equal size is

called pages).

(1) Divide the program into segments (VAS organisation).

(2) Using policies place the segments.

(Base of segment)

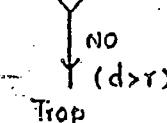
It stores Base address in physical m/r

B	L
0	3500
1	1300
2	1500
3	500
-	500
-	5000
-	800

length of the segment

s d

d



Segmentation fault

Yes

No
(d > r)

Trap

+

b (500 address)

p.A

In page table, all are equal sizes. So, no need to compare the required coord present within limit or not. Here, unequal no. of partitions for checking the offset present in colthin the limit use magnitude comparator.

Segmented paging :-

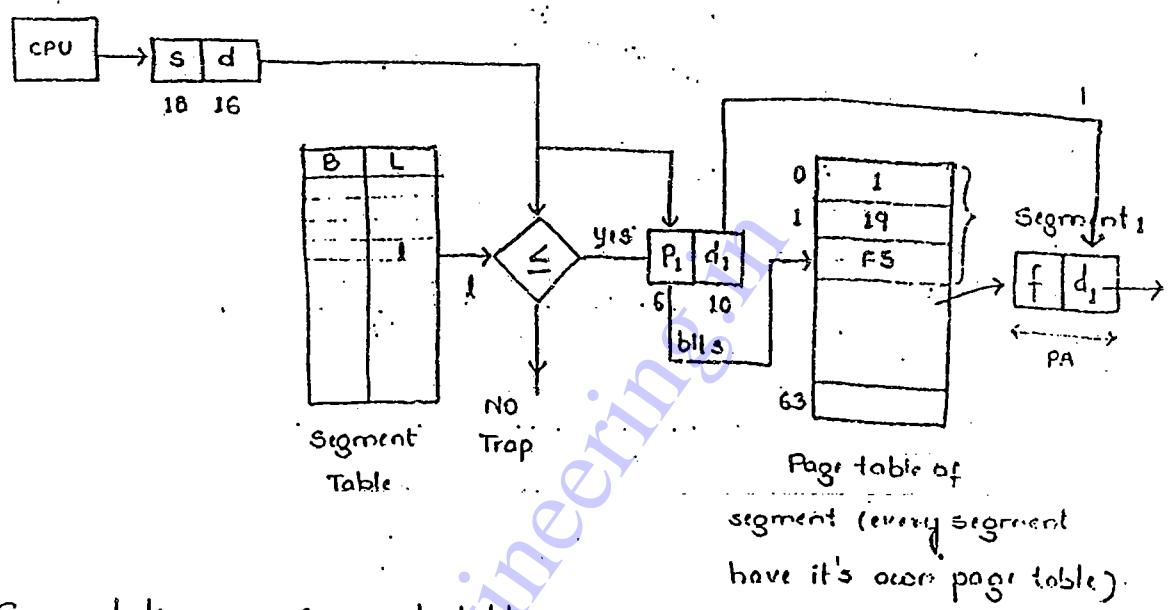
Introducing paging (or) segmentation

Virtual Size = 34 bits $\langle s, d \rangle = \langle 18, 16 \rangle$ bits

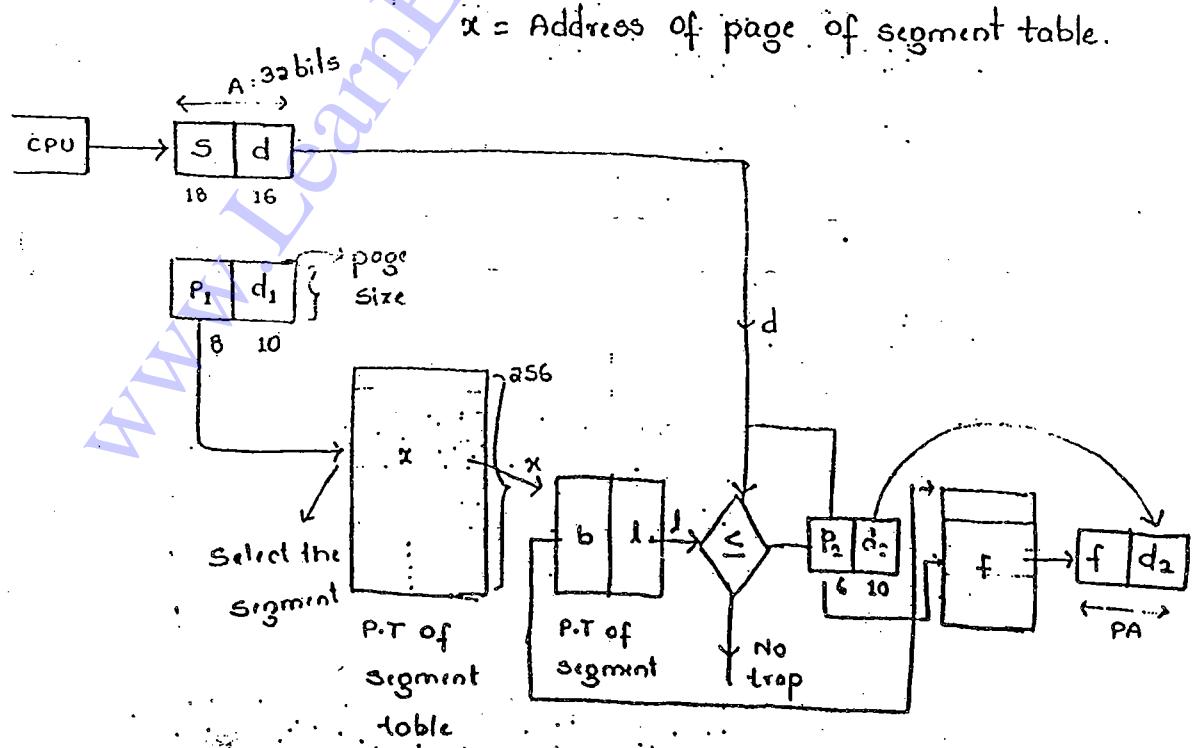
Max. Size of seg. size = 2^{16} = 64 KB
↓
offset

No. of segments = 2^{18} = 256 K

Segmentation of Segment Size :-



Segmentation of Segment table:



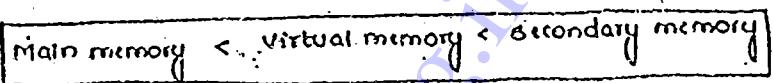
Paging Vs Segmentation :-

	Internal fragmentation	External fragmentation
Paging	✓	✗
Segmentation	✗	✓

Virtual Memory

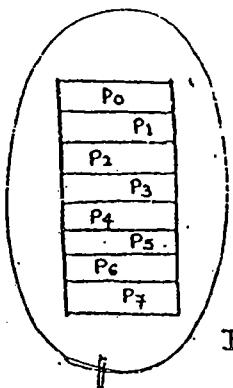
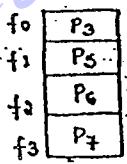
Virtual memory gives an illusion to the programmer that a huge amount of memory available for writing programs greater than the size of available physical memory.

Virtual memory implemented in secondary memory.



Demand paging :

Virtual memory is implemented with demand paging and demand segmentation.



Implemented in S.S. maximum virtual memory is limited by capacity of secondary storage

Demand paging

Pure Demand paging

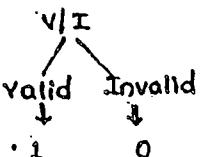
Started with all the frames empty right from the first demand for page.

Protected if all the frame available

Protected Demand paging.

prefetch

load is now



Page fault rate is higher in pure demand paging and less in prefetched demand paging

Page demand Paging is connection less. It takes less
settling time, and is preferable to connection oriented if it takes
more time for execution.

For More Visit : www.LearnEngineering.in

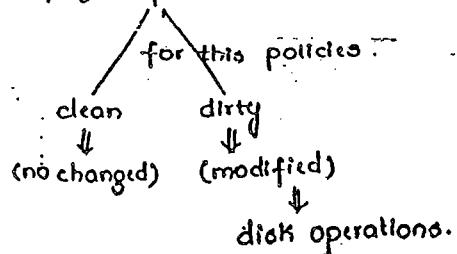
54

CPU	P	d	0	10	1
			1	-	0
			2	-	0
			3	00	1
			4	-	0
			5	01	1
			6	-	0
			7	11	1

when page fault occurs, duty of virtual handler :-

- * Process is blocked
- * Virtual handler \rightarrow CPU
- * V-m handler \rightarrow Device manager
- * Device manager \rightarrow Device controller
- * Device controller \rightarrow Device buffer.
- * D-B \rightarrow m-m (DMA) transfer the page.
- * (i) Empty free is available \Rightarrow (i) disk operation is enough.
(ii) No empty free \Rightarrow page replacement.

page fault service time
 \Rightarrow generally 10^{-3} sec. for
memory accessing, it
takes $\frac{10^{-6}}{10^{-9}}$ sec.



- * Update page table

- * Unblock the process

address

--	--

maximum virtual memory size is less than
disk i.e. not more than Disk

Performance

Main memory Access Time (M.M.A.T) = 'm'

P.E.S.T = 's' ($s \gg m$)

Page fault rate = 'p'

Page hit ratio = $1-p$

$$EAT = (1-p)m + p * s$$

$$E.A.T = (1-p)m + p(m+s)$$

Hit ratio is more \Rightarrow follow the units of main memory access times.
 $\Rightarrow 1.9999 \mu s$

$$\begin{aligned} E.T.T &= \frac{1}{K} (i+j) + \left(1 - \frac{1}{K}\right) * i \\ &= \frac{i}{K} + \frac{j}{K} + i - \frac{i}{K} \end{aligned}$$

Page replacement :-

1) frame Allocation policies:

Total no. of frames = M

No. of processes = 'n'

Demand of each processes for

frames = S_i (it depends on

Total no. of pages of program)

Total demand of process:

$$S = \sum_{i=1}^n S_i$$

	S_i	Equal proportionate	50%
--	-------	---------------------	-----

P_1 10 10 5

P_2 25 10 12

Only given to

1 so% of resource

P_3 3 \rightarrow 10 unjustify

P_4 12 10 6

P_5 20 10 10

70

It is applicable for all, which have equal demand.

If $M = 50, n = 5$ $\frac{M}{n} = \frac{50}{5} = 10 = \frac{n}{s} \times m$ (and policy)

$$P_1 = \frac{10}{70} \times 50 = \frac{500}{70} = 7\frac{1}{7}$$

$$P_2 = \frac{25}{50} \times 50 = 25$$

(b) Page Reference string :-

Set of successively unique pages referred in the given list of logical addresses :-

463, 182, 184, 195, 435, 965, 967, 834, 128, 534, 765, 784, 018, 634, 686

page size = 100 words (consider).

463 \Rightarrow It belongs to
page address

$\frac{463}{100} = 4$ (every page contains
100 words)
 $63 \rightarrow$ offset.

each page	P ₀	0
P ₁	99	100
P ₂	199	

$$463 \% 100 \\ = 63$$

$= \{4, 1, 1, 1, 4, 9, 9, 8, 1, 5, 7, 7, 0, 6, 6\}$ // If occurred successively, then consider only once:
ref. string = $\{4, 1, 4, 9, 8, 1, 5, 7, 0, 6\}$ // '4' occurs again but not successively, so consider length = 10

No. of unique pages referred in the given list of logical addresses :-

$$n = \{0, 1, 4, 5, 6, 7, 8, 9\} = 8$$

Eg. :-

reference string = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 0, 0, 1, 7, 0, 2

phase

length = 80, unique = {0, 1, 2, 3, 4, 7}

3 frames

4 frames

Frame size = 4

$$n = 6$$

page fault = 10

$$3F \rightarrow 16$$

$$4F \rightarrow 10$$

A	A	A	0
B	B	B	0

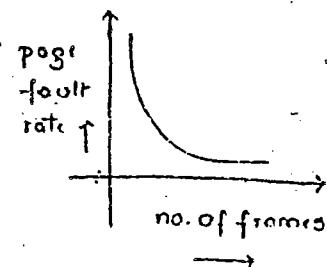
A	B	C	D
B	C	D	0

Pure demand page (PDP) :-

(1) page table policy decides which page is replaced.

* conventional policy is - FIFO.

Criteria : A.I (It maintains A.I in page table)



Reference string :-

1, 0, 3, 4, 1, 0, 5, 1, 0, 3, 4, 5

FIFO :

3F \rightarrow 9

4F \rightarrow 10

↓

so frames increases,

page faults also increase \Rightarrow Anomaly

3 frames

X	A	5
Z	B	3
Y	C	4

4 frames

X	B	4
Z	C	5
Y	D	2
Y	E	3

Belady's Anomaly \Rightarrow unusual behaviour.

* with the increase of page frame to a process, the page fault rate also "sometimes" increased, it is called Belady's Anomaly.

(FIFO & LRU based policies suffer from Belady's Anomaly)

Optimal Replacement :-

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 1, 0, 0, 1, 7, 0, 1

P₇ \rightarrow 8

P₃ \rightarrow 9

X	Z	1
0		
X	Z	7
2		

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

P₃ \rightarrow 7

X	Z
0	
8	4

X	Z
0	
3	

4F \rightarrow 6 (But it is not implementable)

301071a010

Friday

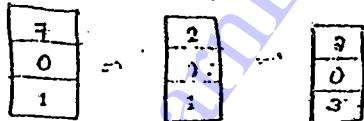
Page Replacement policies

Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

(i) FIFO:3 frames \Rightarrow 154 frames \Rightarrow 10(ii) Optimal replacement:3 frames \Rightarrow 94 frames \Rightarrow 8

(iii) Least Recently used (LRU):

Criteria : Time of reference (TOR)

Least TOR \Rightarrow is considered for victim.

7 is used for more

longer time, so '7'

gets replaced with '2'

P.T	f	v/I	AT	TOR

7	3
0	2
1	3
2	

(iv) Most Recently used (MRU):3 frames \Rightarrow 4 frames \Rightarrow

$$\begin{matrix} 7 & & 7 \\ 0 & & 0 \cdot 3 = \\ \times 2 & & \times 2 \end{matrix}$$

(v) Counting Algorithms:

Least Recently used. (L.R.U.)

Most Recently used. (M.R.U.)

* After loading page in main memory, how many times a page is referred so far:-

• Performance of LRU is most closest to optimal.

∴ O.S uses LRU (or LRU approximations).

X 2	1	⇒ first dropped in
0 →	2	is replaced
1		

LRU Approximations:-

* works like LRU.

* Approximate to the behaviour of LRU

They are not pure LRU
but works like LRU

Reference bit (R) Algorithm:-

* Every page table contains a reference bit within it.

Page Table				
P	f	V/I	AT	R
0	a	1	3	1
1	b	1	a	0
2	-	0	-	-
3	d	1	0	1
4	k	1	1	0

R = 0 - page has not been referred so far during the present epoch.

R = 1 - page has been referred atleast once during the present epoch.

* At the end of epoch, reference bits are cleared to zero.

Epoch ⇒ duration of time ⇒ Time divided into discrete intervals
(Time Quantum)

called epochs.

During current epoch, certain pages are referred (or) not referred are indicated by reference bit (R).

	R
P ₁	0
P _j	0
P _k	0

Initially

	R
P ₁	1
P _j	0
P _k	1

page referred in next epoch.

End of old epoch & starting next epoch has initial value = 0

Start searching the page fault (victim) from initial, using ref. bit of LRU approximation is Reference bit algorithm.

If all reference bit values = 1, then this algorithm fails.

So, we introduce a new algorithm called second chance.

(2) Second chance / clock Algorithm:

Criteria : Arrival time + reference bit
A.T + R

FIFO based algorithm.

Start the search of pages from the A.T, and if it is already referred, then give a second chance to it (i.e., value changed from 1 to 0).

P	f	v/I	AT	R
0	a	1	3	X 0
1	b	1	2	X 0
2	-	0	-	-
3	d	1	0	X 0
4	K	1	1	X 0

(3) Enhanced Second chance: Avoids unnecessary page back
 (not-recently used)

Criteria : $(R) + (m)$ (modified bit)
 +
 (dirty bit)

modified bit \Rightarrow checks whether the contents of page are modified
 (or not).

0 \rightarrow

1 \rightarrow write back.

$R=0; M=1 \Rightarrow$

R	m	
0	0	not R, not m
x	0	not referred but modified
1	0	Referred but not modified
1	1	Referred & modified

R	m	R	m
x	1	0	1

It is referred in old epoch and it is not referred in new epoch but modified.

when page fault occurs, consider the value of "00".

0 \rightarrow LRU

we look for '01' combination next, since, it represents LRU

Then '10' is considered & at last '11' is considered.

Thrashing :- (High paging activity) (page fault rate).

* Excessive / High page fault rate is called Thrashing

* It is also an undesirable state of the system like deadlock.

Reasons for Thrashing:-

Primary

(1) Lack of memory (frames)

i.e., if frame allocation to a page get reduced.

(a) High degree of multiprogramming

Secondary

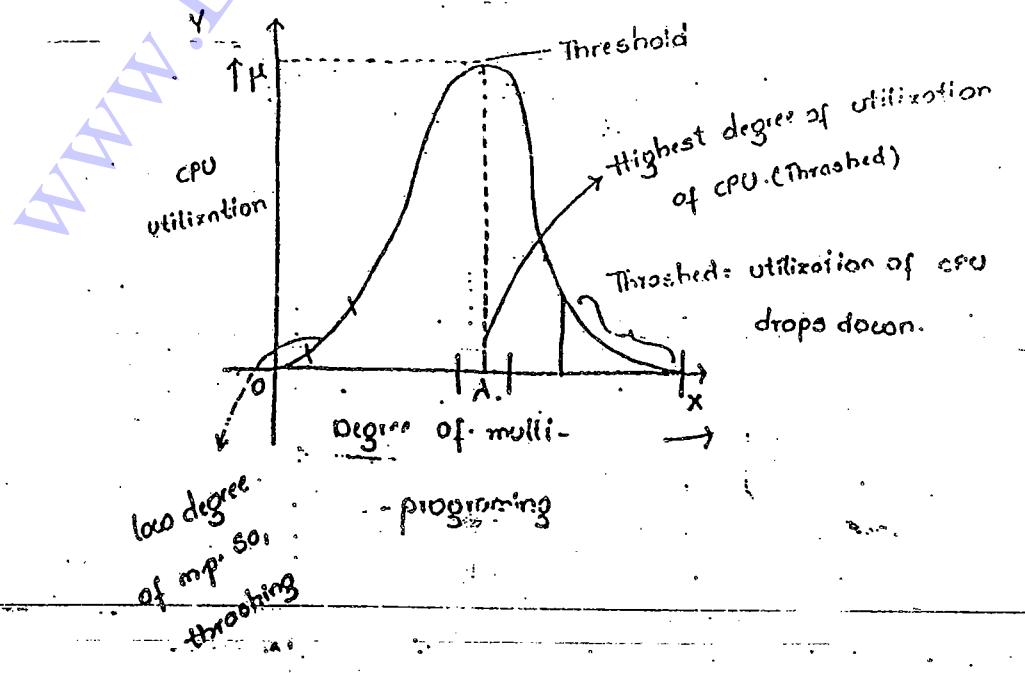
(2) page replacement algorithm.

Drawback: fragmentation

(a) page size

small (2B) → more pages
large (1024B) → less pages (less page faults) when reduces thrashing.

(3) program structure:



Thrashing control strategies :

I. Prevention & Avoidance

- * Controlling degree of m.p. in and around " μ ".

Page size:

Temporal (Time) \rightarrow Thrashing

Spatial (Space) \rightarrow Internal fragmentation.

Program Structures :-

int A(1...128, 1...128)

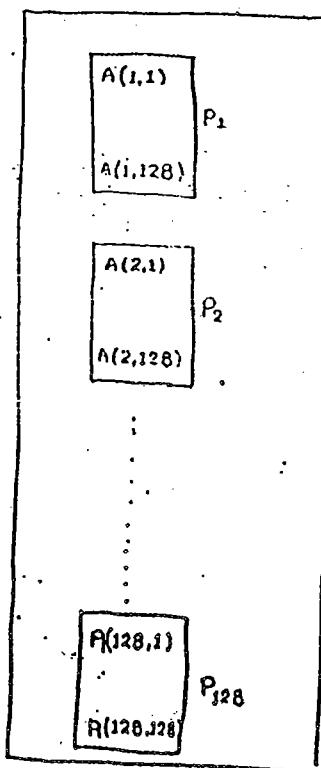
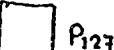
page size = 1280

i) for $i \leftarrow 1$ to 128

 for $j \leftarrow 1$ to 128

$A[i,j] = 5$

CPU Allocated



P.O.P

FIFO

II. Detection & Recovery

- * Low CPU utilization along with high degree of m.p.

- * Paging disk utilization must be high.

Recovery :

- * By applying concept of "process suspension" (swapping out some processes to secondary memory)

(Q) If there are a set of processes (from 1 to 128) to be executed. But, If the CPU allocates only 127 processes to get executed. Then, how the CPU allocation processed for the above two conditions.

Sol:- column $\leftarrow A[j, i]$
major order $A[1, 1]$

$A(i, j) \rightarrow$ row major
order

P_{128}

(1,1) not available.
(2,1) page fault
(3,1) page fault

(128,1) page fault
 $128 * 128$

each page size
is 128

$P_1 f_1$

$P_2 f_2$

$P_3 f_3$

$P_{127} f_{127}$

P_{128}

$P_2 P_1$

P_3

P_{127}

for every 'i' only 1 page
fault occurs. So, total

128 pages, 128 page
faults occur

128

working set model :-

* Based on locality of reference.

main()

{
...
f() — 5K
...
}

f()

{
...
g(); — 10K
...
}

g()

{
...
h(); — 8K
...
}

h(); — 8K

{
...
— 15 K
...
}

Total = 84K

By monitoring the locality,
main block required 15K, but
memory gives 15K based on
locality, it takes frame no's

locality (dynamic)

By Knocking, no. of frames
required based on Guess/
Estimate reference string

and divide.

Objective :-

* The main objective is to reduce page fault rate.

- * Locality may be a function, block, class or module.
- * Hold the pages of the locality at which the present function is performing.

Guess / Estimate : Estimating the ref. string in each function.

Reference String (pri):

			size
			↓
			t
7, 6, 8, 9, 6, 8, 9, 7, 12, 18, 20, 22, 18, 20, 12, 23			
main()	f()	g()	

Working set coincides (WSW):

- * Set of unique pages referred in the given list of reference string during the past ' Δ ' references.

↓
Integer (Guess)

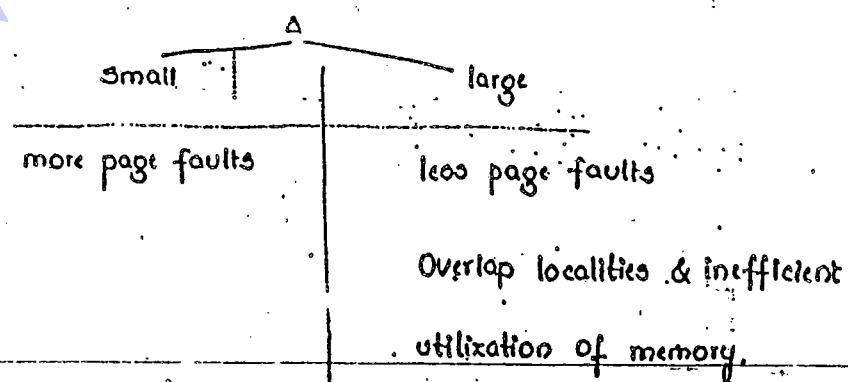
Let $\Delta = 10$ (Guess) \Rightarrow no. of integers in $g() = 10$ (length)

$$\text{WSW} = \{23, 27, 28, 29, 34, 38\} = 6.$$

↓

Unique reference string from the
set of $g()$ function.

Success will depend on the value of Δ .



* If we guess the value of A to be small, then there is a chance of occurring more page faults.

Eg :- If $A = 10$

$$\text{MWSW} = \{ 23, 28, 29, 34, 38, 24, 28, 23, 29, 38, 34 \}$$

page faults = 6.

If $A = 5$

$$\text{MWSW} = \{ 28, 23, 29, 38, 34 \}$$

Page faults = 5.

If $A = 5$

$$\text{MWSW} = \{ 23, 28, 29, 34, 38 \}$$

page faults = 5.

for the same set of reference string, if $A = 10$ (more no. of integers are consider), then page faults = 6. other than if $A = 5$; it have page faults = 10.

* Let $P_i = 1, \dots, n$

$$\{\omega \omega \omega\}_i \text{ where } i = 1, \dots, n$$

$$\text{Total demand} (s) = \sum_{i=1}^n (\omega \omega \omega_i)$$

Total frames available = "m".

* If $m = s$ (no thrashing, system is perfectly balanced).

* If $m > s$ (no thrashing)

(scope of increasing degree of multiprogramming).

* If $m < s$ (Thrashing)

1000 20,000

File System & Device management

Interface :

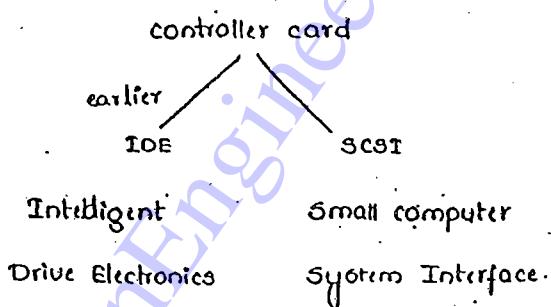


purely-
-electronic

Electro-
-mechanical.

Every secondary/ Tertiary media
must have their own file system.

(1) We need a controller interface as controller card / chip. \Rightarrow Hardware requirement



(2) Software requirement :

* Device driver.

(3) Device Independent software :

* It is called as file system. (Third party device drivers).



files & Directories :-

file :- Collection of logically related set of records of an entity.

- * It is considered for data structure.

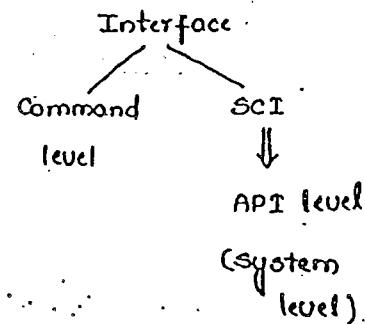
* Data Structure :-

(1) Definition

- ## (a) Representation / Implementation :-

- * Flat structure
 - * Record structure
 - * Tree structure

} \Rightarrow file structures



(3) Operations:

- * Create a file
 - * Open a file
 - * Read/ write/ append/ seek/ modify/ truncate
 - * Close a file
 - * Delete a file

(4) Attributes :-

- * name, extension of file

- * Type of file

- # Size of file

- * Owner.

- ## * permission

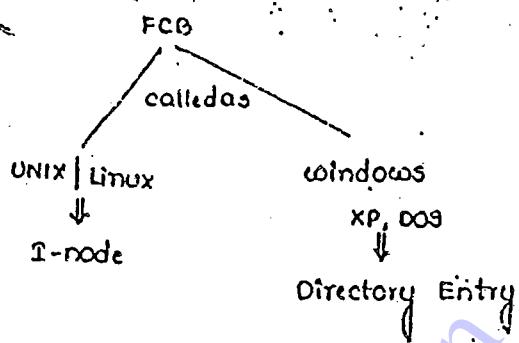
- * mode of access (sequential/random)

- ### * Time & Date Stamps

- ### Link count.

Stored in
file control block (FCB)

- * Attributes of a file are stored in file control Block (FCB).

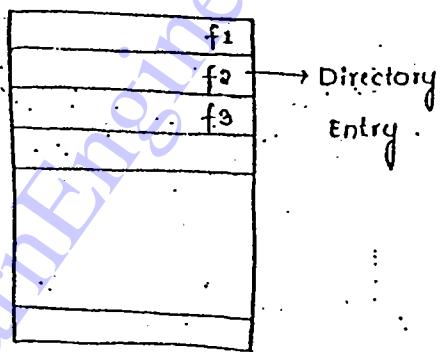


Directory :-

- * It contains information about files i.e., metadata of files.

Abstract view of
directory (metadata of files)

1 Directory Entry
per file.



Organisation of Directory Structure :-

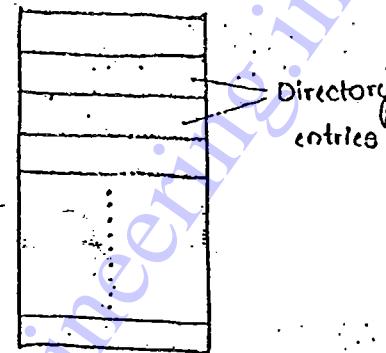
- * Single level directory
- * Two level directory
- * multi level / Tree Structures
- * Acyclic Graph.

31/07/2010

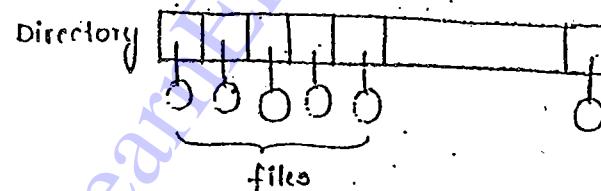
saturday

Directory Structures :-

Abstract view of Directory.



(a) Single-level Directory structure :-



- * No sub-directories are present. (one entry for each file).
- * All the files are managed by one directory.

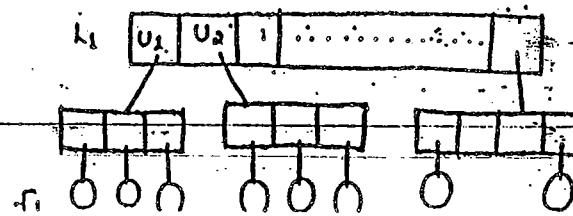
Advantages :-

- * Simplicity
- * Searching
- * Name conflicts

Disadvantages :-

- * Only one directory, no sub-directories.
- * Two different users, simultaneously can have same filename.

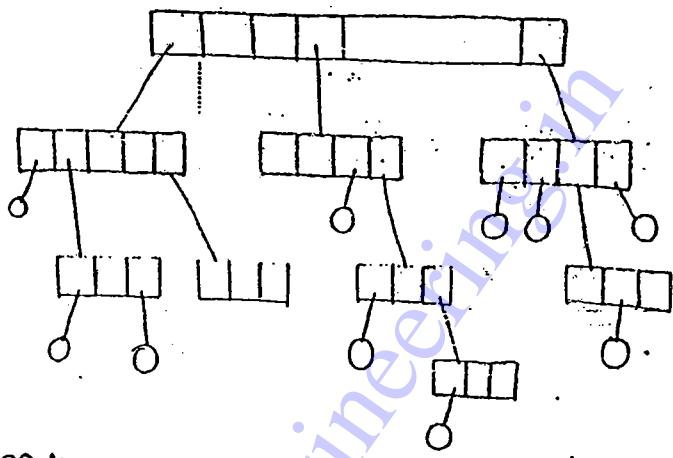
(b) Two-level Directory structure :-



Advantages :-

- * Creating different files with same name. (flexibility)

3) Tree-structured directory :-



Drawback :-

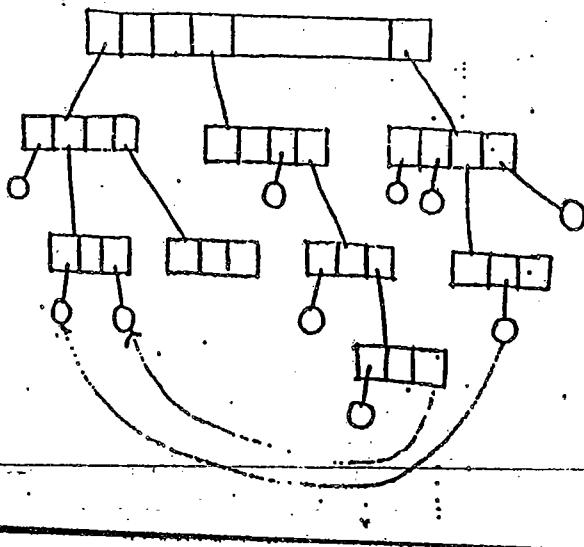
* File sharing :-

* Sharing by duplication has drawbacks of :-

- * Inconsistency

- * wastage of space

To overcome duplication in file sharing, we have the concept of file sharing with links called "Directed Acyclic Graph".

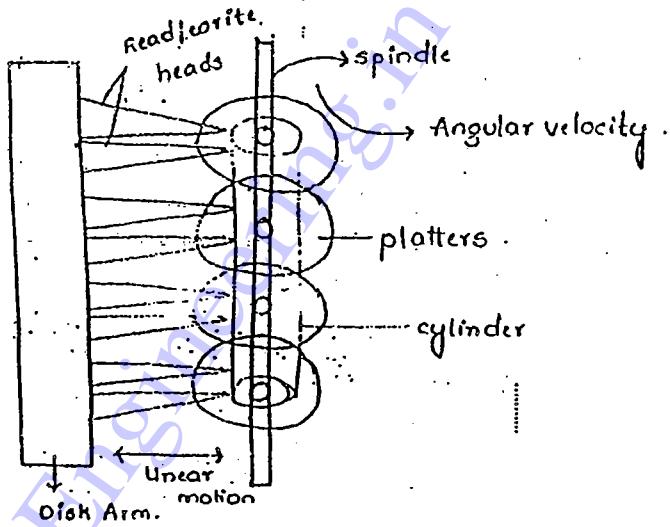


Device characteristics:

- * Major component in secondary storage is disk.

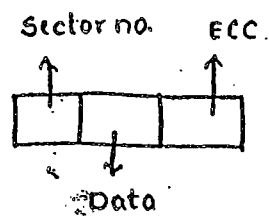
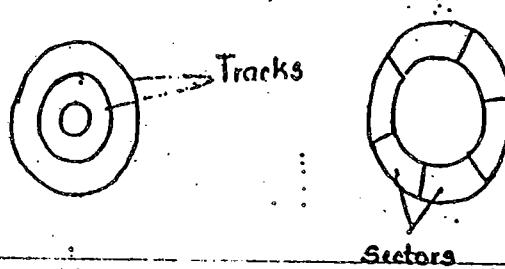
Physical and logical structure of Disk :-

Physical structure / geometry of Hard disk :-



- * Hard disk consists of set of platters, with a spindle.
- * Each surface of platters is associated with Read/ write heads.
- * Two Types of motions are supported :
 - * Linear velocity (move forward & backward) with disk armature.
 - * Angular velocity with disk spindle.
- * Tracks are divided into sectors.

Sectors :-



Sectors can be represented by three fields :-

- * Sector no.
- * Data
- * Error correction code (ECC).

Error Correction code (ECC) :

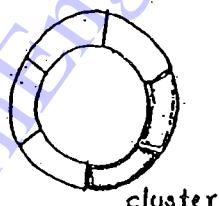
- * To detect the deadlock. By reformatting the bad sector, some sectors can be made good sector.

Cylinder :-

A group of some track number.

Cluster :-

A group of one or more adjacent sectors is a cluster. A cluster is one of the unit of I/O transfer.



I/O Transfer :-

The amount of time taken to a sector to make a move from present area to desired area is called seek time.

seek time:

time taken to

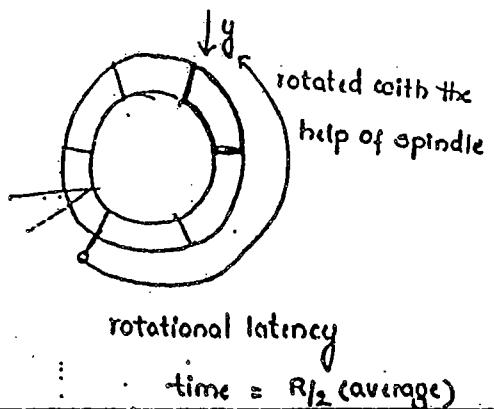
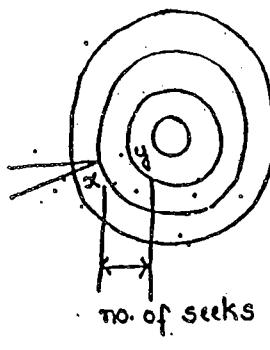
travel from 'x'

to 'y'

(or)

linear motion

from x to y



Seek time + (Rotational) latency time

$$\frac{R}{a} \text{ (on an average)}$$

where: R = Time for one rotation.

Transfer time (Transmission Time) :-

* The transfer time depends on two factors :

* Track size ((S) bytes)

* Rotation rate (RPM)

Transfer time :

* Read the data from sector

Sector size = 'x' bytes ($x < S$) i.e., Sector size < Track size.

Rotation rate = 'r'.

Time for one rotation = 'R'.

$$R = \frac{60}{r} \text{ s}$$

$$3600r = 60s$$

$$1r = ?$$

$$\text{If } \frac{60}{r} \text{ s } \Rightarrow 'S' \text{ bytes}$$

$$\Rightarrow \frac{60}{3600} = \frac{1}{60} \Rightarrow R = \frac{60}{r} \text{ second}$$

$$? \Rightarrow 'x' \text{ bytes (read)}$$

$$\Rightarrow \frac{x * 60}{r * s} \times \frac{\text{sec}}{\text{s}} = \frac{x * 60}{r}$$

bytes

DISK I/o Time :-

$$= \underbrace{(\text{Seek Time}) + (\text{Latency Time}) + (\text{Transfer time})}_{\downarrow}$$

$$= ST + \frac{R}{a} + \frac{x * 60}{r * s} \text{ (sec)}$$

$$\frac{60}{r} = S$$

$$? = x$$

$$x * 60$$

$$r * s$$

Effective Data Transfer Rate (DTR) (Bytes/sec) :: no. of bytes in 1 sec.

Assume Track size = 's' bytes

$$\text{RPM} = r$$

$$\text{Rotation time} = \frac{60}{r} \text{ sec}$$

$$\frac{60 \text{ sec}}{r} \text{ --- } 's' \text{ Bytes}$$

$$1s \text{ --- ?}$$

$$\boxed{\text{DTR} = \frac{r * s}{60} \text{ bytes/sec}}$$

Eg:- consider a hypothetical disk with :-

16 platters

↳ 2 surfaces

Disk seek time = 30 ms

↳ 2 Tracks

RPM = 4000

↳ 512 sectors

↳ 2 KB

Then calculate (i) capacity (ii) I/O time (iii) DTR

Sol:-

$$(i) \text{ Total no. of sectors} = 2K \times 512$$

$$\text{capacity of one surface} = 2K \times 512 \times 2KB$$

$$= 2^11 \times 2^9 \times 2^{11}$$

$$\text{Capacity of two surfaces} = 2^5 \times 2^{11} \times 2^9 \times 2^{11} \times 2^{36}$$

$$= 64.0B.$$

$$(ii) \text{ I/O time} = \underbrace{\text{seektime} + \text{latency time} + \text{Transfer time}}_{\downarrow}$$

$$\frac{R}{2} + \frac{X * 60}{T * S} (\text{s})$$

$$R = \frac{60}{4000} \text{ s} \Rightarrow \frac{R}{a} (\text{latency time}) = \frac{60}{8000} \text{ s}$$

Transfer time :- $\frac{60}{4000} \text{ s} \text{ --- } 1 \text{ MB}$

? --- 2 KB

$$\Rightarrow \frac{60 \times 2 \text{ KB}}{4000 \times 1 \text{ MB}} \text{ s} = x$$

$$\therefore \text{I/o time} = \left[\frac{S \cdot T + L \cdot T + T \cdot T}{R/a + \frac{x \times 60}{r \times s} (\text{s})} \right]$$

$$= 30 \text{ ms} + \frac{60 \text{ ms}}{8000} + x$$

$\xrightarrow{\text{B.P.}} \beta \neq \alpha$

(iii) DTR :-

$$\frac{60}{4000} \text{ s} \text{ --- } 1 \text{ MB}$$

15 --- ?

$$\frac{4000 \times 1}{60} \text{ mb/s}$$

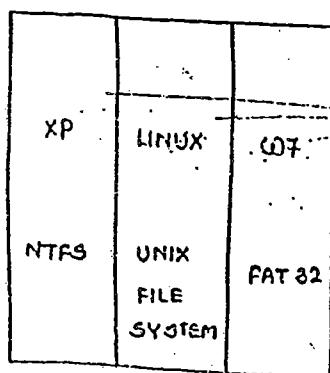
Logical structure of Disk (formatting process) :-

Popular file Systems :-

- * NTFS
- * Solaris
- * FAT 32
- * UFS
- * ZFS

popular.

fdisk ⇒ create, delete, & edit the partitions



partitions

(volumes)

(c; d; e;)

Partitions

Primary

Secondary

(Extended)

* Bootable

(stores O.S onto it)
&
userdata

* Non-bootable

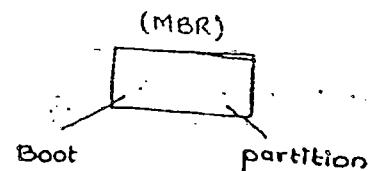
↓
only userdata
get stored.

In the partitioning of a system, there must be atleast 1 primary partition, and the rest may (or) may not be of secondary.

Computer System, which supports multiple O.S to boot is called multi-boot computer.

Master Boot Record

File format system:
we can select any
file format with
different O.S's.

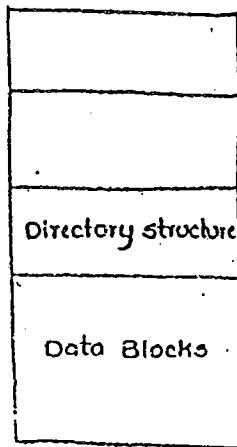


partition table (It represents the complete picture of partition)

(Booting the
selected O.S)

(only one entry for the
partition).

Partition Structures :-



Boot Control block (BCB)

Partition control block
(PCB)

Blocks containing
data of file are
stored.

(Eg.: user data, file data)

BCB block is empty \Rightarrow If there is non-bootable partition.

BCB can be represented in different terms :-

- * Boot Block (In UNIX)
- * ~~Master~~ Partition Boot Sector (In NTFS) (It contains boot of that sector).

PCB gives information about other blocks.

PCB can be represented in different terms :-

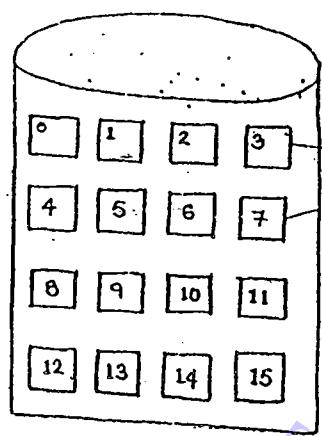
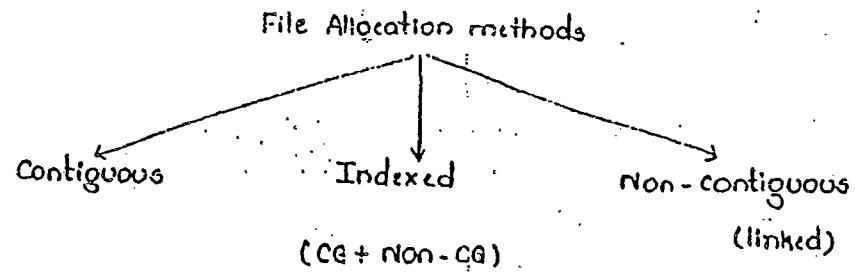
- * Superblock (In UNIX) (gives information about others)
- * Master file Table (M.F.T) (In NTFS)

Disk space is being organised in blocks.

Block \Rightarrow unit of Allocation.

Allocating the disk space methods \Rightarrow file Allocation methods.

Disk file Allocation methods (or) Disk Space Allocation strategies :



Disk blocks (Associated with two factors / parameters)

Disk Block Address (DBA)
(n-bits)

Disk Block size (bytes)

Consider DBA = 16 bits

DBS = 1 KB

Then, maximum file size (in bytes) = 64 MB

depends on/ limited by
disk size

$$\begin{aligned}
 \text{Disk Size} &= 2^{16} \times 1 \text{ KB} \\
 &= 64 \text{ K} \times 1 \text{ KB} \\
 &= 64 \text{ MB}
 \end{aligned}$$

(1) Contiguous Allocation :-

first block of disk used by file
directory entry.

filename	Starting DBA	file size (no. of blocks)	
test.c	5	4	(5, 6, 7, 8 blocks)
temp	10	6	(10, 11, 12, 13, 14, 15 blocks)

Atleast 'm' contiguous blocks require.

Performance :-(1) Fragmentation :-

Internal

External

(not contiguous, but
biferated with memory)

If last block files are
not fully utilised, then
it goes waste, which
represents internal frag-
mentation.

(2) Increasing file size :-

May / May not be possible

Inflexible

(3) Type of Access :-

Sequential

Random

Both are
supported

Sequential (Array)



Random (linked list)



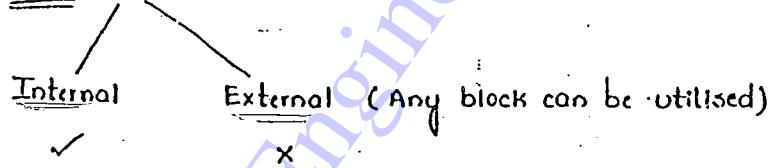
Since, it supports random access, it is faster.

(a) Non-contiguous / Linked Allocation :-

Filename	Starting DBA	Ending DBA	
test.c	4	3	(4,9,6,3)
temp	5	2	(5,1,2)

Performance :-

(1) Fragmentation :



(2) Increasing file size :-

File size increment is possible, as long as free blocks are available.

(3) Types of Access :

Sequential access only possible: No random access. So, it is slow.

Drawbacks :

* Some disk space is consumed for storing pointers; that addresses next instruction block.

* vulnerability of links :

↓ ↓
expose danger breakage of links

↓
Then, file gets truncated.

If any link breaks then Twin caters to
file

Reason for Storing ending DBA :-

* We store ending DBA, (along with starting DBA), because for the following :-

* Creating a new pointer done fastly.

i.e., To extend a new file along with the existing, it can be made faster.

* Checking file system consistency. (scandisk)

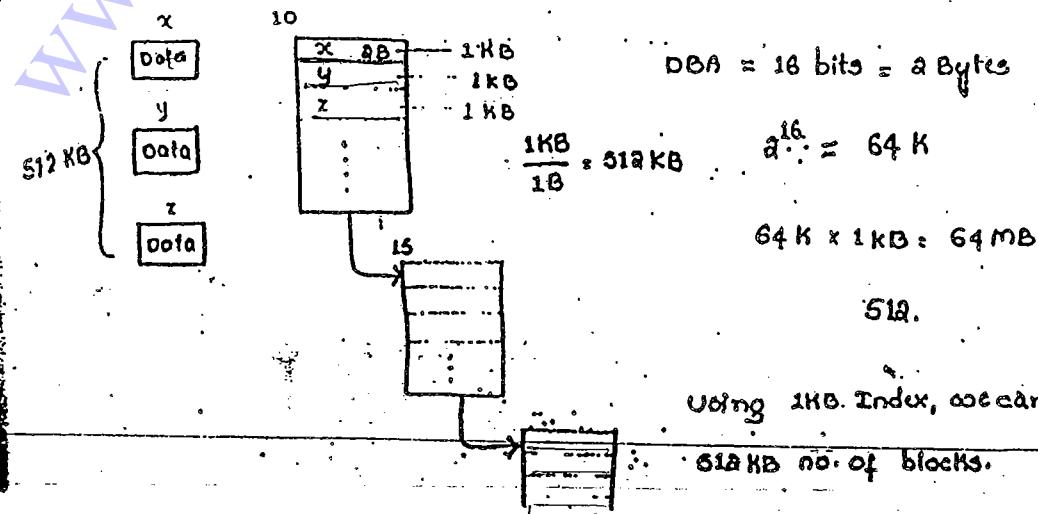
i.e., checking whether file is fully accessed (or not).

(3) Indexed Allocation (Contiguous & Non-Contiguous) :-

Filename	I-Block
test.c	10
temp.	15

* A file can be allocated in both contiguous and non-contiguous order.

* E.g. :- DBA = 16 bits, DSB = 1 KB



04/08/2010

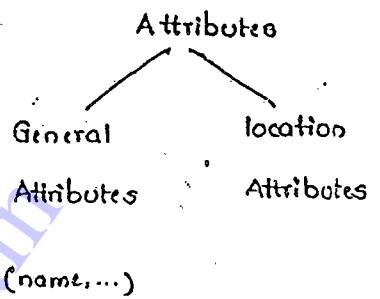
Wednesday

UNIX - I-node structure

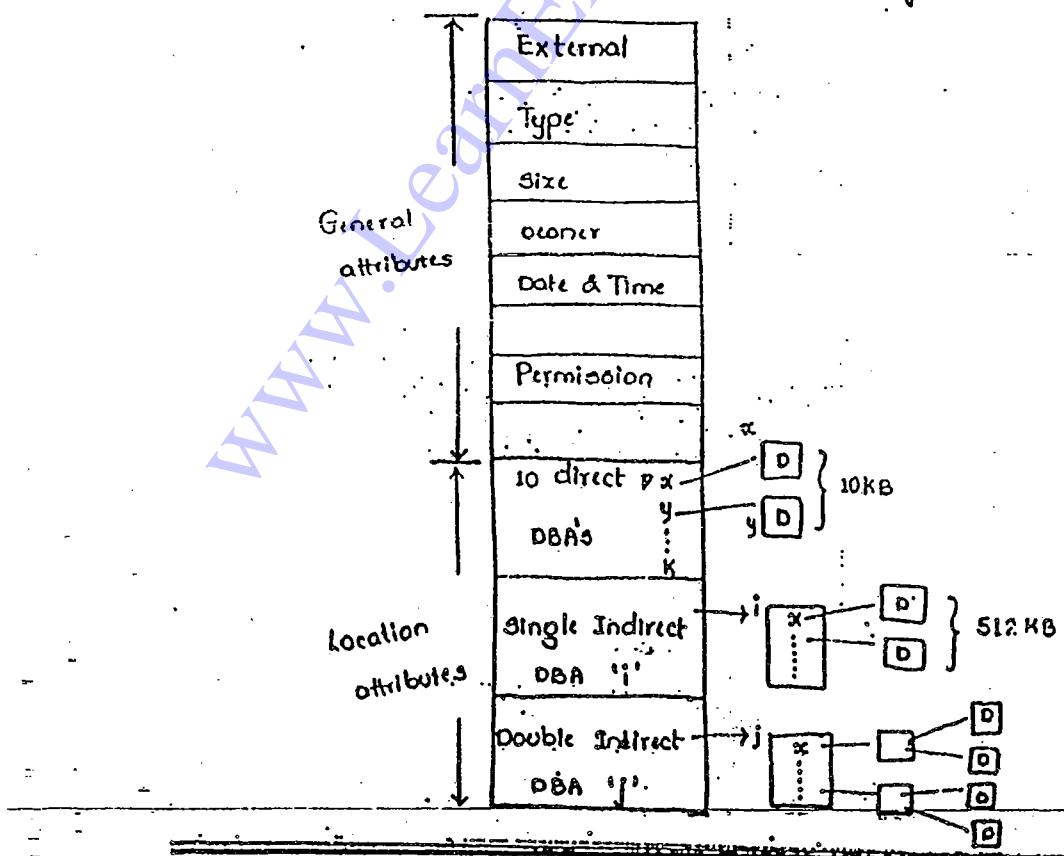
Directory entry consists of two fields :

- * filename
- * I-node (contains Attributes)

filename	I-node(23)
testc	23
temp	40



I-node(23) = block of memory



* Maximum file size is limited by the maximum disk size = 64 KB

Single Indirect DBA :-

$$\text{DBA} = 16 \text{ bits} \quad \text{DBS} = 1 \text{ KB}$$

$$2^{16} = 64 \text{ K}$$

$$64 \text{ K} * 1 \text{ KB} = 64 \text{ MB}$$

$$512 + 10 = 522 \text{ KB} = 0.522 \text{ MB}$$

Double Indirect DBA :-

$$512 * 512 \text{ KB} = 2^9 * 2^9 * 2^{10}$$

$$= 2^{28} = 256 \text{ MB}$$

Total size = 256.522 MB (logical) \Rightarrow not considered

DOS directory Implementation :- because max. file size is only 64

* DOS directory implementation is complex than that of UNIX.

* All attributes are embedded in the directory entry itself. There is no I-node in DOS implementation.

General Attributes

FN	EXT	Type	Size	Date & Time	First DBA(x)
----	-----	------	------	-------------	-------	--------------

test.c Directory entry

$\alpha \rightarrow$ first block address
of data.

* For the remaining "data" addresses to represent, we use file Allocation Table (FAT).

No. of Entries in F.A.T = No. of blocks on disk

n blocks = 0 to

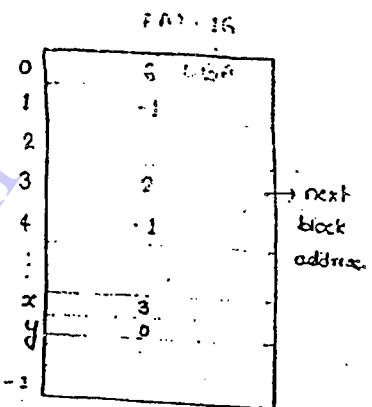
(n-1)

- * Only the first block address of data is being indicated as a field, and the remaining data addresses are stored in FAT table, and referred whenever required.

- * To know the (block) address of data, move to

'x' address on FAT. It represents the next data address. If there is last address, then it is represented by "-1" (i.e., it is last data)

- * FAT entry represents \Rightarrow address of disk block.



Tabular linked
allocation

DISK FREE SPACE MANAGEMENT

Eg. :-

consider disk = 30 MB;

$$DBA = 16 \text{ bits} \quad DBS = 1 \text{ KB}$$

$$\text{Total no. of blocks} = \frac{30 \text{ MB}}{1 \text{ KB}} = 30 \text{ K blocks}$$

Initial, 30K blocks are free.

Approaches, to keep track of these 30K blocks :-

(1) free linked list :-



If any file requires the block, it deletes the particular size (e.g. 10) and allocates it to the file.

३८

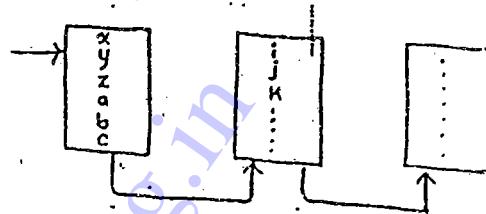
(a) Free list ::

- * The addresses of the files are stored in linked lists.

1 block \Rightarrow 512

? \Rightarrow 20 K

$$\frac{20x^2}{512} = 40$$



- * If the disk is almost free, it requires 40 blocks.

- * If the disk is almost full, it requires 1 block.

(3) Bit-map method

Associate a binary bit 0 - block is free

with each block 1 - block is in use.

Size of bit map for the
blocks = $20K$ bits.

i block can store = 8K bits

? — 20 K

$$\frac{20K}{8} = 2.5$$

\approx 3 blocks.

It requires searching time (more) for a file.

So it is slower than free list.

S - DGS

B - Blocks

D - DPA

~~P - Preblocks~~

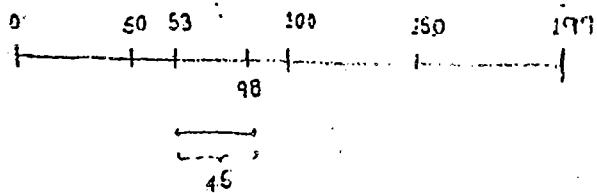
Block size, 5, 2⁰.9.

Disk Scheduling :

Ready Queue

P_a P_b P_c

CPU



CPU
(P_b)

Process Queue

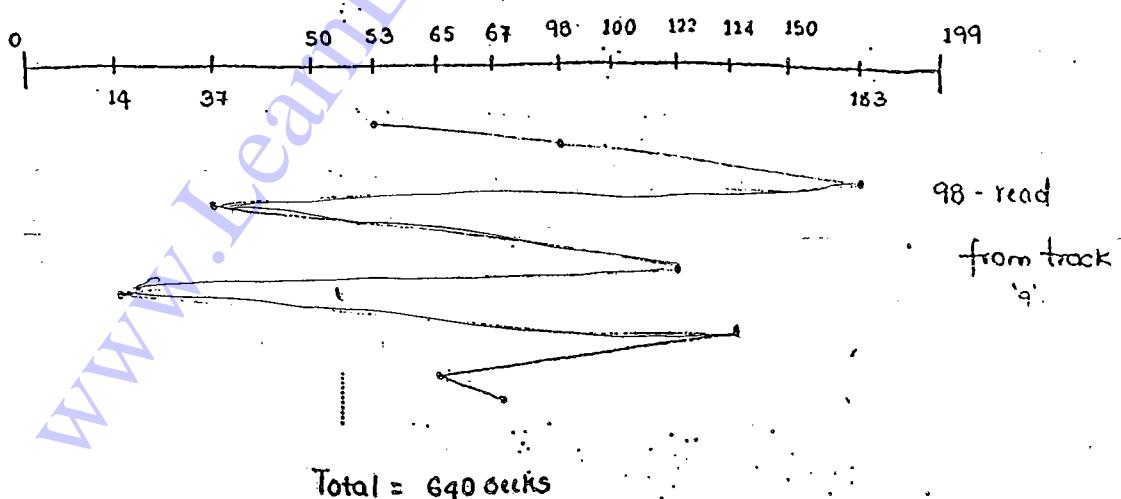
P_a P_b P_c

Device
(disk)

- Disk scheduling is used to represent which process is serviced next, when a process completes its operations.

(1) FCFS :

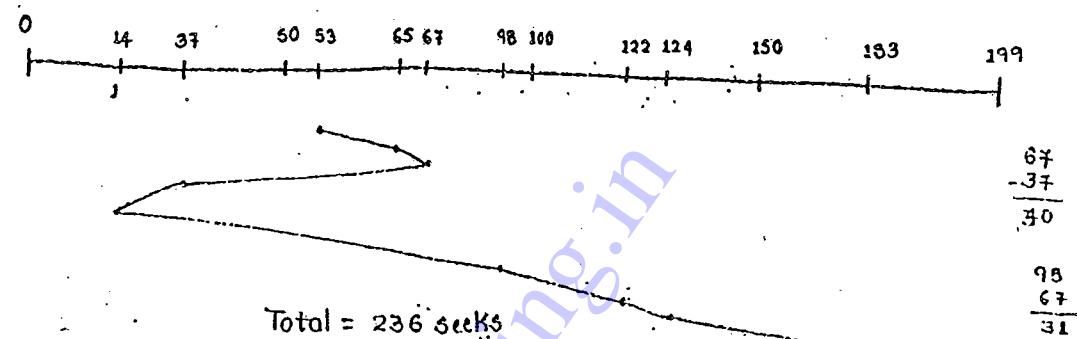
Requests : 98, 183, 37, 122, 14, 104, 65, 67



To reduce no. of seeks, we have SJF ..

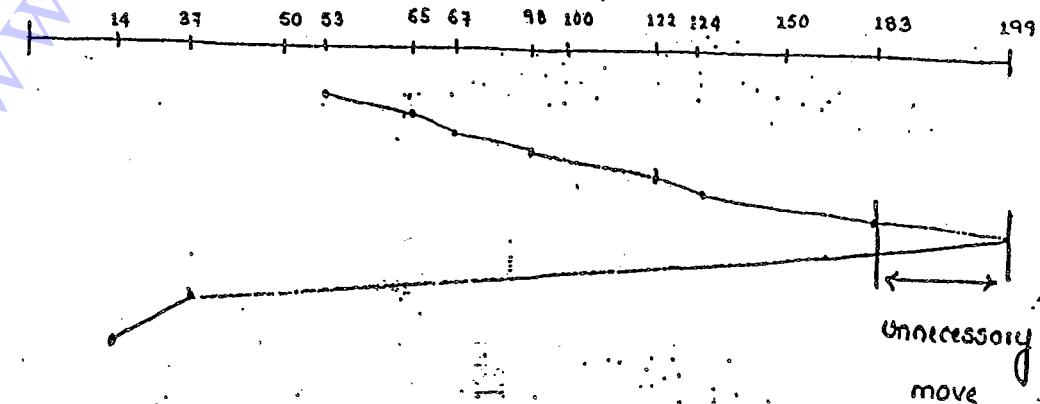
(a) Shortest Seek Time first (SSTF) / Nearest Track Next (NTN) :-

(Greedy method)



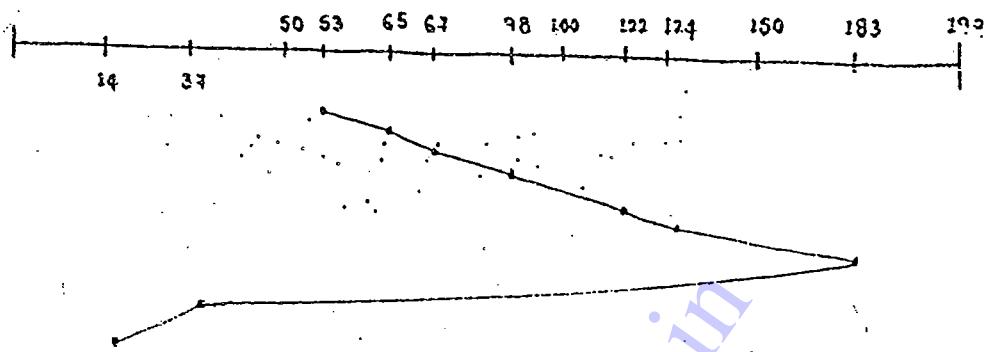
(b) SCAN / Elevator algorithm :-

- * Read-write Head \Rightarrow represents directional scan.
 - * Scanning through the request encountered in certain direction.
- Drawback:
- * It moves to the last track unnecessarily.
 - * Unnecessary causing starvation, by pending the other track in other direction.



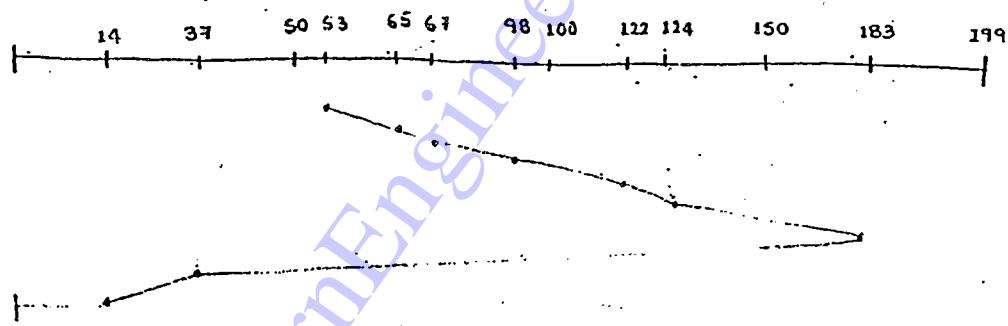
(4) Look :

Look up to last request pending and take reverse turn.



(5) Circular scan (cScan):

Eg :- Disk

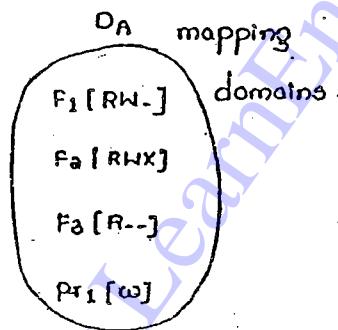


Security vs Protection

Security :

- * Making system secure from external threats. (virus, worms, trojans etc)
- * Security / protection has difference of authorised & unauthorised users.
- * Protection mechanism is implemented using "domain() method"
- * Protection Domain (objects, Rights)

↓
refers either user/process (with Reg. permissions) :

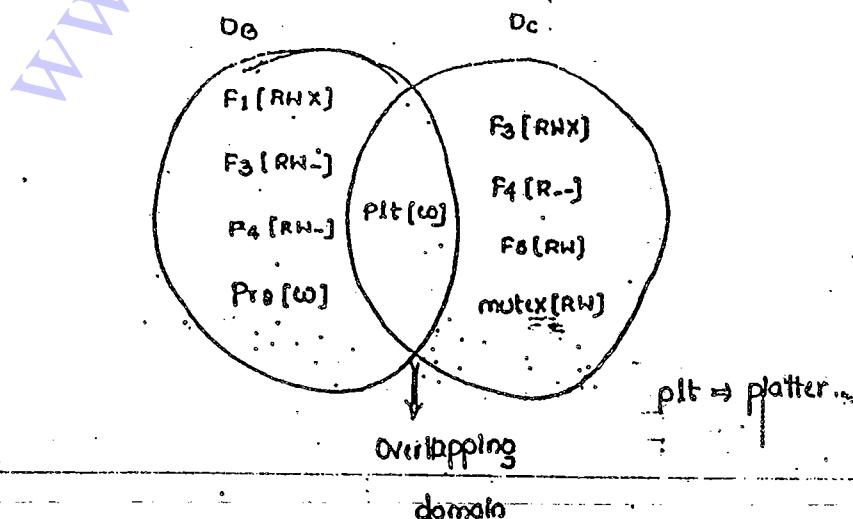


Protection :-

Internal threats
(system level)

Security :

External threats



(1) Protection Domain matrix :

	F_1	F_2	F_3	F_4	F_5	P_{r_1}	A_2	P_{r_3}	P_{l_1}	mutex
D_A	RW-	RWX	R--	-	-	W	-	-	-	-
D_B	RW		RW	RW			W			
D_C			RW	R	RW					RW

(Printer)

$n \times m$

$n \rightarrow$ no. of domain

$m \rightarrow$ no. of objects

entry \rightarrow permission.

= They support direct access. (so, they are faster)

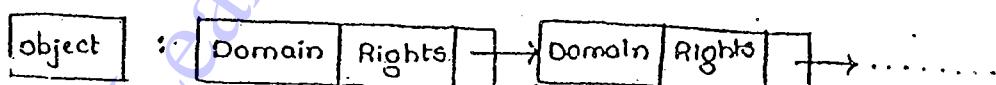
Drawback : space wastage (called sparse matrix).

↓
most are null entries.

To overcome

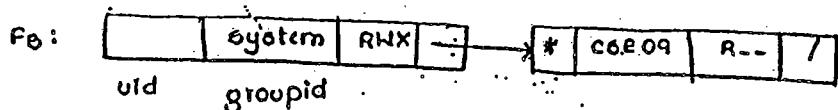
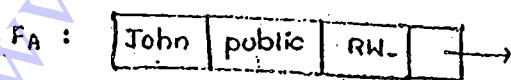
↓

(2) Access Control List (A.C.L) mechanism :



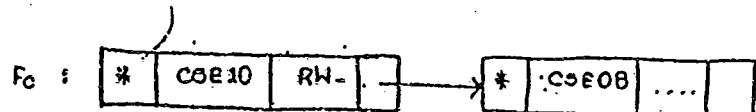
linked lists headed by objects

Used in : UNIX / LINUX



(domain) \Rightarrow

file used by all ese people.



$F_D :$ \Rightarrow All users of comp.
& use people

(3) Capability List (c-list)

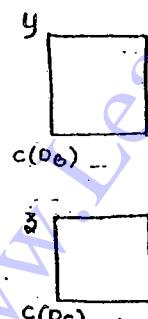
Type	Rights	Name/Address	→ stored in fixed place
FILE	RW-	F_1	
FILE	RWX	F_2	
FILE	R--	F_3	
PRINTER	W-	Pr_1	

Capability (DA) windows

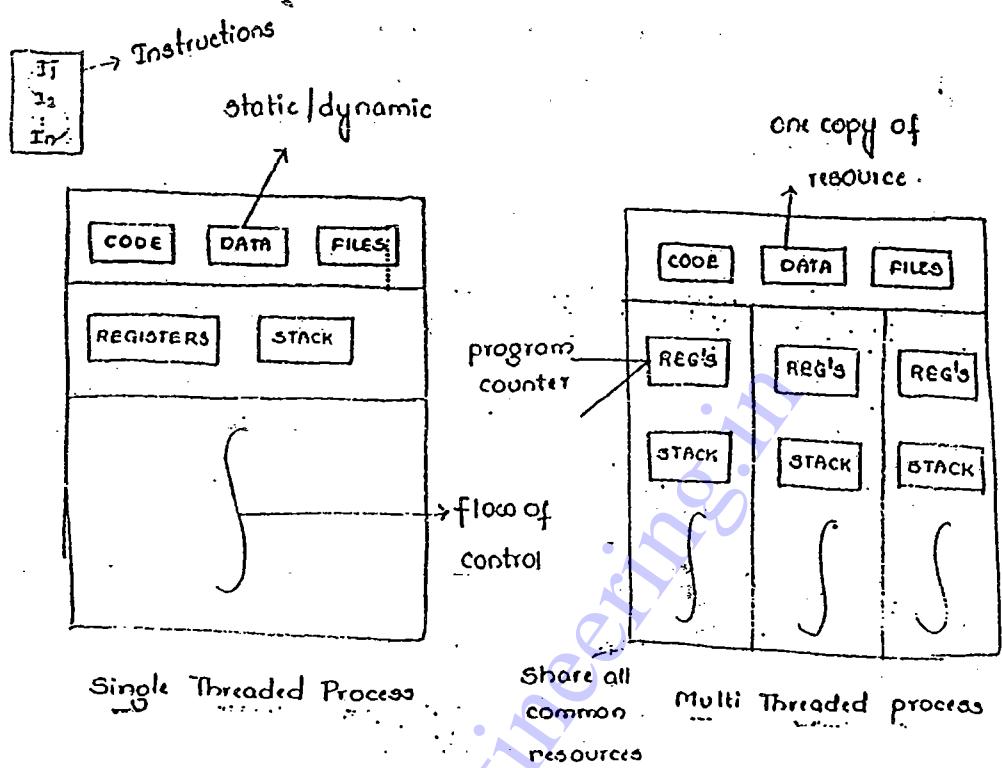
protection mgmt follows

c-list

Domain	Address	(capability)
DA	x	
DB	y	
DC	z	

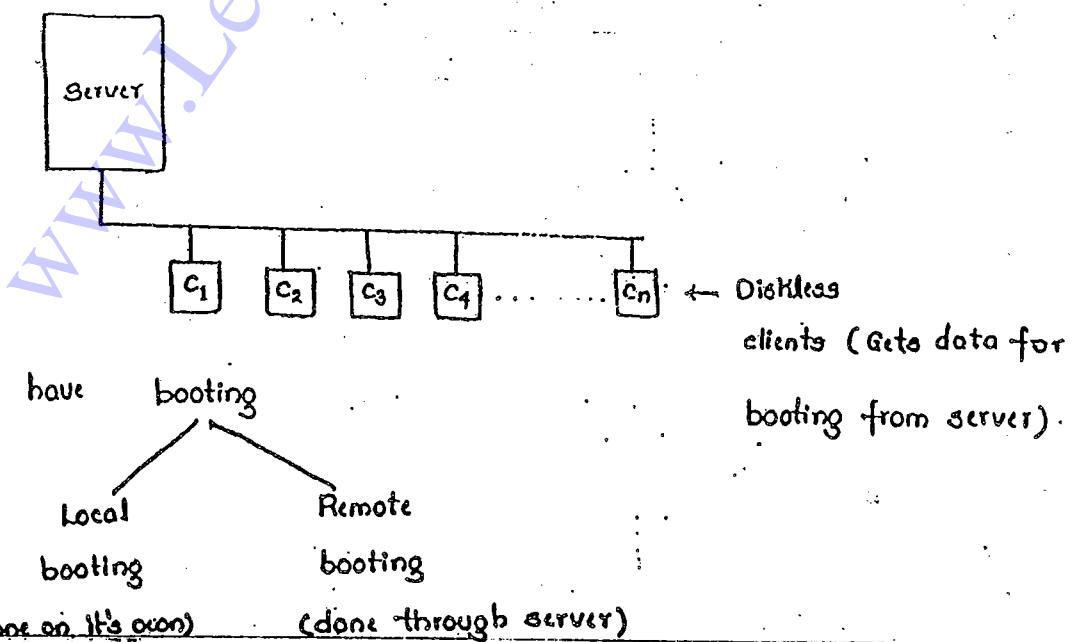


Threads & Multithreading



Thread - Light weight process.

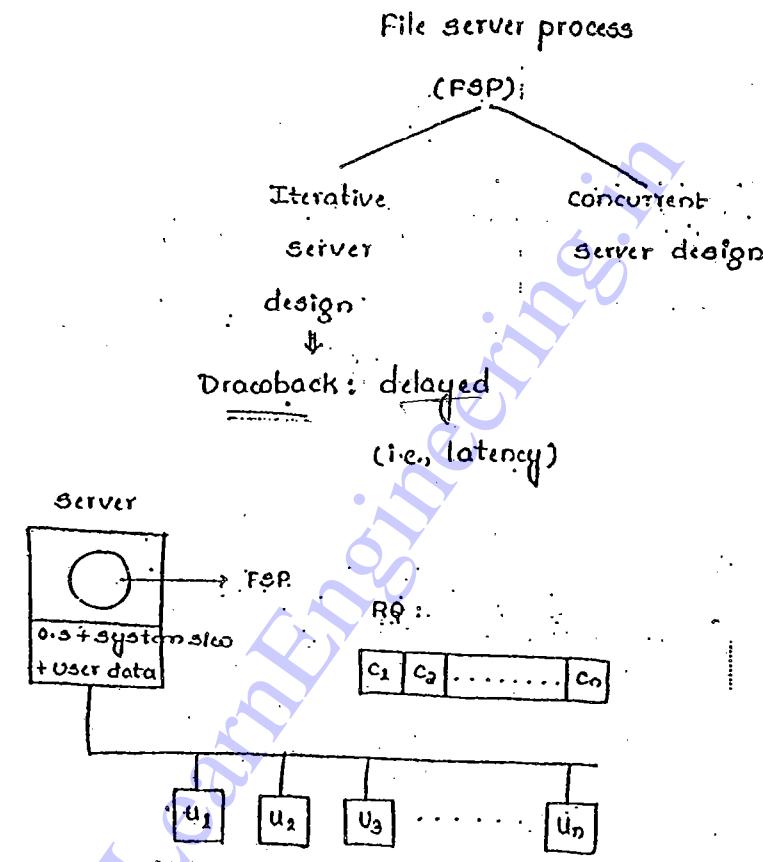
LAN (File Server process) :-



$$| TCB | < | PCB |$$

74

- * If server is alive, it transfers the kernel code to the client to get booted, which is called as "remote booting".



- * Concurrency is achieved through multi-process mechanism (using forks).
 - * The same code of server (parent) get copied in the childs (clients), i.e., duplication of code is done at all the clients.
- Drawback :
- * wastage of resources :
 - * Since code is duplicated (i.e., 'K' copies of code, data, files and accessed by all the clients. Same functionality is processed by all clients, instead of single copy and get shared by them).

In non-sharing clients, every independent process must have program counter, General purpose registers and stack for each, coherence as a single copy of these can be accessed by all the clients in sharing.

Benefits of multithreading

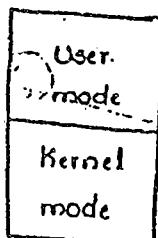
- * Resource sharing
- * cost effective : Economical
- * Improved performance.
- * Achieve parallelism (multi-CPU).
- * $|TCB| < |PCB|$
- * As processes have PCB's, Threads also have TCB's (Id, state, priority)
 - * Attributes that can't be shared are stored in TCB (i.e., Id) and which are shared are stored in PCB.
- * Context-switch becomes fast and less context-switch overhead. So, there is improved performance.
- * Since, it satisfies (5) & (6)th conditions, a thread is called as "Light weight process".

Types of Threads



Threads which are created by user

(at the)



- * Threads, which have packages, libraries.

Eg : Java threads (It won't require O.S support)

- * O.S visualizes it as a process.

* It is transparent to the O.S, that it is only viewed as a process, but not as a thread.

* Java Virtual machine (JVM) allocates the time to the threads.

* User level Thread switching (U.L.T.S) is faster than Kernel level Thread switching (K.L.T.S), because for every processing, there is no need to go to Kernel always, everything is managed at user level.

Drawback :-

* When a process requires I/O, the total process gets blocked instead of a single thread, because of transparency.

* To overcome this drawback, we move to kernel level threads.

* Threading management can be handled by Kernel level.

* If one thread requires I/O, then that particular process only gets blocked but not the others.

* Context switching also done through Kernel level. So, it is slightly slow compare to user level thread.

The entire man

is done by thread
& environment

library package

Pthread :

- * Portable operating system interface for UNIX \Rightarrow POSIX

Monitors

Synchronization

mechanisms

with

Busy waiting
(spinlock)

without

Busy waiting
(blocking)

Spinlock \Rightarrow busy waiting.

{ livelock \Rightarrow states of process are always ready (on running).

Deadlock \Rightarrow states of process are always blocked

monitors :

It is a collection of procedures, variables and data structures, that are all group together in a special kind of modular package.

- : Procedures running outside the monitor, cannot access the monitor's internal variables & datastructures .. However, they can activate monitor's internal procedures.
- * Monitors have an important property that "only one process can be active at any time".

Producer-consumer problem:

```

monitor producer.consumer;
begin
    integer count = 0;
    condition full, empty;
    Procedure Enter
    begin
        if(count = n)  wait(full);
        Buffer[in] = itemp;
        in = (in+1) mod n;
        count = count + 1;
        if(count = 1)  signal(empty);  $\Rightarrow$  // wakeup consumer
    end
    Procedure Remove
    begin
        if(count = 0)  wait(empty);
        itemc = Buffer[out];
        out = (out+1) mod n;
        count = count - 1;
        if(count = n-1)  signal(full);
    end
    Procedure producer
    begin
        cohile (True)
        begin
            Produce-item(itemp);
            Producer.consumer..enter;
        end
    end
    Procedure consumer
    begin
        cohile (True)
        begin
            Producer.consumer..remove;
            process item/itemc;
        end
    end

```

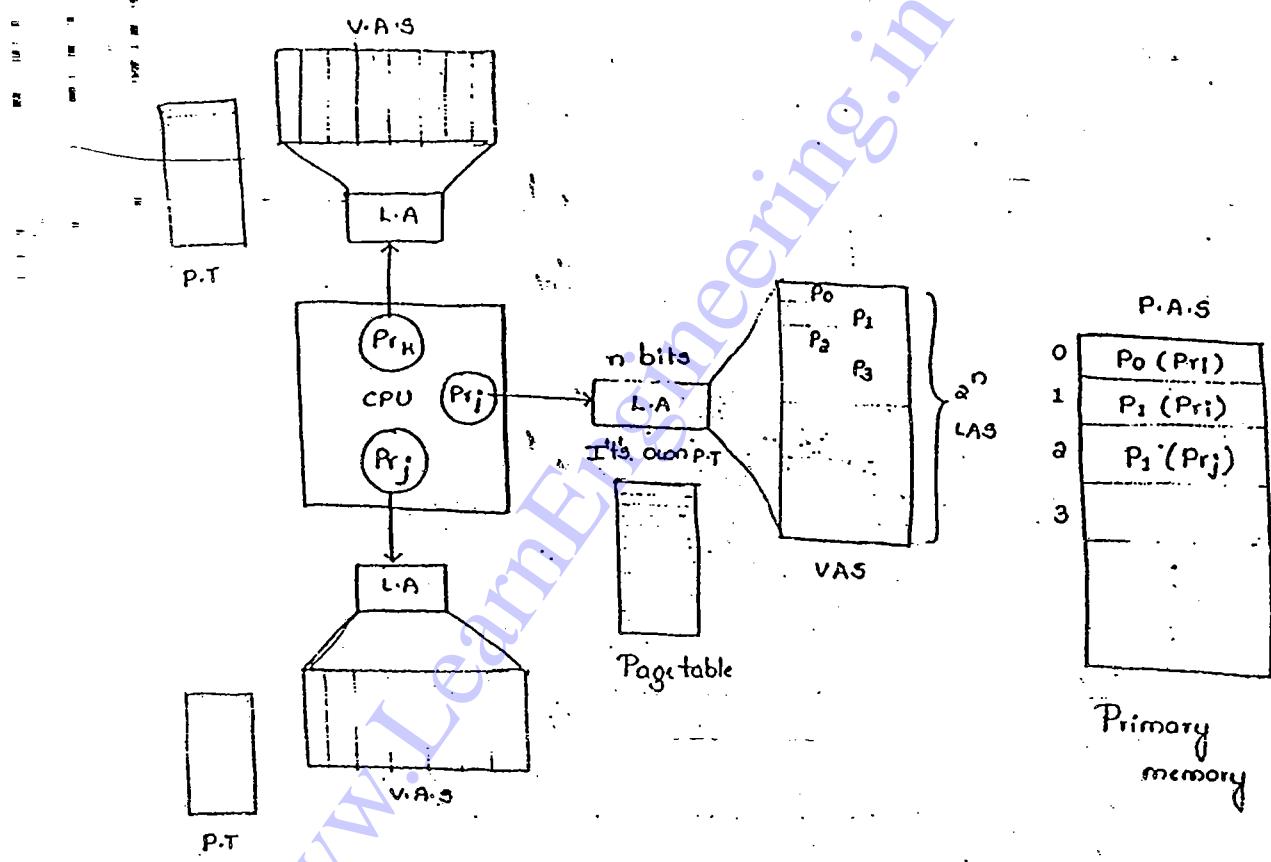
$\text{mutex} = 1 \Rightarrow$ no process is inside

$= 0 \Rightarrow$ some process is accessing.

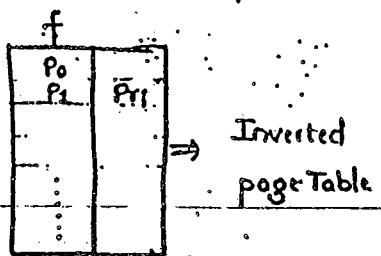
Initially $\text{mutex} = 1$;

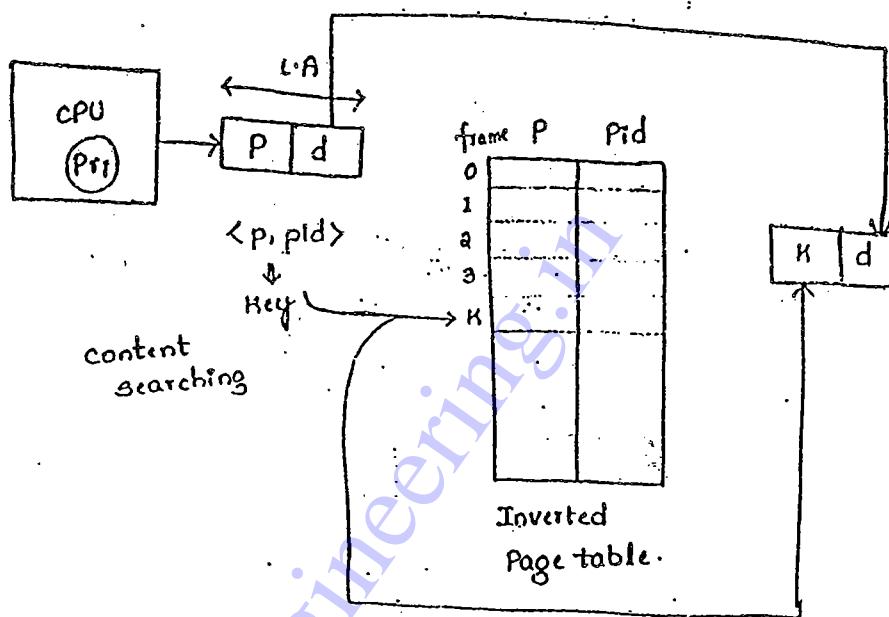
then DOWN(mutex) ;

Inverted paging :-



- All the processes maintain single primary memory each.
- Within the page table, frames are stored within the page. The Inverted page table contains pages get stored in frames.





* The Inverted page table maintains the "key" generated by adding $\langle p, pid \rangle$, thus, by referring the key, the particular page gets accessed, and the same offset is considered. This reduces the time for searching the content.

www.LearnEngineering.in

08/07/2020

Thursday

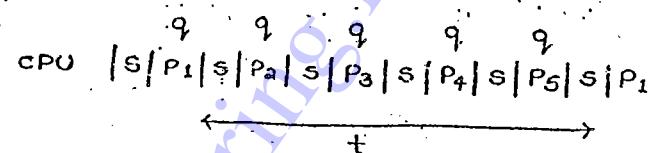
Page no : 34

(1) n - processes

context switching - 5 seconds.

$$T \cdot Q = q$$

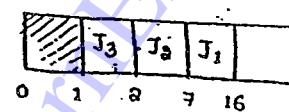
$$n = 5$$



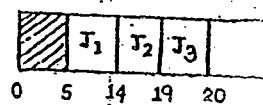
$$q \leq \frac{t - ns}{n-1} \quad (\text{at least})$$

$$q = \frac{t - ns}{n-1} \quad (\text{exactly})$$

(4)

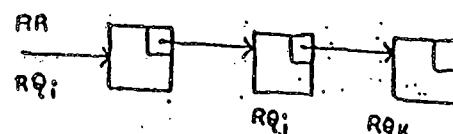
(a) $\{3, a, 1\}, 1$ 

T	AT	BT
4	0.0	9
a	0.6	5
3	1.0	1

(d) $\{1, 2, 3\}, 5$ 

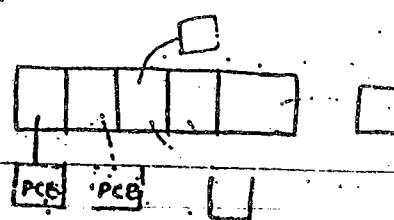
P.no: 4a

(1)



Priority of process increases

Drawback : starvation.



P.no: 44

(9)

$$(a) \text{ CPU Efficiency } (\mu) = \frac{T}{T+S} \quad (TQ = \infty)$$

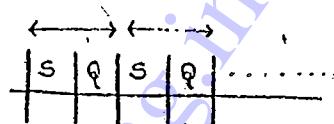
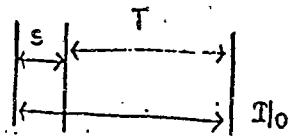
$$(b) \mu = \frac{T}{T+S} \quad (TQ > T)$$

$$(c) \mu = \frac{Q}{Q+S} \quad (S < Q < T)$$

$$(d) \mu = \frac{Q}{QS} \quad (Q = S)$$

$$= .50\%$$

$$(e) \mu = 0 \quad (Q = 0)$$



(10) Trap & Interrupt :

Trap : Indicates error condition always

Interrupt : Indicates error condition (sometimes but not always)

27
2010/07/2010

Thursday

Pno: 101

(6) $n = 4, m = 3, (r_1, r_2, r_3) = (2, 3, 8)$

n	Alloc.	Request.	Available.
P ₁	1 1 0	1 0 0	0 0 1
P ₂	1 0 1	0 1 1	safe = {P ₃ , P ₂ , P ₁ , P ₄ }
P ₃	0 1 0	0 0 1	
P ₄	0 1 0	0 0 0	

4/07/2010

Saturday
=

P.no: 140

5) min. size of partition = max. path length (in sizes of modules)

All the paths are considered as per their sizes

and the maximum one is represented.

P.no. 142

$$(20) \quad e \Rightarrow \text{frame no.} \quad \text{PAS} = 64 \text{ MB}$$

$$VAS = 32 \text{ bit}$$

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

$$\text{P.T Size} = N * e$$

$$N = \frac{\text{VAS}}{\text{PS}} = \frac{2^{32}}{2^{12}} = 2^{20} = 1M$$

$$M = \frac{64 \text{ MB}}{4 \text{ KB}} = 2^{14} \approx \text{abytes}$$

$$m = \frac{64 \text{ MB}}{4 \text{ KB}}$$

$$= \frac{2^{26}}{2^{12}} = 2^{14}$$

$$e = N * c$$

$$= 1M * 20$$

$$= 2 \text{ MB}$$

P.no: 152

$$(17) \quad LAG = 8 \text{ KB} \times (8 \text{ pages of } 1024 \text{ words})$$

$$m \text{ size} = 32 \text{ K} \text{ (80 frames of 1024 words)}$$

$$LAG = 8 \times 1024$$

$$= 8 \text{ KB} \times 13 \text{ hits}$$

$$\text{PAS} = 32 \times 1024 = 32 \text{ K} = 16 \text{ hits}$$

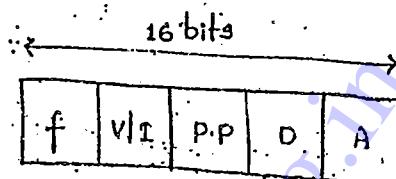
Sol 07/2010

friday

P.no: 160

(i) (b)

$$LAS = PAS = 2^{16} B$$



P.T entry

$$f = \frac{2^{16}}{2^9} = 2^4 \quad \text{page size} = 512 \\ = 2^9$$

P.no: 165

(d6) (a) Stack:

Since, only top of the stack is accessed, so, there is a less no of page faults.
 ∴ It is good.

(b) Hashed symbol table:

It distributes identifiers across the table. So, it generates more no. of page faults.

(c) Sequential search & Binary search:

↓
 Likely to cause more page faults than seq. search.

(e) pure code:

Read only code (non-modified code).

∴ It is good to have non-modified in demand paged environment.

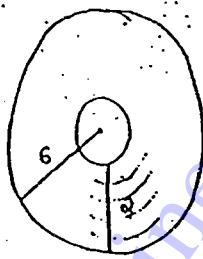
(f) vector operations :-

Array (contiguous allocation) \Rightarrow good

(g) Indirection :-

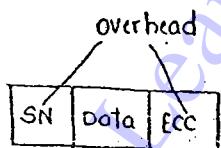
Linked lists (non-contiguous allocation) \Rightarrow bad.

(8)



$$\text{no. of tracks} = \frac{(6-2) \text{ cm}}{0.02 \text{ cm}}$$

= 400 tracks on one surface.



$$400 + 96 = 4096 = 4 \text{ KB} \quad \text{Each track has } 80 \text{ sectors}$$

$$(400 * 80 * 4 \text{ KB}) * .8 \text{ surfaces}$$

$$\Rightarrow 2^9 \times 2^5 \times 2^{12} \times 2^3 = 2^{49} = 512 \text{ MB}$$

$\approx 500 \text{ MB}$

(b)

$$R = \frac{60}{8600} \text{ sec}$$

$$\frac{60}{8600} \text{ sec} = 80 * 4 \text{ KB}$$

$$8600 * 80 * 4 \text{ KB}$$

60

15.

P.No: 184

(4) program size = 64 KB

T.S = 80 KB

R = 80 ms

80 ms = 8 KB

S.T = 30 ms

? = ? KB

P.S = 2 KB

no. of pages = $\frac{64 \text{ KB}}{2 \text{ KB}} = 32 \text{ KB}$

(a) Time for loading one page = $30 \text{ ms} + 10 \text{ ms} + \frac{20 * 2}{32} \text{ ms}$

(b). page stored on track to go to that track, we have seek time

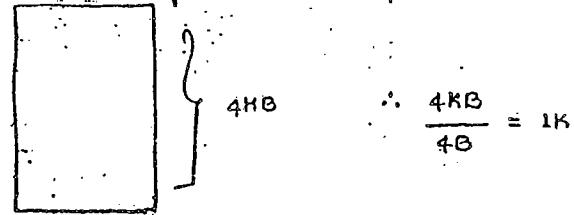
= $30 \text{ ms} + 10 \text{ ms} + \frac{20 * 4}{32} \text{ ms} * 16$

04/08/2010

Wednesday

(3) (a) max. no. of files \Rightarrow depends on directory

Directory = 8 bits \Rightarrow 4B



(b) DBA = 8 bits

 \therefore 256 blocks possible for a file of 8 bitsBut, there are 2 data blocks. So, $= 256 - 2 = 254$

(254 * 4 KB) \approx 1 MB

P.no: 183

(1) $B = \text{blocks}$ $S = DBS$ $D = DBA$ $F = \text{free blocks}$ $DBA = D \text{ bits}$

no. of bits = B

free = f

Bit map space consumption is 'B' bits (depends on blocks)

Size of free list (space) = $F \times D$

$$\therefore FD < B$$

 $S = DBS$ $B = \text{blocks}$ $D = DBA$ $F = \text{free blocks}$ Disk size = $B \cdot S$ (maximum
possible)

20MB . 64MB

Block = 80K

OS = 80MB

DBA = 16 bits

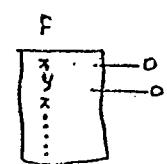
DBS = 1KB

 $2^D \geq B \quad (B \cdot S \leq 2^D)$

(max.)

 $2^{16} = 64KB * K = 64MB$

(max.)



P.no: 179

(1) b

(2) b (since, it takes more access time, if placed anywhere).

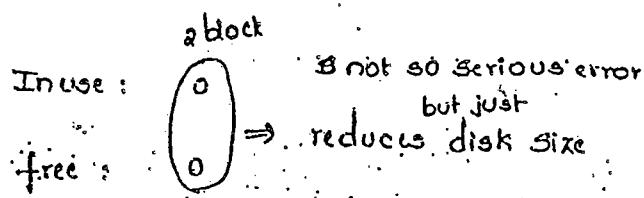
(3) a

(4) b

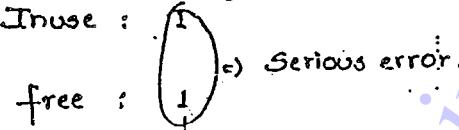
(5) If there is only one process, then there is no matter whether to use any disk scheduling approach.

So $\Rightarrow 0\%$.

(6)



6 block



Since it is free, it may also given to other file.
but a file is already being accessing. So, it is
considered as a serious error.

To solve this error :-

copy the Inuse file to some other block and use it.

P.no: 37

(Q5) (a)

(Q6) (a)

Q P.no: 15a

$$(a) \quad VA = 48 \text{ bits}$$

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

$$\text{no. of pages} = 8K = 2^{13}$$

$$\text{no. of entries} = \frac{2^{48}}{2^{13}} = 2^{35} = 32G \text{ entries}$$

Inverted page table:

$$= \frac{2^{32}}{2^{13}} = 2^{19} = 512K \text{ entries}$$

26th June, 2010

Saturday

Operating Systems

Topics :

(1) Background

(2) Process management

- * Process concepts

- * CPU scheduling

- * IPC & Synchronisation

- * Concurrency

- * Deadlocks

- * Threads

(3) Memory management

- * Concepts

- * RAM chips

- * Techniques

- * Virtual memory

(4) File system & I/O management

- * Interface

- * Device characteristics

- * Implementation issues

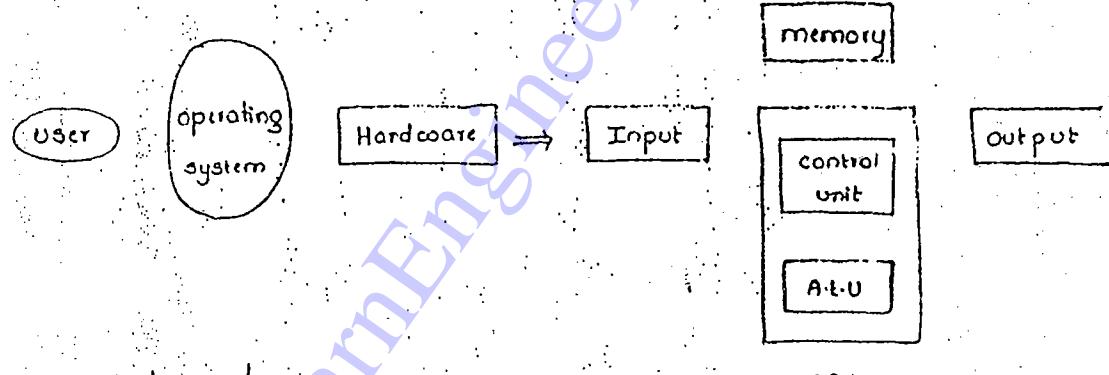
(5) Protection mechanisms

(q). What is an operating system ?

Ans: Interface between user and Hardware.

- * Resource manager
- * control program(s)
- * A set of utilities to simplify application development.
- * Acts like a government.

Von-Neumann Architecture



i) Control signals

ii) Mathematical operations (Data, Register)

$$\text{clock cycles}$$
$$c = a + b$$

Load R_i m(a)

Load R_j m(b)

Memory :

* Primary (main memory, physical \Rightarrow RAM, ROM, cache, Registers)

* Secondary (Auxiliary \Rightarrow HD, DVD, Pendrives)

* Processor cannot fetch data directly from secondary memory. Instead Instruction Register (IR) is used, from where instructions are fetched and executed sequentially.

* If IR requires any operands, then it fetches the operands and executes as :

- * Fetch cycle
 - * Decode cycle
 - * Execute cycle
 - * Interrupt cycle
- Instruction cycle

* The communication among user and CPU can be represented by Von-Neumann architecture.

Main Objective of operating system:

* It takes the complete control of Hardware, and create a platform, which is easy to user to write different applications.

Command Interpreter:

* A program that interprets the commands of the user.

* It acts as interface between the User and Kernel.

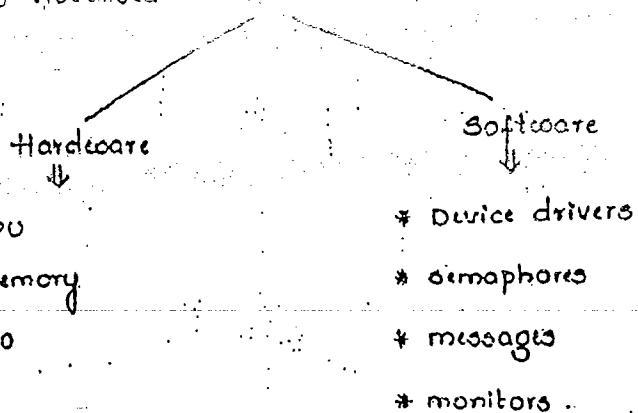
* These are of two types :

* Text based OS (shell) : DOS, UNIX, NETWARE

* GUI based OS (Windows, Linux)

It. Operating system is a set of command interpreter.

* Operating System is visualised as resources.

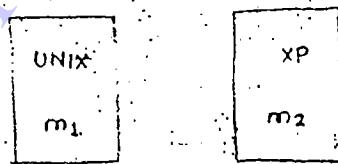


Functions & Goals:

Goals:

- (1) convenience (user friendly)
- (2) Efficiency (resource utility)
- (3) Reliability & Robustness
- (4) Scalability (Ability to evolve)
- (5) portability (different platforms)

Consider an example:



To find a particular content in "unix" o.s., we must write its syntax.

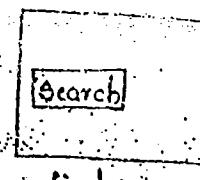
Eg:- `$find`

`/-name "textic"`

so, that it requires exact syntax, but it takes less time to execute the given statement.

To find the same in "xp" o.s., it is an easy process that any user can

approach through (i.e., clicking "start" button & then "Search")



- * It takes more time than unix to execute, but it is more familiar to all kinds of user to operate.
- * So, "convenience" is more important than "efficiency".
- * In some real time systems; "Efficiency" is more important than "convenience".

First Generation

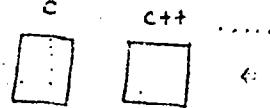
No operating System

Card Readers, Punch cards

manual Intervention.

Second Generation

Magnetic Tapes



Batch processing

↓
Clubbing similar programs
at one place & processed
one at a time.

Third Generation

Hard Disk Technology

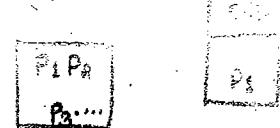
Uniprogramming

(PDP)

main memory

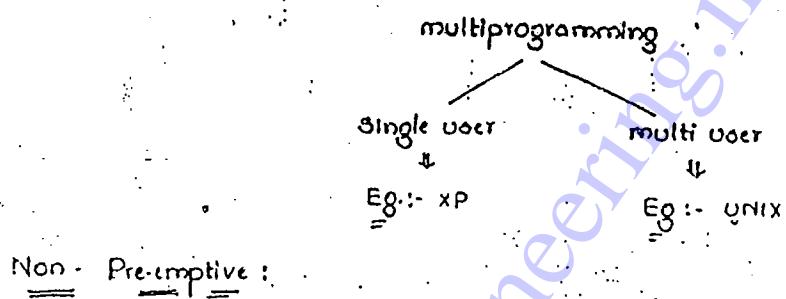
multiprogramming

(UNIX, XP)



Multiprogramming:

- * Ability of operating system to hold multiple ready-to-run programs in the main memory so that if the running program requires I/O, the CPU can be switched to another ready program.
- * Multiplexing of CPU among ready programs in memory.



Non-Pre-emptive:

- * Running process is released voluntarily (on its own) as follows:
 - * Completion of process
 - * I/O request

Pre-emptive:

- * Running process can be forced to release based on:
 - * Priority
 - * Time sharing

Time sharing systems (pre-emptive):

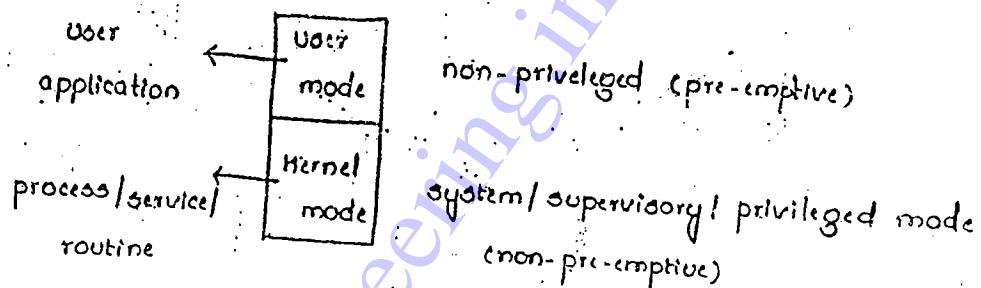
Eg.: - Windows XP

first version of windows : 3.0

3.1 } non-preemptive
3.11 }

Architecture of multiprocessor support:

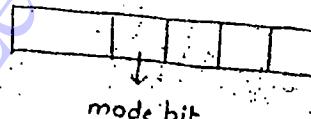
- (1) Direct memory Access (DMA) \Rightarrow Since computation & I/O required, we need DMA.
- (2) Address Translation
- (3) Atleast two modes of CPU execution.



- * In Kernel mode, the process have atomic execution (without pre-emption)
- * In user mode, the process have non-atomic execution (with pre-emption)

Program status word (PSW):

It represents the mode, in which the user operates in.



0 - User mode

1 - Kernel mode

System call Vs Library call.:

```
main()
{
    int a,b,c;
    scanf("%d %d", &a, &b); (BSA)  $\Rightarrow$  Branch & save
    User mode
    c = a+b;
    Address (BSA)
    for (i=0; i<10; i++)
        printf("%d", i);
}
```

* `fork()` creates child process and the return of `fork()` is done in User mode because it is a part of User mode and the conversion takes place (from User mode to System mode).

* Execution of Supervisory call (SVC) involves the software interrupt, which handles Interrupt Service Routine (ISR).

* When a SVC is generated, ISR changes to 1 (non-preemptive mode). Then after the completion of all the functions in Kernel mode, again an SVC is generated, where ISR value is changed from 1 to 0 (pre-emptive mode).

System Call Interface (SCI) ↳ UNIX

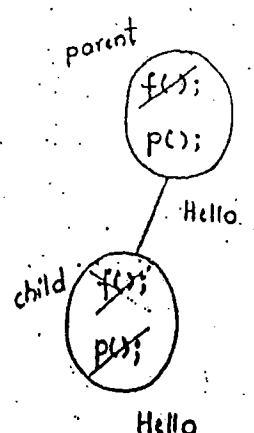
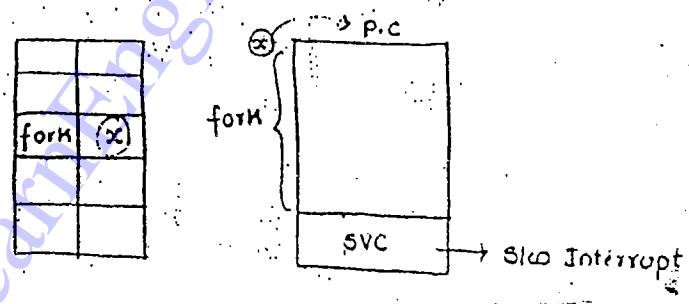
In windows O.S, SCI is considered as API.

ISR → changes
mode bit
&
Dispatch the
table

```
(1) main()
{
    fork();
    printf("Hello");
}
```

Output : Hello

Hello



(a) main()

```
{
    fork();
    fork();
    printf("Hello");
}
```

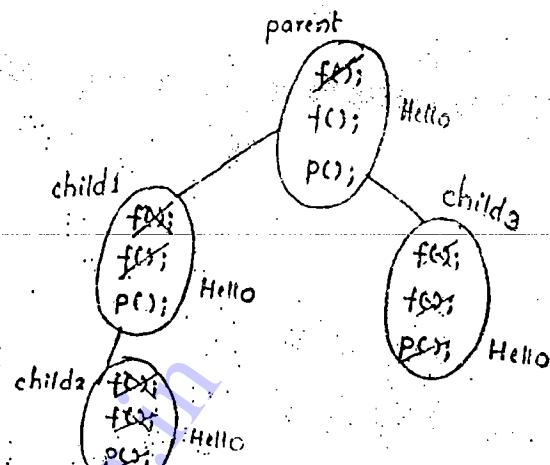
* If 1 fork \Rightarrow 2 prints

2 forks \Rightarrow 4 prints

3 forks \Rightarrow 8 prints

n forks $\Rightarrow 2^n = \text{Total}$

$$\text{no. of child} = 2^n - 1, \text{ parent} = 1$$



Output :

Hello
Hello
Hello
Hello.

(b) main()

```
{
    int i, n;
    for(i=1; i<n; ++i)
        fork();
}
```

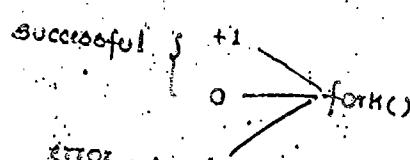
(a) $n-1$

(b) $n!-1$

(c) n^2-1

(d) 2^{n-1} (Cause n forks)

* Any system call may return either +ve, 0, -ve values.



child parent error

```
main()
{
    int id;
    id = fork();
    if(id == 0) // child
    {
        // child code
    }
    else if(id > 0) // parent
    {
        // parent code
        else printf("Error in creation");
    }
}
```

```
id=fork();
if(id==0)
{
    // child code
}
else if(id>0)
{
    // parent code
}
```

```
child id 0
id = fork();
if(id==0)
{
    // child code
}
else if(id>0)
{
    // parent code
}
```

Difference between Kernel mode & user mode:

Compilers, editors and similar application-independent programs are not part of the O.S. even though they are typically supplied by computer architectures. This is crucial, but subtle point. The O.S system is that portion of the software that runs in Kernel mode. It is protected from user tampering by hardware.

compilers & editors runs in user mode. If a user doesn't like a particular compiler, he/she is free to write their own clock interrupt handler, which is a part of O.S.

0st July, 2010

Thursday

Process concepts

Program Vs Process :

- * Program under execution.
- * Unit of execution
- * Schedulable / Dispatchable unit
- * Instance of program
- * Central locus of control (CoC)
- * Animated spirit

* Executing program resides in secondary memory.

Program

Process

- | | |
|---|--|
| <ul style="list-style-type: none">* In secondary memory* Without resources* passive | <ul style="list-style-type: none">* In main memory* Utilization of resources* Active (Alive) |
|---|--|

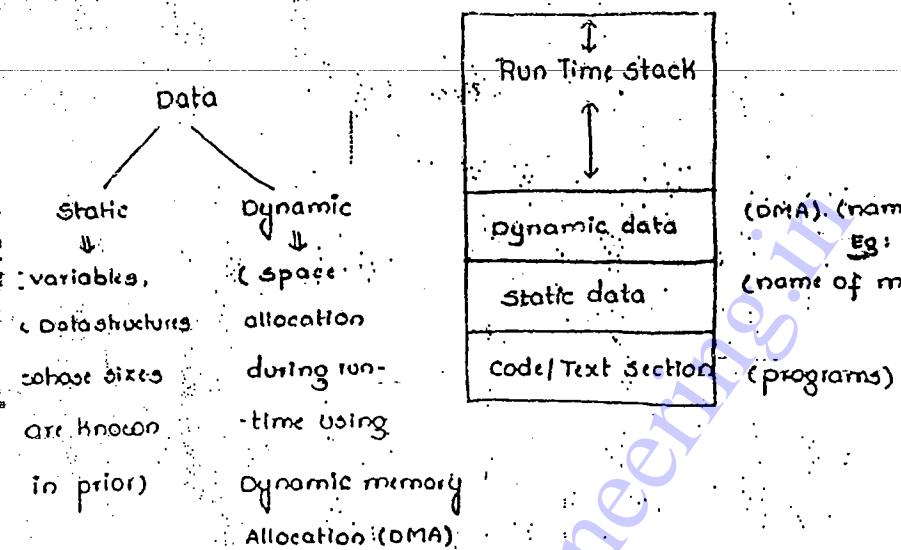
Formal definition of process :

- * Developer / Designer views process as a simple Datastructure.
- * Every Datastructure is always defined with four parameters:
 - * Definition
 - * Representation (process structure)
 - * Operations
 - * Attribute (pushes, pops) Implementation

Process Structure (Representation) Implementation in main memory

Abstract view of process:

* program consists of data and instructions.



(DMA) (name of memory area)
Eg: Heap
(name of memory) Eg.: Stack

RunTime stack \Rightarrow Activation records are maintained wherever function

calls are generated.
Every process

maintains this for storing addressed (recursive) use this stack.

stack

Operations :

* Essential resources required for stopping the execution of process, then some resource utilities are used.

* When a program is loaded in main memory, resources are allocated and then CPU schedules all the process.

* Create (resource allocation)

* Scheduted (CPU)

- * Execution (CPU)

- * Blocked (I/O)

- * Resume

- * Suspend()

Attributes :-

- * Process ID (pid)

- * Priorities

- * Process state

- * Program counter

- * memory limits

- * list of files

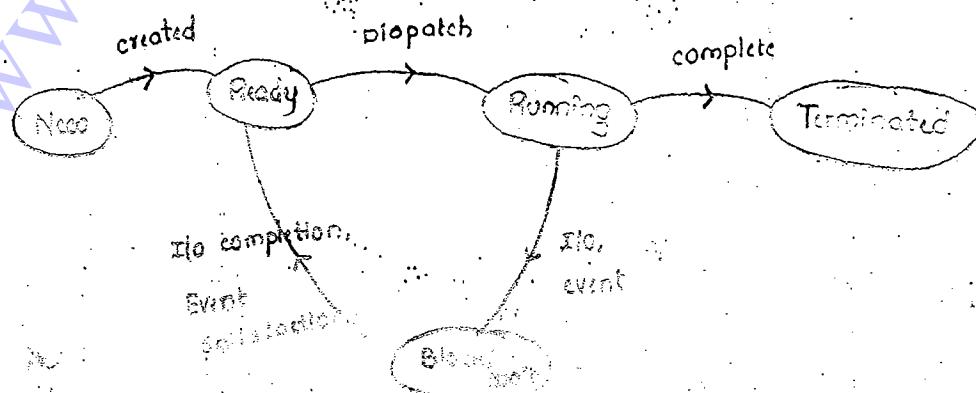
- * list of devices

- * Type & size (in bytes)

- * Protection.

- * All the process attributes are stored within the process control block (PCB)

Process State :-



Process control Block (PCB)

ID of process

Pid	
state	Priority
PC	GPR
mtr limits	files
devices	i

process context / process

environment.

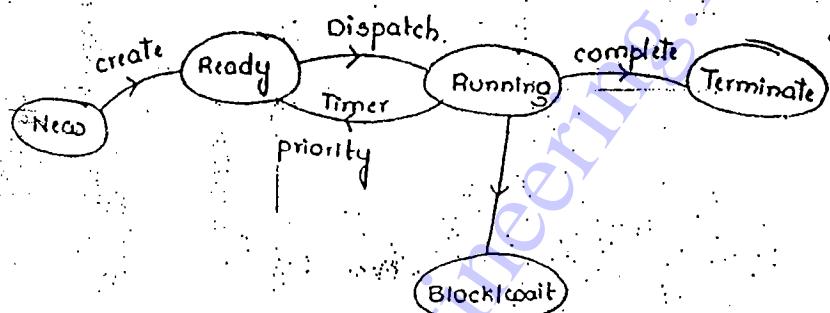
Above diagram represents :

- (a) Uniprogramming (no "ready" is present)
 - (b) pre-emptive multiprogramming
 - (c) non-preemptive multiprogramming
 - (d) multiple CPU based OS
-
- ```

graph LR
 New((New)) --> Run((Run))
 Run --> Term((Term))
 Run --> Block((Block))

```

### Pre-emptive multiprogramming



Degree of multiprogramming = no. of processes

Degree ↑, when process ↑, so, we suspend process whenever required.

process suspension  $\Rightarrow$  swapped out on temporary basis and later on resumed.

There is no suspension for the following process states : ↑ performance

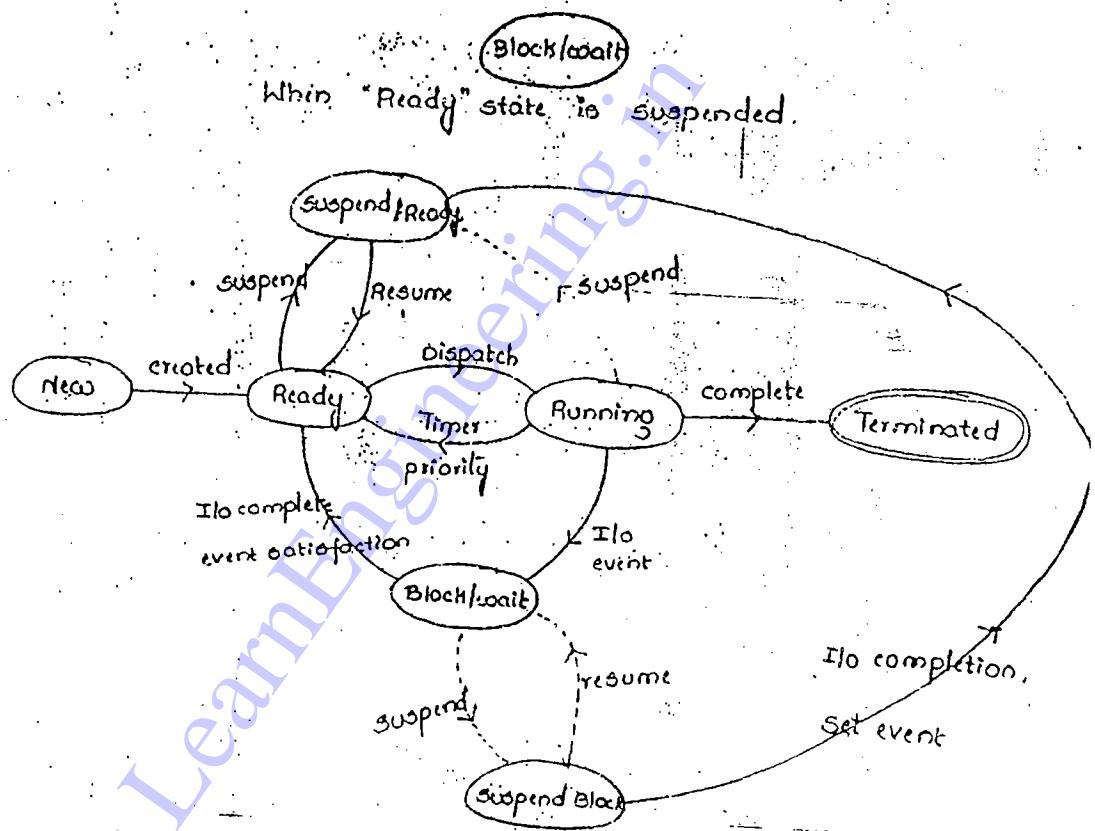
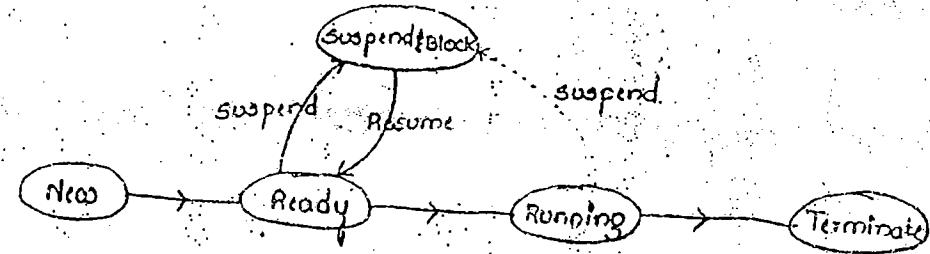
\* New

\* Terminate

Most desirable state for suspension is : "Ready" state.

Because, it is waiting for I/O (or other events), so, when suspended it gets confused.

If "Ready" process is suspended, then it moves to "suspend" state.



- \* Only Ready, running, Block states  $\Rightarrow$  can be suspended
- \* processes waiting for I/O operations are not suspended (dangerous). If get suspended, it moves to suspend block state.
- \* Suspension of running process are not preferred always. If suspended, it releases the resource prematurely.

so, ready block is the only option  
temporarily stored in secondary memory.

(Q) consider a system with  $n$  CPUs &  $m$  processes (where  $n \geq m$ ) calculate lower bound and upper bound on the process states.

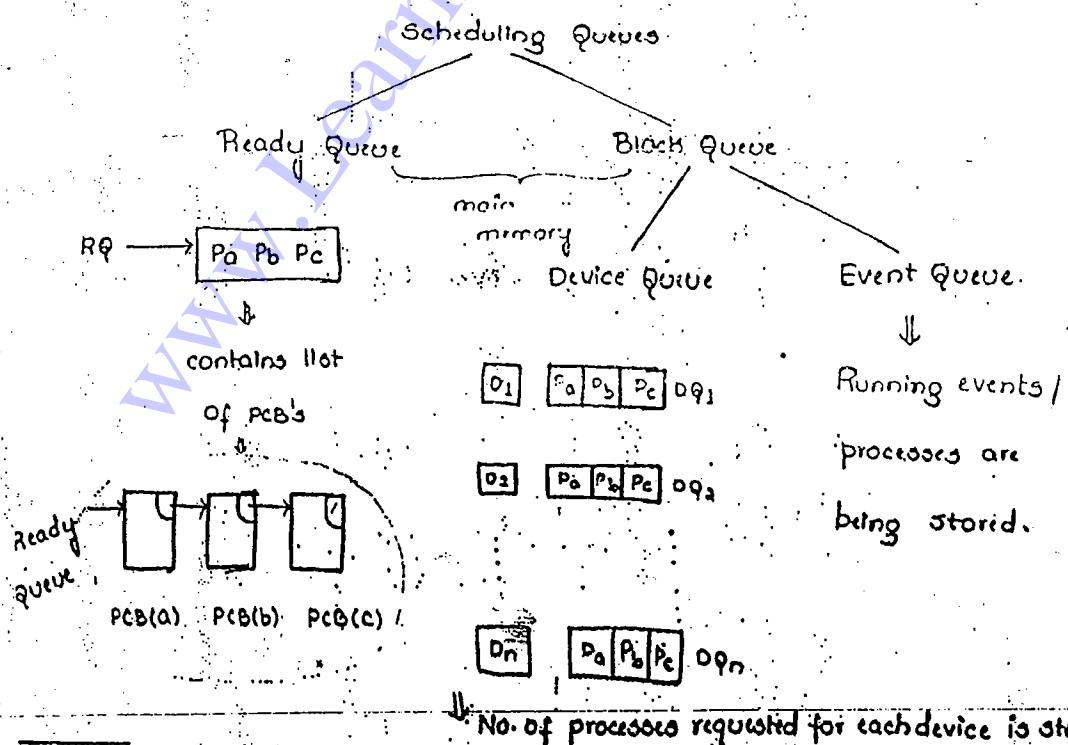
- \* Ready
- \* Running &
- \* Block state;

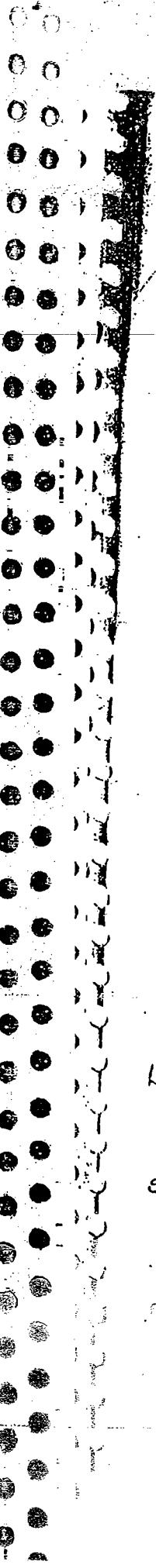
Ans :- maximum Ready state processes are =  $m$ .

maximum running state processes are =  $n$ .

|         | Min<br>LB | Max<br>UB |
|---------|-----------|-----------|
| Ready   | 0         | $m$       |
| Running | 0         | $n$       |
| Block   | 0         | $m$       |

Scheduling Queues & State - Queueing Diagram :-





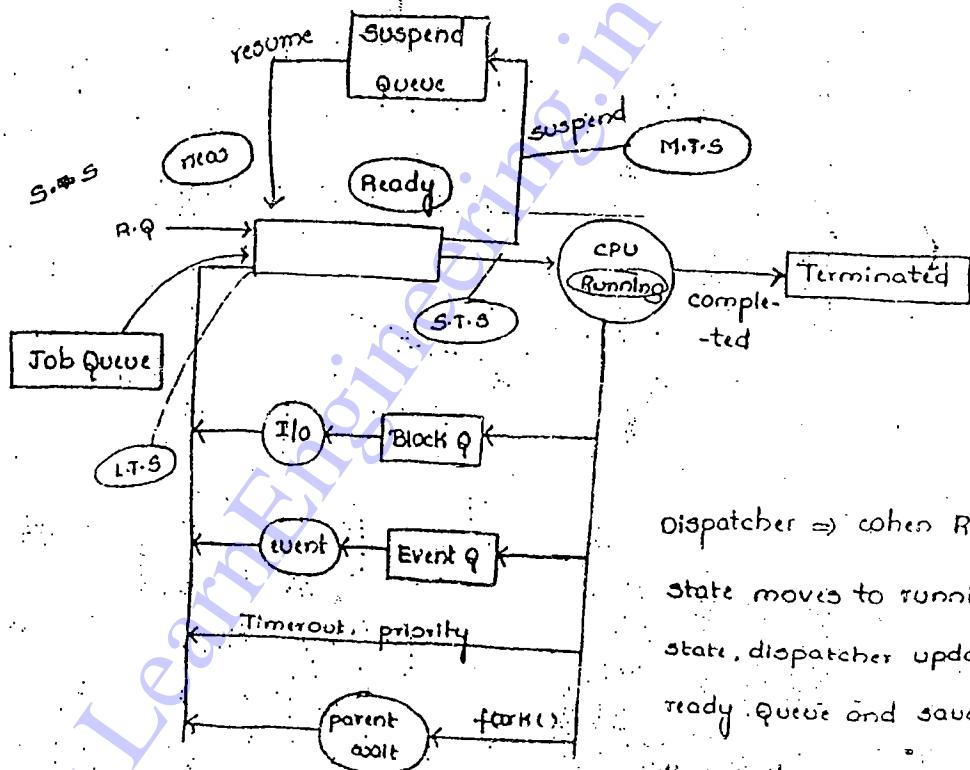
Scheduling Queues

Suspend Queue  $\Rightarrow$  It contains suspended processes.

Secondary memory

Job Queue  $\Rightarrow$  It contains PCB's of Job Queue.

Process State Queue diagram :-



Dispatcher  $\Rightarrow$  when Ready

state moves to running

state, dispatcher updates

ready queue and saves

the position.

Long term scheduler - loading new programs from secondary to main memory.

Job Queue

Short term scheduler (CPU scheduler) :-

It decides which process (ready process) should run next.

Middle term scheduler (dead period scheduler)

(It decides which process should be suspended, if required.)

\* Switching and changing the PCB's from one process to another is called "Context Switching".

\* "context switching" has directly impact on volume of information in PCB.

multiprogramming  $\Rightarrow$  long term scheduler.

### CPU Scheduling

CPU scheduler :

\* Decision of which process to run is taken.

Goals :-

\* Maximize CPU utility

\* Minimize Response time, waiting time.

Process Times :

\* Arrival Time (A.T) - Submission time.

\* Waiting Time (Ready Queue / BQ)

\* Burst Time (CPU) - Service Time (S.T)

\* Completion Time (C.T)

\* Turn Around Time (TAT) = completion Time (C.T) + Arrival Time (A.T)

Response Time (R.T) :

\* Time of submission of request by a process, to obtain a result / response (first response)

Deadline (D) :

\* No. of processes completed within the deadline.

| A.T | R.Q | BT <sub>1</sub> | I/O | RQ               | BT <sub>2</sub>  | RQ | BT <sub>3</sub> | C.T |
|-----|-----|-----------------|-----|------------------|------------------|----|-----------------|-----|
|     |     | WT <sub>1</sub> |     | COT <sub>2</sub> | COT <sub>3</sub> |    | WT <sub>4</sub> |     |

→ clock Time

Notation :

(i) n-processes ( $P_1, \dots, P_n$ ) :

(a) A.T( $P_i$ ) =  $A_i$

(b) B.T( $P_i$ ) =  $x_i$

(4) C.T( $P_i$ ) =  $c_i$

(5) Deadline ( $P_i$ ) =  $D_i$

(6)

formulae :

(a) TAT( $P_i$ ) =  $C_i - A_i$

(b) WT · TAT( $P_i$ ) =  $\frac{C_i - A_i}{x_i}$

(c) Avg. TAT( $P_i$ ) =  $\frac{1}{n} \sum_{i=1}^n (C_i - A_i)$

(d) WT( $P_i$ ) = TAT - BT

$$= C_i - A_i - x_i$$

(e) Avg. WT( $P_i$ ) =  $\frac{1}{n} \sum_{i=1}^n (C_i - A_i - x_i)$

(f) Schedule length(L) =  $\max(C_i) - \min(A_i)$

(g) Throughput( $\mu$ ) =  $\frac{n}{L}$

b) Deadline overshoot :  $C_i > D_i$

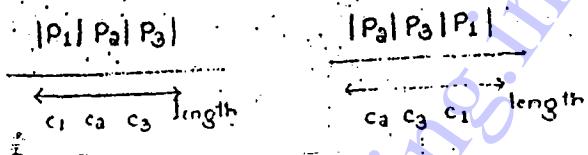
If  $(C_i - D_i) < 0$  under run

If  $(C_i - D_i) = 0$  on deadline

If  $(C_i - D_i) > 0$  overrun.

### Schedule :

n-processes. n=3. ( $P_1, P_2, P_3$ )



07/07/2010

today

### CPU scheduling Techniques

Non-preemptive

pre-emptive

(1) First come first serve (FCFS) :

Criteria : Arrival Time (process)

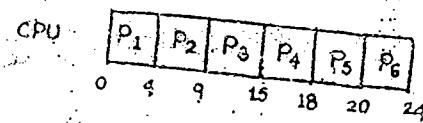
mode : non-preemptive

$$TAT = CT - AT$$

$$WT = TAT - BT$$

| P.NO | A.T | B.T (CPU) | CT | TAT | WT |
|------|-----|-----------|----|-----|----|
| 1    | 0   | 4         | 4  | 4   | 0  |
| 2    | 1   | 5         | 9  | 8   | 3  |
| 3    | 2   | 6         | 15 | 13  | 7  |
| 4    | 3   | 8         | 18 | 15  | 12 |
| 5    | 4   | 9         | 30 | 16  | 14 |
| 6    | 5   | 4         | 24 | 19  | 15 |

Gantt chart :



Arrival Time of P<sub>1</sub> = 0

i.e., at '0' clock time, P<sub>1</sub> arrives

$$P_3 \text{ C.T.} = P_1 \text{ C.T.} + P_2 \text{ B.T.}$$

Eg. :-

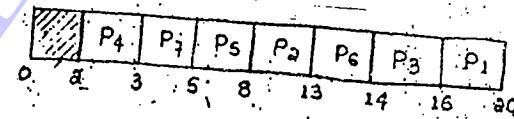
$$\begin{aligned} \text{completion Time (P}_n\text{)} &= \text{completion Time (P}_{n-1}\text{)} \\ &+ \text{Burst Time (P}_n\text{)} \end{aligned}$$

| PNO | AT | BT | CT | TAT | WT |
|-----|----|----|----|-----|----|
| 1   | 8  | 4  | 20 | 12  | 8  |
| 2   | 5  | 5  | 13 | 8   | 3  |
| 3   | 7  | 2  | 16 | 9   | 7  |
| 4   | 2  | 1  | 3  | 1   | 0  |
| 5   | 4  | 3  | 8  | 4   | 1  |
| 6   | 6  | 1  | 14 | 8   | 7  |
| 7   | 3  | 2  | 6  | 2   | 0  |

$$TAT = CT - AT$$

$$WT = TAT - BT$$

Gantt chart :



$$L = 20 - a = 18$$

$$L = \max \text{ of } C_i - \min \text{ of } A_i$$

$$= 20 - a$$

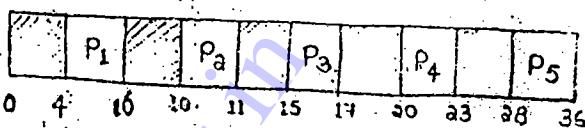
$$= 18$$

Since, there are no processes arriving at 'clock time = 0'. The process P<sub>4</sub> arrives at 'a' clk time. So, the time (0-a) is subtracted from the total completion time. And, since P<sub>4</sub> arrives earlier than that of process P<sub>1</sub>.

Process P<sub>4</sub> is considered first and then the other processes arrived within the particular Burst time.

Eg. :-

| PNO | AT | BT |
|-----|----|----|
| 1   | 4  | 2  |
| 2   | 10 | 1  |
| 3   | 15 | 2  |
| 4   | 20 | 3  |
| 5   | 28 | 8  |
| 16  |    |    |



$$WT(P) = 0$$

$$L = 36 - 4 = 32$$

Shortest Job First (SJF) :

- \* It is also called as shortest process next (SPN).

Criteria : Burst Time

Mode of working : Non-preemptive (default).

It also works under pre-emptive mode.

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

| P <sub>1</sub> | P <sub>4</sub> | P <sub>6</sub> | P <sub>2</sub> | P <sub>3</sub> | P <sub>5</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 4              | 5              | 6              | 8              | 11             |

Initially, Process P<sub>1</sub> is ready for execution, since it has arrival time = 0. So, it has been processed.

For processing 'P<sub>1</sub>', it requires 4 clock cycles (B.T=4), within

the time, processes P<sub>4</sub>, P<sub>6</sub>, P<sub>2</sub>, P<sub>3</sub> are also ready for execution.

Within those processes, select the process which have less

Q. 1

Q. 2

Q. 3

Q. 4

Q. 5

Q. 6

Q. 7

Q. 8

Q. 9

Q. 10

Q. 11

Q. 12

Q. 13

Q. 14

Q. 15

Q. 16

Q. 17

Q. 18

Q. 19

Q. 20

Q. 21

Q. 22

Q. 23

Q. 24

Q. 25

Q. 26

Q. 27

Q. 28

Q. 29

Q. 30

Q. 31

Q. 32

Q. 33

Q. 34

Q. 35

Q. 36

Q. 37

Q. 38

Q. 39

Q. 40

Q. 41

Q. 42

Q. 43

Q. 44

Q. 45

Q. 46

Q. 47

Q. 48

Q. 49

Q. 50

Q. 51

Q. 52

Q. 53

Q. 54

Q. 55

Q. 56

Q. 57

Q. 58

Q. 59

Q. 60

Q. 61

Q. 62

Q. 63

Q. 64

Q. 65

Q. 66

Q. 67

Q. 68

Q. 69

Q. 70

Q. 71

Q. 72

Q. 73

Q. 74

Q. 75

Q. 76

Q. 77

Q. 78

Q. 79

Q. 80

Q. 81

Q. 82

Q. 83

Q. 84

Q. 85

Q. 86

Q. 87

Q. 88

Q. 89

Q. 90

Q. 91

Q. 92

Q. 93

Q. 94

Q. 95

Q. 96

Q. 97

Q. 98

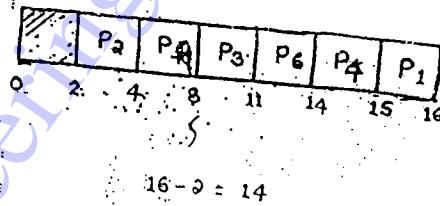
Q. 99

Q. 100

Burst time and process it. Here,  $P_4$  has a  $B.T = 1$ . So, it is selected.  
 Total clock cycles =  $4+1=5$ . At the time of 5<sup>th</sup> clock cycle, process  $P_6$  (new) is available on ready queue. Since, the  $B.T=1$  of process ' $P_6$ ', after completion of ' $P_4$ ', ' $P_6$ ' is to get executed, even though it added recently in queue.

Eg. :-

| PNO | AT | BT |
|-----|----|----|
| 1   | 5  | 1  |
| 2   | 2  | 2  |
| 3   | 3  | 3  |
| 4   | 2  | 4  |
| 5   | 4  | 1  |
| 6   | 3  | 3  |



$$16 - 2 = 14$$

(3). Shortest Remaining Time First (SRTF) :-

In pre-emptive mode, SJF is known as shortest Remaining Time first.

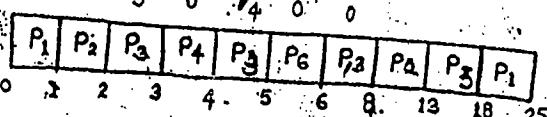
"Pre-emption of running process is based on the arrival of a new shorter process".

Criteria : Burst Time

mode : pre-emptive.

Eg. :-

| PNO | AT | BT |
|-----|----|----|
| 1   | 0  | 8  |
| 2   | 1  | 6  |
| 3   | 2  | 4  |
| 4   | 3  | 1  |
| 5   | 4  | 6  |
| 6   | 5  | 1  |



$$TAT(P_3) = 8 - 2 = 6$$

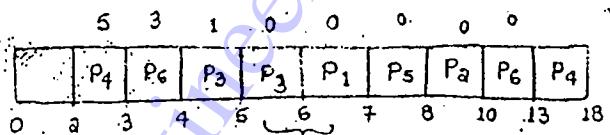
P<sub>2</sub>

Initially, process P<sub>1</sub> has arrived, and within 1 clock cycle time arrives when

P<sub>3</sub> & so on. So, only 1 clock cycle is considered for P<sub>1</sub> within B.T = 8, so the remaining B.T = 7 remains. In P<sub>2</sub>, B.T = 5, P<sub>3</sub>  $\Rightarrow$  B.T = 3, P<sub>4</sub>  $\Rightarrow$  B.T = 0.

At clock cycle = 4, process P<sub>5</sub> also arrives, but if we consider it, it has more Burst time than that of Process P<sub>3</sub>. So, we consider P<sub>3</sub> as it is SRTF. Next, within 5 clock cycles, P<sub>6</sub> also arrives, which have B.T = 1, so we consider P<sub>6</sub> after P<sub>3</sub>, and so on..... upto the end.

| PNO | AT | BT |
|-----|----|----|
| 1   | 5  | 1  |
| 2   | 7  | 3  |
| 3   | 4  | 2  |
| 4   | 2  | 6  |
| 5   | 6  | 1  |
| 6   | 3  | 4  |



P<sub>1</sub>, P<sub>3</sub> have equal B.T's  
but P<sub>3</sub> is considered

### Performance of shortest Job First (SJF):

Burst Time

#### Advantages:

- \* Enhances Throughput
- \* Average WT & TAT reduces.

#### Disadvantage:

- \* Starvation to longest job.
- \* It is non-implementable, because burst times of processes are not mentioned.

Non-preemptive SJF :-

| P.NO | AT | BT |
|------|----|----|
| 1    | 0  | 3  |
| 2    | 2  | 6  |
| 3    | 4  | 4  |
| 4    | 6  | 5  |
| 5    | 8  | 2  |

NP-SJF :

| P <sub>1</sub> | P <sub>2</sub> | P <sub>5</sub> | P <sub>3</sub> | P <sub>4</sub> |
|----------------|----------------|----------------|----------------|----------------|
| 0              | 3              | 9              | 11             | 15 20          |

HRRN :-

| P <sub>1</sub> | P <sub>2</sub> | P <sub>3</sub> | P <sub>5</sub> | P <sub>4</sub> |
|----------------|----------------|----------------|----------------|----------------|
| 0              | 3              | 9              | 13             | 15 20          |

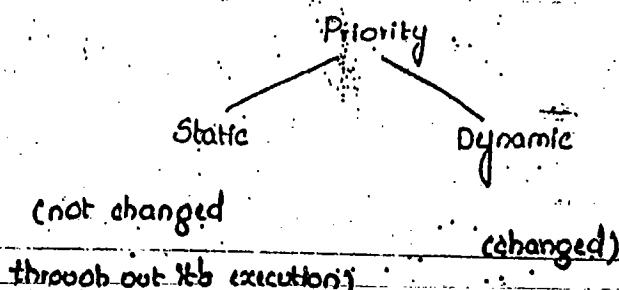
\* process having high R.R. it is first considered.

\* At clock cycle 9, Process P<sub>5</sub> is in Ready Queue, which have very little B.T. = 2. Process P<sub>3</sub> executed instead of P<sub>5</sub>, because of long time waiting. This scheduling algorithm gives importance to long waiting processes as well as B.T. processes.

#### (5) Priority based Scheduling :

Criteria : priority

Mode : non-preemptive



Increasing the priority level regularly at the runtime is called as  
"Aging Algorithm".

Drawback :

Starvation.

\* Higher Integer

Eg:-

| Priority | PNO | AT | BT |
|----------|-----|----|----|
| 4        | 1   | 0  | 4  |
| 5        | 2   | 1  | 5  |
| 6        | 3   | 2  | 8  |
| 8        | 4   | 3  | 6  |
| 2        | 5   | 4  | 3  |
| 10       | 6   | 5  | 5  |

NP - priority :-

| P <sub>1</sub> | P <sub>4</sub> | P <sub>6</sub> | P <sub>3</sub> | P <sub>2</sub> | P <sub>5</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|
| 0              | 4              | 10             | 15             | 23             | 28             |

high priority

preemptive priority :-

| CPU | P <sub>1</sub> | P <sub>3</sub> | P <sub>3</sub> | P <sub>4</sub> | P <sub>4</sub> | P <sub>6</sub> | P <sub>4</sub> | P <sub>3</sub> | P <sub>3</sub> | P <sub>3</sub> | P <sub>5</sub> |    |
|-----|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----|
|     | 0              | 1              | 2              | 3              | 4              | 5              | 10             | 14             | 21             | 25             | 28             | 31 |

a:-

| Prio. | PNO | AT | BT |
|-------|-----|----|----|
| 12    | 1   | 5  | 4  |
| 9     | 2   | 6  | 5  |
| 7     | 3   | 2  | 4  |
| 6     | 4   | 1  | 6  |
| 4     | 5   | 4  | 9  |
| 8     | 6   | 3  | 3  |

|  | P <sub>4</sub> | P <sub>3</sub> | P <sub>6</sub> | P <sub>6</sub> | P <sub>1</sub> | P <sub>3</sub> | P <sub>6</sub> | P <sub>3</sub> | P <sub>4</sub> | P <sub>5</sub> |    |    |
|--|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----|----|
|  | 0              | 1              | 2              | 3              | 4              | 5              | 9              | 14             | 15             | 18             | 23 | 25 |

### Round-Robin (m.p./Time sharing) :-

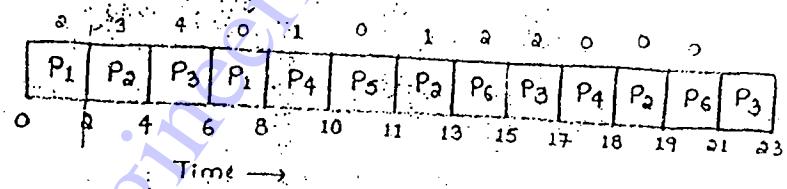
Criteria :  $A.T + \underbrace{\text{Time Quantum (TQ)}}_{\text{Slice}}$

mode : pre-emptive

$$TQ = a$$

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

$$R.Q : P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_3 P_4 P_2 P_6 P_3$$



- \* Initially,  $P_1$  arrives and executes till  $B.T = a$ . (Since Time slice =  $a$ ). Then, within a clock cycles, processes  $P_2, P_3$  also arrives, then considered their  $B.T$  at  $TQ = a$  and hence  $P_1$  requires another  $a$  clk cycles, it must again be maintained in Ready Queue.
- \* At time = 8, all the processes arrives, based on their  $B.T$ , ready Queue is maintained.
- \* If  $TQ$  is very small, efficiency ( $\mu$ ) = 0.
- \* If  $TQ$  is low  $\Rightarrow$  context switching occurs.
- \* If  $TQ$  is high  $\Rightarrow$  FCFS approach is followed } so  $TQ$  value must be chosen moderately.

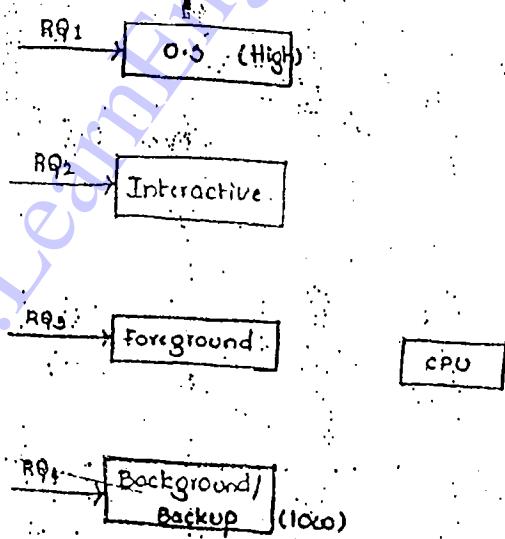
## Performance of Round Robin

| Time-Quantum<br>(TQ)             |                                   |                                   |                            |
|----------------------------------|-----------------------------------|-----------------------------------|----------------------------|
| Very Small.                      | Small                             | large                             | Very large                 |
| Efficiency ( $\mu$ ) $\approx 0$ | * more context switching overhead | * less context switching overhead | * works like FCFS approach |
|                                  | * Improves response time          | * less interactive response       | * Very poor response time  |

08/07/2010

Thursday

## Multi-level Queue Scheduling



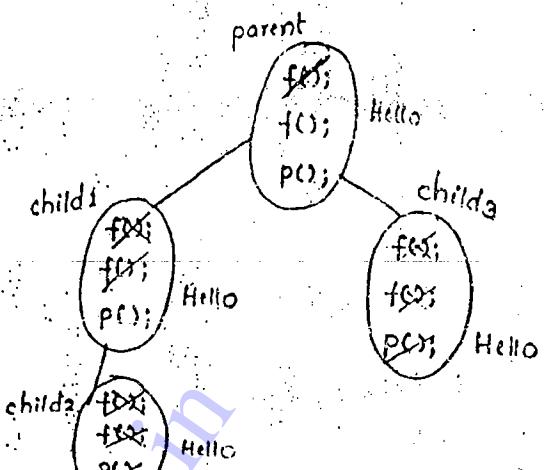
- \* In multi-level queue, the drawback is that, the second queue to serviced only when first queue is empty / complete.

```
(a) main()
{
 fork();
 fork();
 printf("Hello");
}
```

- \* If 1 fork  $\Rightarrow$  2 prints
- 2 forks  $\Rightarrow$  4 prints
- 3 forks  $\Rightarrow$  8 prints

$$n \text{ forks} \Rightarrow 2^n = \text{Total}$$

$$\text{new } n = 2^n - 1, \text{ parent } = 1 \\ (\text{child})$$



Output :

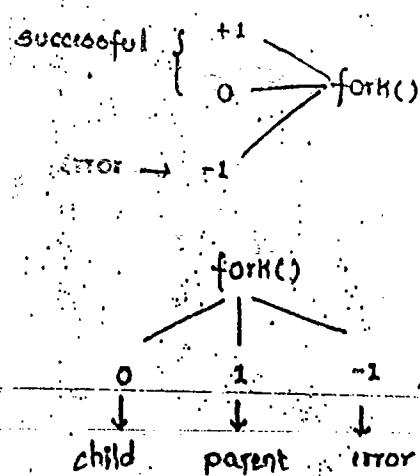
Hello  
Hello  
Hello  
Hello.

(b) main()

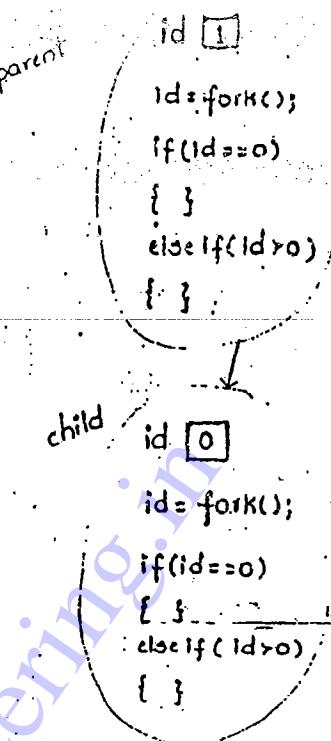
```
{
 int i, n;
 for(i=1; i<=n; ++i)
 fork();
}
```

- (a)  $n-1$       (b)  $n!-1$       (c)  $n^2-1$       (d)  $2^n-1$  (new forks)

- \* Any system call may return either +1, 0, -1 values.



```
main()
{
 int id;
 id = fork();
 if(id == 0) // child
 {
 // child //
 }
 else if(id > 0)
 {
 // parent //
 }
 else printf("Error in creation");
}
```



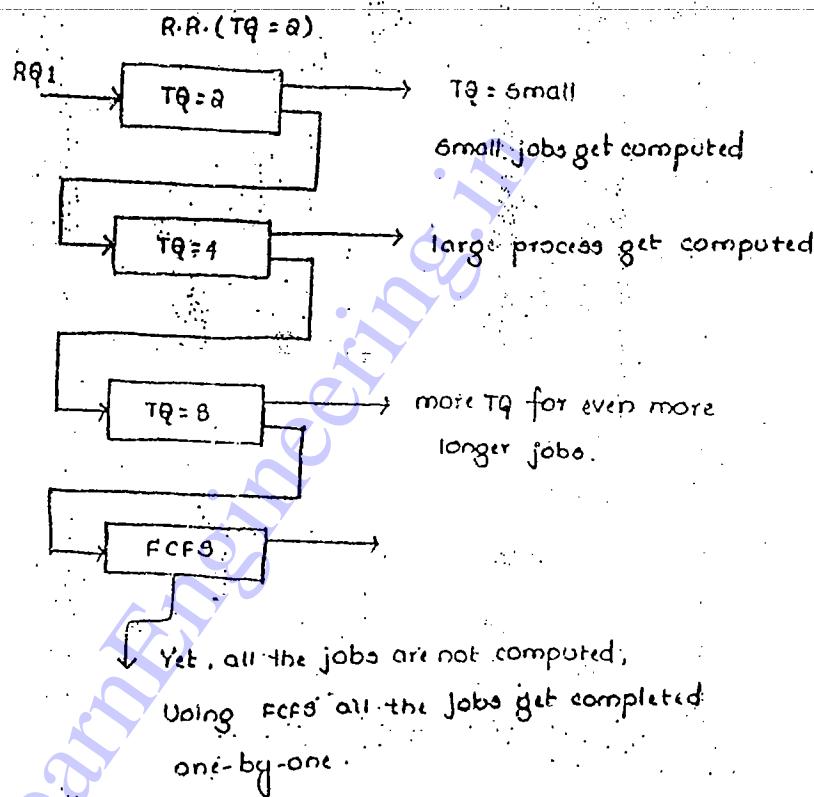
### Difference between Kernel mode & user mode:

Compilers, editors and similar application-independent programs are not part of the O.S., even though they are typically supplied by computer architectures. This is crucial, but subtle point: The O.S. system is that portion of the software that runs in Kernel mode: It is protected from user tampering by hardware.

Compilers & editors runs in user mode. If a user doesn't like a particular compiler, he/she is free to write their own clock interrupt handler, which is a part of O.S.

\* Here, the feedback is pre-empted to the same Queue.

\* So, we have multi-level feedback Queue to overcome this drawback.

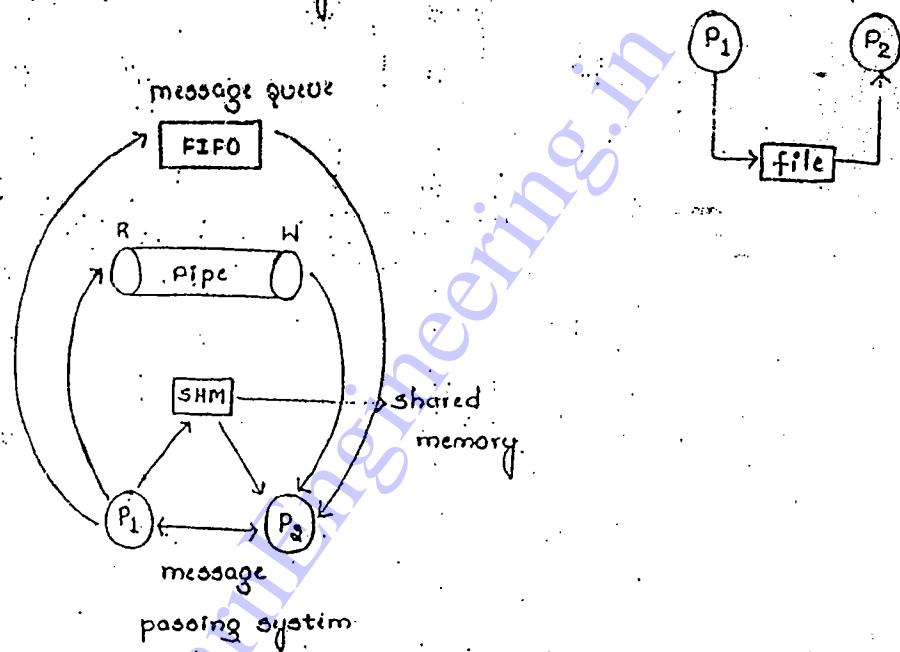


## Inter Process communication & Synchronization:

### Inter process communication (I.P.C) :-

\* For the communication among any two processes, there must be a media to communicate.

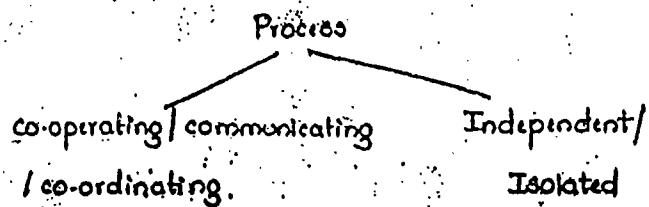
Eg. :- file (It is not always used)



Problems due to lack of synchronization :-

- \* Loss of data
- \* Inconsistency (wrong results)
- \* Dead locks.

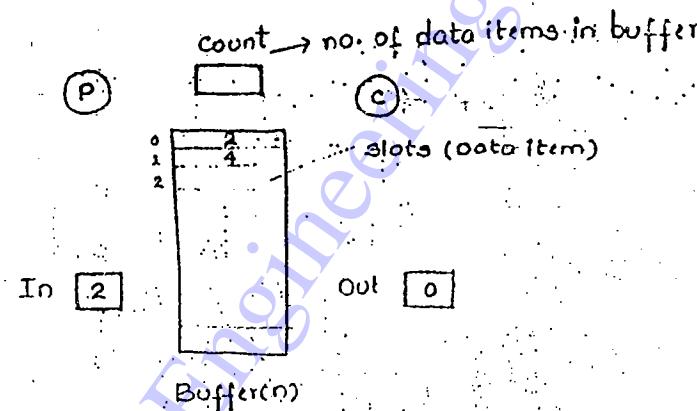
How to achieve synchronization?



### Co-operating process :-

- \* Two (or) more processes are said to be co-operative, if they get affected (or) affects the execution of other process.
- \* Otherwise, they are said to be Independent.

### Producer-Consumer problem :-



- \* If Buffer is full, producer get affected.
- \* If Buffer is empty, consumer get affected.

```
#define n 100

int Buffer[n];

void producer(void)
{
 int itemp, in=0;
 while(1)
 {
 ProduceItem(itemp);
 while(count==n); // Busy wait
 Buffer[in]=itemp;
 in++;
 }
}
```

```
 in = (in+1) mod n;
 count = count + 1;

void consumer(void)
```

```
{
 int itemc, out = 0;
 while(1)
 {
 while(count == 0);
 itemc = Buffer[out];
 out = (out+1) mod n;
 count = count - 1;
 processitem(itemc);
 }
}
```

Lack of Synchronization in Producers consumer problem :-

Inconsistency :-

In producer,

- count = count + 1; // This step needs to fetch  
(1) load Rp, m[count]  
(2) Inc Rp  
(3) Store m[count], Rp

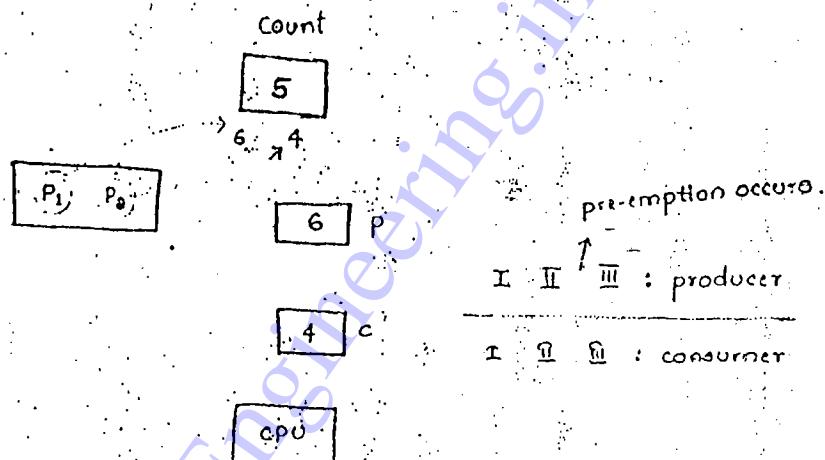
In consumer,

$\text{count} = \text{count} - 1;$

(1) Load R<sub>c</sub>, m[count]

(2) Dec R<sub>c</sub>

(3) Store m[count], R<sub>c</sub>



\* In producer,

while first two steps get executed, pre-emption occurs. i.e., consider the count = 5 (initially). In step1, the value is loaded into register 'R<sub>p</sub>', and then in step2, the value in the register get incremented. so, the value of count = 6.

\* Before storing the "count" value from the register, it get pre-empted. so, the producer have the count value = 6.

\* In consumer,

count value = 5 is stored in a register 'R<sub>c</sub>', and then decremented. Before transferring the value, from 'R<sub>c</sub>' it get pre-empted. so it have count = 4. so, there is inconsistency, because count value is never equal to 5.

13/07/2010

Tuesday

## Synchronization Mechanisms.

### Producer- Consumer :

### Lack of synchronization :

- \* Inconsistency
- \* Loss of data
- \* Deadlocks

### Terminology :

#### 1) Critical Section(s) & non-critical section(s) :-

- Critical section is that part of a program where shared resources are accessed.
- \* Non-critical section is that part of a program where no shared resources are accessed.  
 $CS \Rightarrow$  shared resource (any variable/device)

#### (a) Race conditions (concurrency) :

processes racing to get into critical section.

#### (b) pre-emption :

A process gets executed in the middle of other process.

$count = count + 1$

(1) load

(2) Inc

(3) store

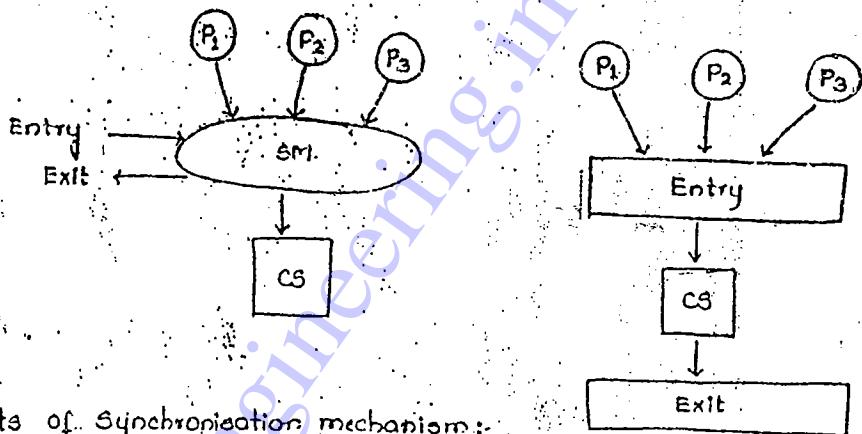
#### (4) mutual exclusion :

No two processes may be present in the critical section at same time.

### Architecture / model of synchronisation mechanism :

#### Synchronisation mechanism :

- \* To ensure mutual exclusion so that there is no problem of race condition among processes.

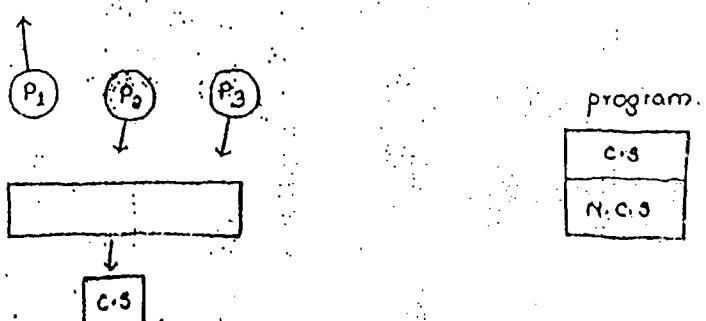


#### Requirements of Synchronization mechanism :

(1) mutual exclusion must always be guaranteed.

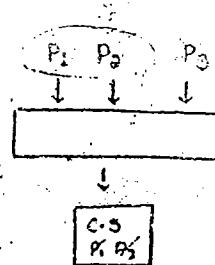
(2) progress :

process running outside critical section should not block (the process or the other interested process from accessing / entering the critical section).



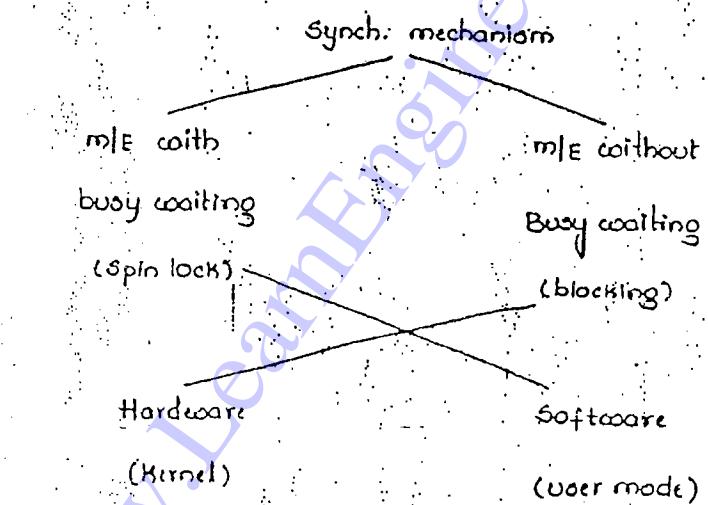
(3) Bounded waiting :

\* No process has to wait forever to access its critical section.



If  $P_1$  enters C.S and after leaving, if  $P_2$  wants to enter C.S, then it can enter and after performing operations, it leaves. Then, if again  $P_1$  wants to enter along with  $P_3$ , where  $P_1$  has high priority than  $P_3$ , then  $P_1$  enters & then  $P_3$  and so on, where there is no chance for  $P_3$  to enter C.S. (i.e.,  $P_3$  remains idle forever).

(4) No Assumptions about the speed (or) no. of CPU's may be assumed.



### Disabling Interrupts

- \* m/s without busy waiting.
- \* multi-process

\* Kernel mode.

\* The main cause of pre-emption is "Interrupt".

\* Disable the source of pre-emption in the critical section at entry level and enable the interrupts at exit level.

\* critical section

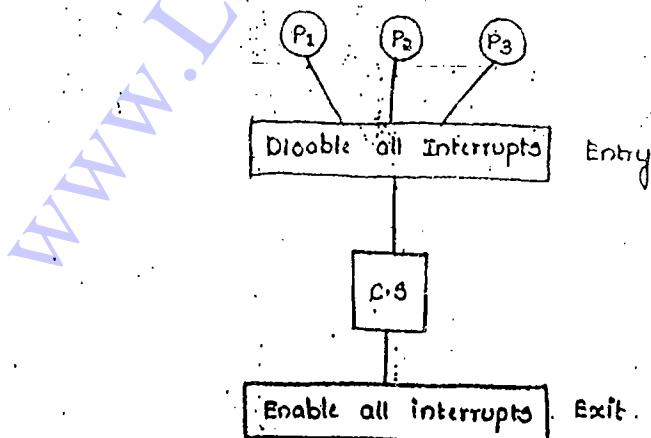
\* It guarantees m/s because if  $P_1$  running in c.s., other interrupts are disabled.

\* Race condition

\* Enabling / Disabling of Interrupts done only in kernel mode.

\* pre-emption.

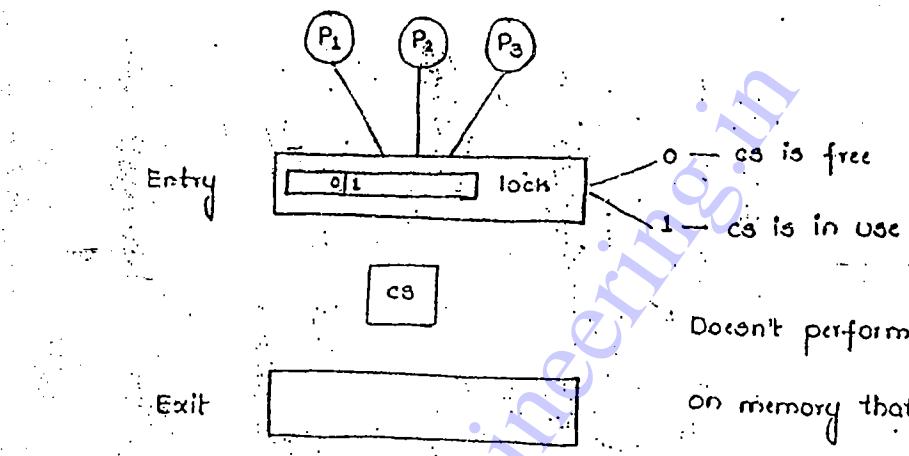
So, o.s process can only enable / disable the interrupts and user process cannot enable / disable  $\Rightarrow$  Restriction to o.s process.



\* Sloc mechanism implemented in user mode.

via busy waiting.

& multiprocs.



Doesn't perform direct manipulation

on memory that load into ALU

reg.

```
void Entry_section(int process)
{
 while(lock!=0); // Busy waiting
 lock=1;
}
```

cs

```
void Exit_section(int process)
```

```
{
 lock=0;
}
```

Here, initially the lock value is '0' ( $lock=0$ )

when  $P_1$  needs to enter c.s, it follows

all the steps.

Step 1: The lock value is stored in the register

R1

Step 2: It is compared whether lock is '0' or not

Step 3: If lock value is not zero (i.e.,  $lock=1$ ), then again process same steps (1&2)

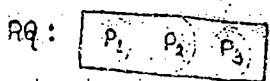
Step 4: If  $lock=0$ , then the process enters c.s and while entered then it changes

$lock=1$ .

### Assembly language :-

#### Entry Section :

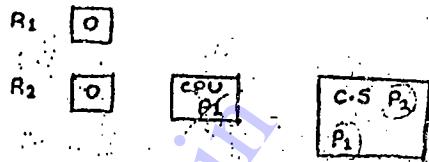
1. Load R<sub>i</sub>, m(lock)
2. cmp R<sub>i</sub>, #0 ;
3. JNE steps ;
4. Store m(lock), #1 ;



lock



Entry Section



c.s

P<sub>1</sub> : 1 2 3 preempt

P<sub>2</sub> : 1 2 3 4

5

P<sub>1</sub> : 4 5

#### Exit Section :

Store m(lock), #0 ;

- \* It fails to guarantee mutual exclusion.
- \* Busy waiting results in wastage of CPU cycles / CPU time.

Hint for testing either supports to guarantee m/E :-

- (1) Identify the shared variable (lock)
- (2) Read the value of shared variable (load)
- (3) compare (optional)
- (4) Update / action (based on result of comparison)

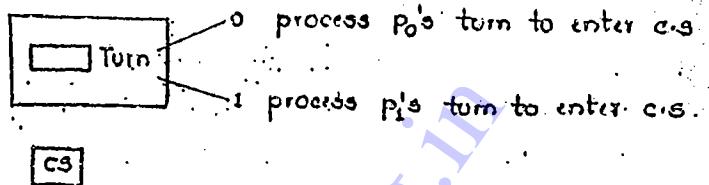
⇒ If process 'P<sub>1</sub>' entered c.s, and before changing lock=1, it gets pre-empted.

Since the lock=0, another process 'P<sub>2</sub>' wants to enter c.s and gets resides, in CPU, and gets into c.s.

\* Since P<sub>1</sub> and P<sub>2</sub> gets into c.s simultaneously, it fails to guarantee mutual exclusion.

### (3) Strict Alternation :

- \* Slocat user mode:  $(P_0, P_1)$  or  $(P_i, P_j)$
- \* a-process
- \* mle with busy waiting



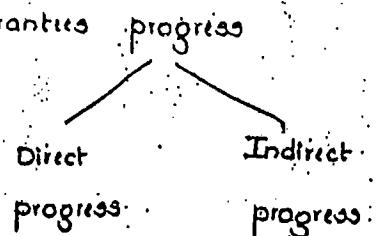
- \* Strictly on alternate basis, processes ( $P_0$  &  $P_1$ ) takes turn to enter c.s.

```
void P0(void)
{
 while(1)
 {
 Non-cs()
 Entry: cobile(turn!=0); // busy waiting
 cs
 turn=1;
 Exit: turn=0;
 Because "turn" value is
 not immediately updated,
 but updated at exit section.
 so, it guarantees mle.
 }
}
```

```
Void P1(void)
{
 while(1)
 {
 Non-cs()
 cobile(turn!=1);
 cs
 turn=0;
```

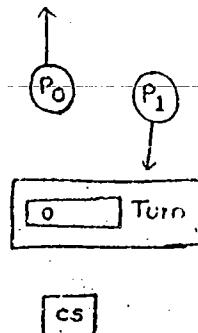
- \* Guarantees mle.

- \* Not Guarantees progress



### Direct progress :-

Initially, the turn=0, and if  $P_0$  is not interested in CS, and  $P_1$  is interested, then the turn must be equal to '1' to enter into CS. But, unless ' $P_0$ ' enters, "turn" can't be changed, and ' $P_0$ ' not interested to do so. Hence, progress is not guaranteed.

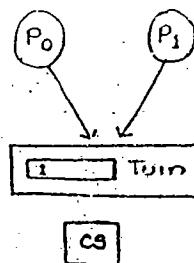


' $P_0$ ', which is outside block

### Indirect progress :-

If ' $P_0$ ' enters CS and while exiting turn changed to '1', so that ' $P_1$ ' need to enter. If again ' $P_0$ ' wants to enter (i.e., interested) and  $P_1$  is not interested.

Then,  $P_0$  is discarded to enter CS because turn=1 and it needs  $P_1$  to change turn=0. Hence, progress is not guaranteed.



$P_1$  is blocking  $P_0$

indirectly.

### (4) Peterson's Solution (Dekker's Algorithm):-

- \* no busy waiting
  - \* slow at user mode.
  - \* a-process solutions ( $P_0$  &  $P_1$ )
- ↓  
disadvantage

```

#define N 2
#define TRUE 1
#define FALSE 0

int interested[N] = {false};

int turn;

void entry_section(int process)
{
 (1) int others;
 (2) others = 1 - process;
 (3) interested[process] = TRUE;
 (4) turn = process;
 (5) while(interested[others] == TRUE && turn == process); // busy waiting
}

void exit_section(int process)
{
 interested[process] = false;
}

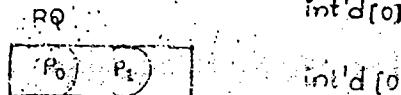
```

\* progress is guaranteed, m/E guaranteed.

\* Bounded waiting is guaranteed (Since, it is not guaranteed for more than 2 processes).

#### Drawbacks

- \* wastage of CPU time.



int'd[0] = ✓ T

int'd[0] = F T

turn

1 0

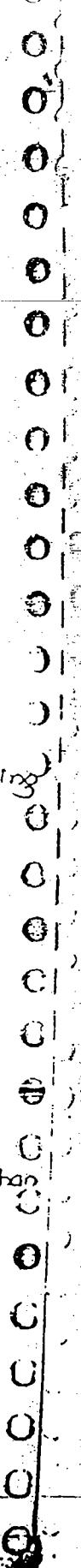
P<sub>0</sub>: 1, 2, 3

P<sub>1</sub>: 1, 2, 3, 4, 5

P<sub>0</sub>: 4, 5

Prt: [CS]

CS



### 5) Test & Set lock instruction (TSL) :-

(privileged instruction  $\Rightarrow$  atomically)

- \* slow at user mode
- \* no I/O with busy waiting
- \* multi-process (advantage over p Peterson)

Eg : TSL register, flag;

0 - cs is free

1 - cs is busy

\*\* "copies the value of flag into register and sets the value of flag to one (always)".

#### Entry-Section :

- (1) TSL regi, m[flag];
- (2) cmp regi, #0;
- (3) JNZ step1;
- (4) [cs]
- (5) store m[flag], #0;

#### Priority Inversion (P/TSL) :

RQ:

\* Preemptive priority based CPU scheduling is used.

[PHPL]

If  $H > L \Rightarrow$  deadlock

when a high priority process enters the CPU. And  $P_L$  requires CPU to perform its instructions, in order  $P_H$  to enter into C.S. so,  $P_H$  enters in CPU and needs to enter C.S &  $P_L$  to

in C.S and needs CPU to exit; hence, Deadlock occurs.

CPU  
PHPL

P/TSL

Entry  
Section

CS  
PL

### Producer & Consumer :

\* MLE without Busy waiting (Blocking)

(i) sleep() and wakeup() :-

```
void producer(void)
```

```
{ int itemp, in = 0;
```

```
while(1)
```

```
{ produce(itemp);
```

```
if(count == N) sleep(); \Rightarrow If Buffer = full
```

```
Buffer[in] = itemp; producer \Rightarrow sleep()
```

```
in = (in + 1) mod N;
```

```
count = count + 1;
```

```
if(count == 1) wakeup(consumer); \Rightarrow when item is placed in
```

```
Buffer consumer \Rightarrow wakeup.
```

```
void consumer(void)
```

```
{ int itemc, out = 0;
```

```
while(1)
```

```
{ if(count == 0) sleep();
```

```
itemc = Buffer[out];
```

```
out = (out + 1) mod N;
```

```
count = count - 1;
```

```
} If (count = N-1) wakeup(producer);
```

RQ: (P) (C)

N=3

~~∅ Y X 3~~

CPU

|   |   |
|---|---|
| 0 | X |
| 1 | Y |
| 2 | Z |

Buffer(0-a)

(C)

Preemption  
sleep()

(P): ...x

Inconsistency: Y

z

sleep(); } Deadlock

(C): sleep();

- \* while consumer executing instructions, before "sleep()" gets executed, pre-emption takes place and producer executes instructions and after buffer is full, producer goes to sleep() state assuming that consumer could wake-up, when consumer starts executing, sleep() instruction gets executed so, consumer expects that producer could be wake up. Hence, it represent the situation of "Deadlock."

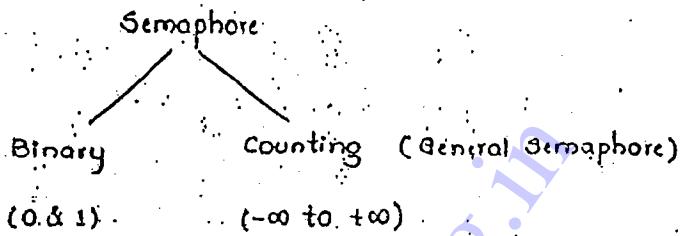
17/07/2010

Saturday

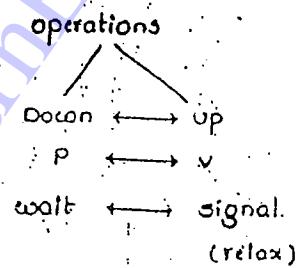
## Semaphores

- \* A variable (Semaphore) that takes on integrated values.

↓  
(Integers)



- \* It is an operating system resource.
- \* Semaphore operations executed in kernel mode (automatically)
- \* Implementation by Dijkstra (variable)



Struct SEMAPHORE

```

{
 int value;
 QueueType L;
};

// List of PCB's of those processes that get blocked while
// performing "down" operation unsuccessfully;

```

```

 SEMAPHORE s;
 s.value = 4;
 DOWN(s);
 <c,s>

```

"Semaphore" gets executed in kernel mode.

\* If the value after decrementing is negative, then it is unsuccessful.

\* If value is zero (or) positive then it is successful.

(+) value  $\rightarrow$  Successful Doon processes.

(-) value  $\rightarrow$  Blocked processes.

Eg :- s.value = 4

Doon.

$$4 - 1 = 3$$

$$3 - 1 = 2$$

$$2 - 1 = 1$$

$$1 - 1 = 0$$

$$0 - 1 = -1$$

$$-1 - 1 = -2$$

4 successful processes [ (+)ve. value of counting available in stack ] Semaphore always indicate no. of doon operations successful ].

-2 { 2 blocked processes (unsuccessful processes)  $\Rightarrow$  waiting for semaphore.

DOWN(SEMAPHORE s)

$$s.value = s.value - 1;$$

if (s.value < 0) //unsuccessful.

put this process (PCB) in

SLC & Block it (Sleep(s));

}

After Blocking the process,

just it moves off (i.e., no

doon operations are performed)

UP(SEMAPHORE s)

$$s.value = s.value + 1;$$

if (s.value < 0)

{ Select a process from S.L()

and wakeup();

wakeup process gets

into c.s but not

If  $s = -a$  perform "down" operation

After  $p_1 = -2 + 1 = -1$

$$p_1 = -1 + 1 = 0$$

$$p_2 = 0$$

$$= 0 + 1$$

\* When we have 3 processes, semaphore value

starts with 1  $\Rightarrow$  performing down operation

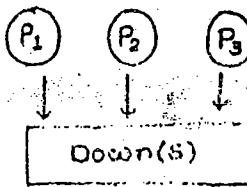
on it, the value becomes '0'. (i.e., the down

operation is successful, & it enters into critical

section). After  $P_1$  enters into CS, all the other

processes perform docon operation

$$(P_1 = 0, P_2 = -1, P_3 = -2)$$

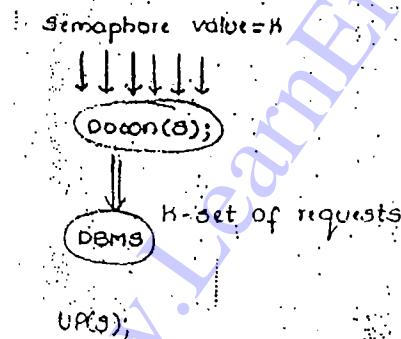


CS

$P_1$  -1  
UP

\* Process never get blocked while performing "UP" operation and it gets blocked while performing "Docon" operation.

Allowing K processes into CS



After process wakes up, don't perform any docon operations, simply execute CS the critical section.

$$\text{Eg: } S = 12$$

16 p, 2 v, 8 p, 1 v, 4 p, 3 v, 8 p, 2 v

S.L()  $\rightarrow$  Semaphore Queue consists  
list of processes;

$$S = 12 \text{ (16 down)}$$

p  $\rightarrow$  docon

$$= -4 \text{ (2 up)}$$

v  $\rightarrow$  up

$$= -9 \text{ (8 docon)}$$

$$= -10$$

$$= -9$$

$$= -13$$

$$= -10$$

$$= -16$$

$$\Rightarrow -14$$

## Binary (mutex) Semaphore (0/1):

Struct BSEMAPHORE

```
{
 enum value {0,1};
 Queue Type L;
}
```

BSEMAPHORE S;

S.value = 1;

DOWN(S);

<or>

DOWN(BSEMAPHORE S)

if (S.value == 1) // successful.

S.value = 0;

else // unsuccessful.

{ put this process (PCB)

in BLC & Block it

sleep();

condition-I : If there are Blocked processes.

(or)

condition-II : If there is a process in critical section

Then, value of Binary semaphore must be zero.

After performing down(s) operation  
on Bsemaphore,

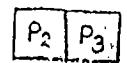
if value = 0  $\Rightarrow$  unsuccessful

value = 1  $\Rightarrow$  successful.

(1-1)

If  $S=1 \Rightarrow P_1 = 0$  (After down(s))

operation, semaphore  
must be 1).



$P_2$  checks semaphore value. Since, the value = 0, no down(s) operation is performed. Simply it moves to Block() state. Similarly, by having semaphore value =  $P_3$  also gets blocked.

After completion of all processes, semaphore value gets incremented.

(Because chance given to newly entered processes).

UP(BSEMOPHORE S)

{

If (Q.L() is not empty)

{

Select a process from  
S.L() and wake up()

}

else

{

S.value = 1;

}

}

Semaphore value = 0.  $\Rightarrow$  It represents that there is no compilation to present blocked processes.

After completion of processing all the jobs in blocked Queue, increment semaphore value. It represents that don't release semaphore until all the jobs are completed.

Initial semaphore = 0, no blocked processes

S = 1

so 10 p, 3V, 15p, 7V, 8p, 2V, 3p, 1V S=0  
0 1 0 1 0 1 0 1  
9 9 6 21 14 22 10 23 22

p = down

V = up

| process | Blocked | 9 | 6 | 21 | 14 | 22 | 20 | 23 | 22 |
|---------|---------|---|---|----|----|----|----|----|----|
|         |         |   |   |    |    |    |    |    |    |

### Four Cases of Binary Semaphores:

Case (i):

$$S = 1$$

P(s);

s value becomes = 0

Status = successfully.

case (ii):

$$S = 1$$

V(s);

$$S = 1$$

Status = successful.

case (iii):

$$S = 0$$

P(s);

$$S = 0$$

Status = unsuccessful.

case (iv):

$$S = 0$$

V(s);

first value (initial value)

$$S = 1$$

(Up operation) Status = successful.

### Classical Interprocess Communication problems

(1) producer-consumer :-

```
#define N 100
```

```
int Buffer[N];
```

```
semaphore empty = N; // no. of empty slots in Buffer
```

```
semaphore full = 0; // no. of full slots
```

```
semaphore mutex = 1; // used to ensure m/e between P & C on buffer
```

```
void producer(void)
```

```
{ int itemp, in = 0;
```

```
while(1)
```

```
{
```

```
producer_item(itemp);
```

full = 0  $\Rightarrow$  Buffer is empty

no item available

```
down(empty); // check for Buffer is full or not
```

```
down(mutex)
```

(Blocking producer)

If Empty = 0, down(empty) = 0-1  
 $\Downarrow$  = 0

```
Buffer[in] = itemp;
```

Buffer = full, then block producer.

```
in = (in+1) mod n;
```

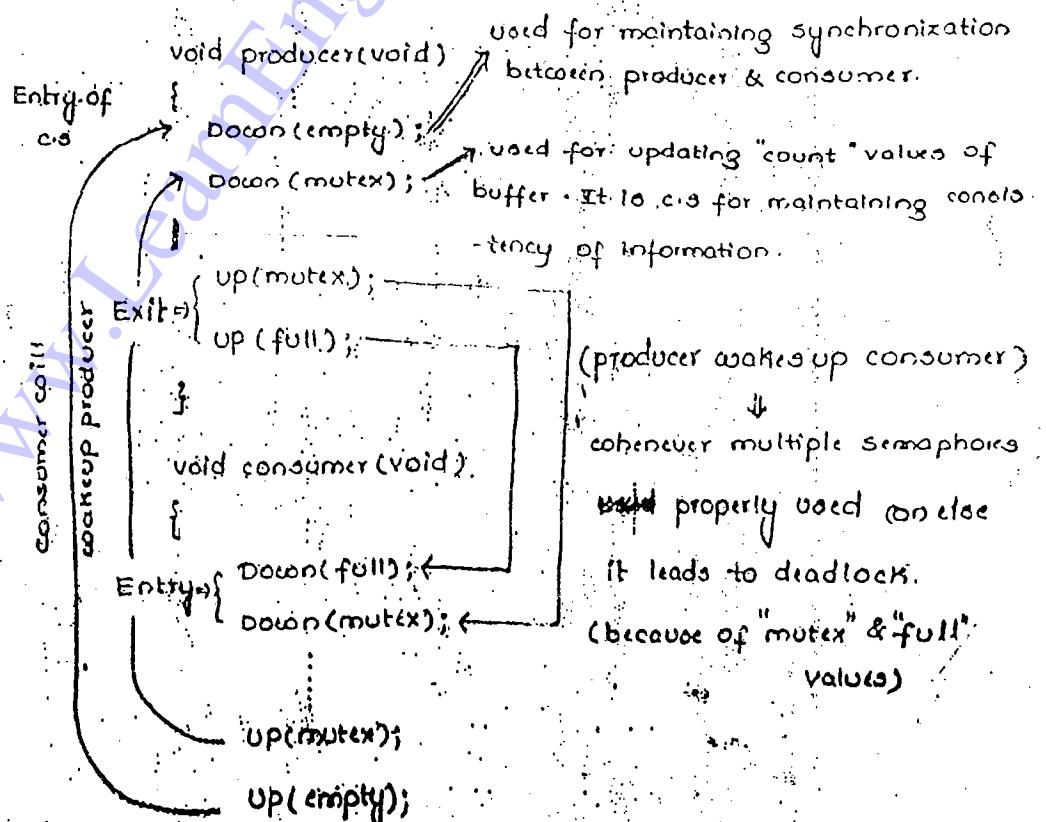
```
Exit \leftarrow { up(mutex); // release the critical section.
 up(full);
```

```
} consumer-item(itemp);
```

```

void consumer(void)
{
 int itemc, out=0;
 cobjfe(1);
 {
 consumer-item(itemc);
 down(full); // Interchange If
 down(mutex); deadlock occurs to
 itemc = Buffer[out]; up(full)
 out = (out+1) mod N; up(mutex).
 up(mutex);
 up(empty);
 producer-item(itemc);
 }
}

```



(a) Reader-writer problem :-

(1) 1 Reader - 1 writer

(2) Multiple Readers - 1 writer

(3) Multiple Readers - Multiple writers

\* we can perform multiple "Read" operations at a time.

Database enters into critical section, when there is a writer. So, at a time, only one write operation can be performed.

```
int RC=0; // Readers count
Semaphore mutex = 1; // Used for mutual exclusion b/w readers
 // RC must be in c.s.
Semaphore db = 1; // Used to ensure m/e between readers & writers
 // on DBMS
void Reader(void)
{
 critical(C1)
 {
 Down(mutex); // To increment readers count, check "mutex" value
 RC = RC+1;
 if(RC==1) // If first reader enters into DBMS, it's responsibility
 // is to check whether any writer present in c.s
 // (or not), using (RC==1) condition.
 DOWN(db);
 UP(mutex);
 CS <READ-DBMS>
 DOWN(mutex);
 RC = RC-1;
}
```

```
if(RC==0)
```

UP(db);  $\Rightarrow$  release db

UP(mutex);  $\Rightarrow$  release mutex

Suppose db have writer, it's val becomes = 0.. so, no reader enter in the first stmt.

```
void writer(void)
```

```
{
```

```
while(1)
```

```
{
```

```
down(db);
```

```
< update drama >
```

```
up(db);
```

```
}
```

(3) Dining philosopher's problem:-

Normal case :-

```
void philosopher(int i)
```

```
{ while(1)
```

```
think();
```

```
take-fork(i);
```

```
Take-fork((i+1)%N);
```

```
eat();
```

```
put-fork(i);
```

```
put-fork((i+1)%N);
```

P0 - f0

P1 - f1

P2 - f2

P3 - f3

P4 - f4

deadlock

For More Visit : [www.LearnEngineering.in](http://www.LearnEngineering.in)

• To avoid the problem of deadlock, in this, philosopher breaks the rule.

i.e., Even positions members follow one approach & odd position members follow another approach.

### Actual problem :-

- Philosophers take the position on a table, they first try to take left hand side fork, if they succeed, then they take right fork. If both the forks are available, then they start eating & then put off the forks.
  - But, at a time, all philosophers can't have both forks, where dead lock situation occurs.

Eg:- If  $P_0$  first takes  $f_0$ ,  $P_1$  takes  $f_1$ ,  $P_2$  takes  $f_2$ ,  $P_3$  takes  $f_3$ ,  $P_4$  can't have its left fork because  $P_0$ 's right is  $P_4$ 's left. Hence, deadlock occurs. So, no one wants to release their left forks.

State [N] stores state of philosopher either thinking  
(0) eating

Semaphore mutex = // used for releasing & taking the fork  $\Rightarrow$  it is c.s.

Semaphore  $S[N] = \{0\}$ ;

```

#define N 5

#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (i+N-1) % N
#define RIGHT (i+1) % N

int state[N] = {0}; // Either thinking or hungry or eating.

Semaphore mutex = 1; // To handle forks either by taking (or) releasing

void philosopher(int i)
{
 critical(i)
 {
 Think(i); // initial state
 take_forks(i);
 eat();
 put_forks(i);
 }
}

void take_forks(int i)
{
 Docon(mutex); // taking forks.
 state[i] = HUNGRY;
 test[i];
 UPC(mutex);

 DOWN(folk); // fails to take forks so blocked.
}

```

```
 SEMAPHORE;
```

```
 state[i] = THINKING;
```

```
 test(LEFT); } state again changed.
```

```
 test(RIGHT); } to THINKING.
```

```
 UP(mutex); }
```

```
void test(int i) // testing forks are available or not
```

```
{
```

```
 if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
```

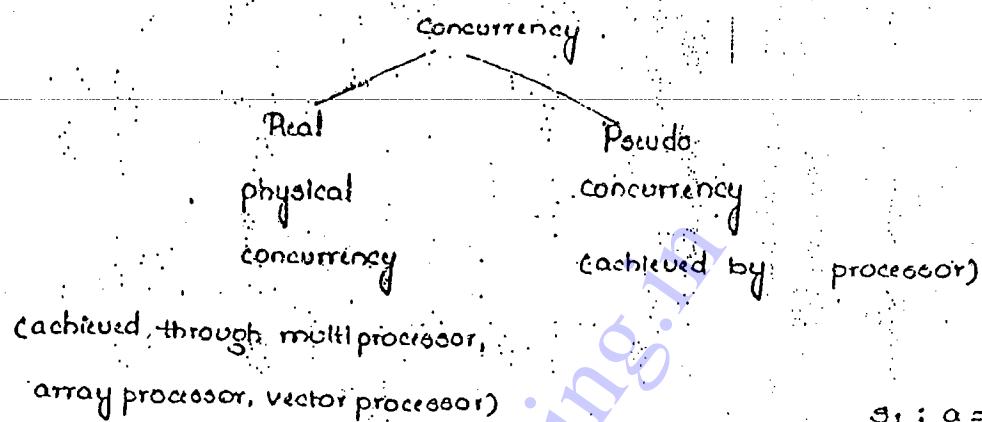
```
 {
```

```
 state[i] = EATING
```

```
 UP(s[i]); // Eating is completed & set s[i]=1
```

```
}
```

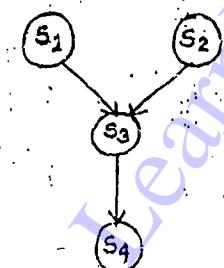
## Concurrency & Concurrent programming



Precedence Graph indicates

- \* It indicates which statements executes concurrently.

$$\begin{aligned}
 S_1 &: a = b + c; \\
 S_2 &: d = e + f; \\
 S_3 &: k = a + d; \\
 S_4 &: l = k * m;
 \end{aligned}$$



There is no precedence.  
Any statement can be executed first.

Concurrency conditions :-

$$S_1 : a = b + c;$$

$$S_2 : d = e + f;$$

$$S_3 : k = a + d;$$

$$S_4 : l = k * m;$$

Read set  $\rightarrow$  The values read into equation

$$R(S_1) = \{b, c\}$$

Write set  $\rightarrow$  update value in equation

$$W(S_1) \in \{a\}$$

$a + z = \dots, b + c, \dots$  all  $a, b, c$  updated. So,

$$R(S) = W(S) = \{a, b, c\}$$

- (1)  $R(s_i) \cap W(s_j) = \emptyset$  // Reading & writing set of two different processes equal, then we can't perform concurrent execution at a time.
- (2)  $W(s_i) \cap R(s_j) = \emptyset$
- (3)  $W(s_i) \cap W(s_j) = \emptyset$

$$a = c * d$$

$$a = l * m \text{ (not supported)}$$

If (count ≠ 0) → all processes are not completed.

fork & Join :

fork t;

$s_j$ ; // after completion of  $s_j$  go to  $s_i$ .

go to  $t_1$

$t_1 : s_k$ ; // it executes immediate instructions  $t_1$

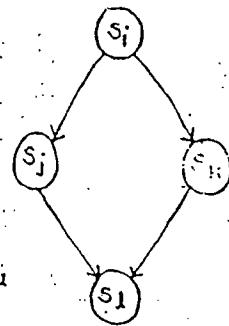
$t_1 : \text{Join count}$ ; // in

$s_1$

Join (int count)

count = count - 1;

if (count ≠ 0) // all the processes are not completed. So, there is no execution of  $s_1$ .  
exit;



Count = 3

s<sub>1</sub>;

fork L<sub>1</sub> (s<sub>1</sub> followed by fork)

s<sub>2</sub>;

s<sub>4</sub>;

fork L<sub>2</sub>

s<sub>5</sub>;

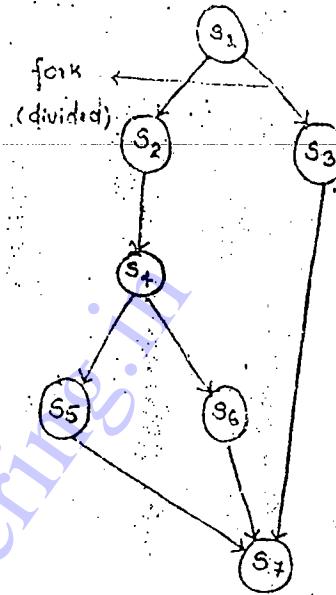
L<sub>1</sub>: s<sub>3</sub>

go to sL<sub>3</sub>

L<sub>2</sub>: s<sub>6</sub>

L<sub>2</sub>: Join count

s<sub>7</sub>;



Count = 3

s<sub>1</sub>;

fork L<sub>1</sub>

s<sub>2</sub>;

go to L<sub>1</sub>;

go to L<sub>2</sub>;

let fork L<sub>2</sub>

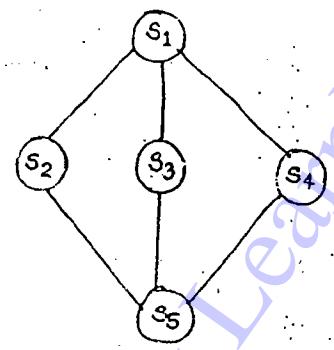
s<sub>3</sub>;

go to L<sub>2</sub>;

- L<sub>2</sub>: s<sub>4</sub>;

go to L<sub>3</sub>;

- L<sub>3</sub>: Join count (check for  
s<sub>5</sub>; all processes  
either complete  
(or) not).



s<sub>2</sub>, s<sub>3</sub>, s<sub>4</sub> are independent processes so, they can be accessed in anyway. But,

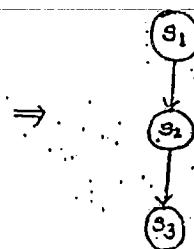
s<sub>5</sub> is purely dependent on s<sub>2</sub>, s<sub>3</sub> and s<sub>4</sub>.

Eg:- If s<sub>2</sub>, s<sub>3</sub> accessed & completed, and s<sub>4</sub> needed to be accessed now,  
even though there is s<sub>5</sub> to be accessed, we won't access. After the  
completion of all the processes at the upper level (s<sub>2</sub>, s<sub>3</sub>, s<sub>4</sub>), then only  
the last level process (s<sub>5</sub>) must be computed. for such checking, "Join".

condition is used which counts the completed functions that are accessed, u. for completion of  $s_5$ .

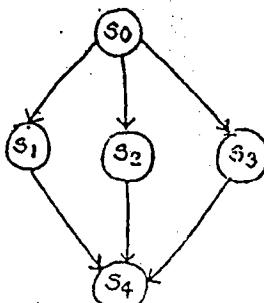
Parbegin - Parenrd & co-begin - Cend :-

$\text{begin}\{$   
 $s_1;$   
 $s_2;$   
 $s_3;$   
 $\text{end} \}$



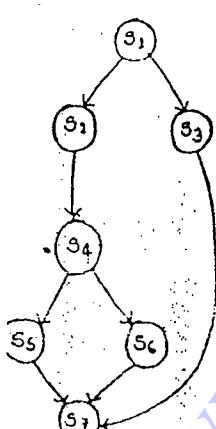
$s_0;$   
parbegin $\{$   
 $s_1;$   
 $s_2;$   
 $s_3;$   
parenrd  
 $s_4;$

They support parallelism

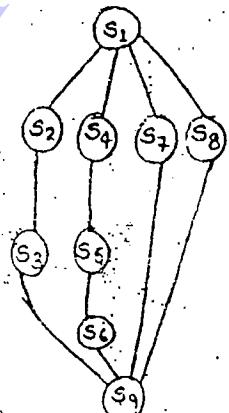


Begin-end  $\Rightarrow$  represents sequential actions

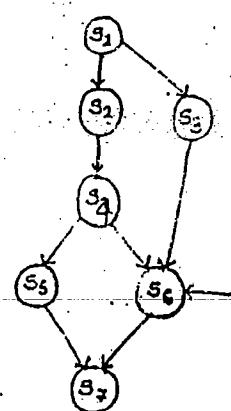
Parbegin - Parenrd  $\Rightarrow$  represents parallel actions.



$s_1;$   
Parbegin  
begin  
 $s_2;$   
 $s_4;$   
Parbegin  
 $s_5;$   
 $s_6;$   
Parenrd  
 $s_7;$   
Parenrd  
 $s_8;$



$s_1;$   
Parbegin  
begin  
 $s_2;$   
 $s_3;$   
end  
begin  
 $s_4;$   
 $s_5;$   
 $s_6;$   
end  
 $s_7;$   
 $s_8;$   
Parenrd  
 $s_9;$



For this graph, we cannot write a code because for execution of 'S6', we need S4 and S3. With the help of code, we cannot impose such condition. We can overcome this problem by using "Semaphore".

### Parbegin - Parend with Semaphore :-

Semaphore a,b,c,d,e,f,g = {0};

Parbegin

begin S1;

Parbegin

v(a);

v(b);

Parend

end;

begin P(a); S2; S4;

Parbegin

v(c);

v(d);

Parend

end;

begin P(b); S3;

v(e);

end

begin P(c); S5;

v(f);

end

begin P(d); P(e); S6; v(g); begin P(f); P(g); S7; end.

Parend



### Implementation with fork & Join :-

c1=2; c2=2; L1: S3;

S1;

fork l1;

S2;

S4;

fork l2;

S5;

go to L3;

L1: Join c1;

S6;

L3: Join c2;

S7;

```
Eg:- void P(void) { A; B; C; } void Q(void) { D; E; } main() { Parbegin(); P(); Q(); Parenend(); }
```

What are the possible outputs for the above code:

(a) A,B,C,D,E

(b) D,E,A,B,C

(c) A,B,C,E

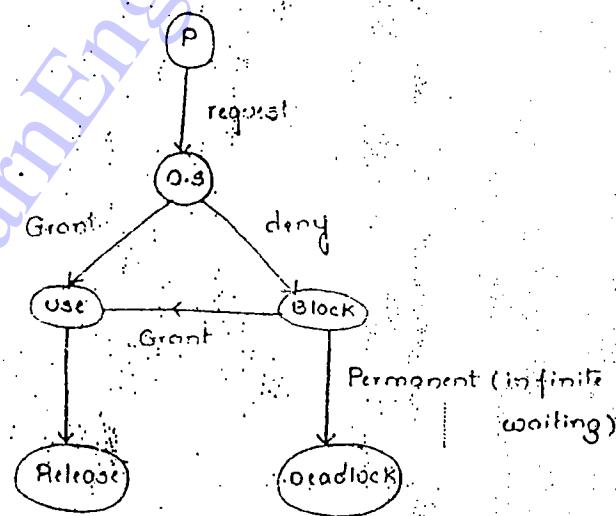
(d) D,A,C,E,B

(e) E,C,B,D,A

A,B,C and D,E are sequential actions. C,B cannot be happen. So, only first three are possible outputs.

## DEADLOCK ( Lockingup )

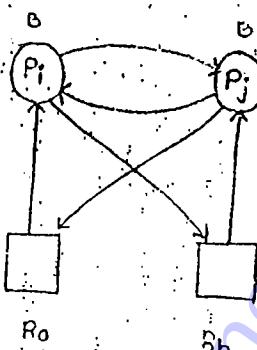
- \* Two (or) more processes are said to be involved in deadlock, iff they wait for the happening of an event, which would never happen.
- \* System model (for avoiding deadlocks)
  - \* 'n' processes ( $P_1, \dots, P_n$ )
  - \* 'm' resources
- \* Starvation  $\Rightarrow$  long waiting.
- \* Deadlock  $\Rightarrow$  Infinite waiting.



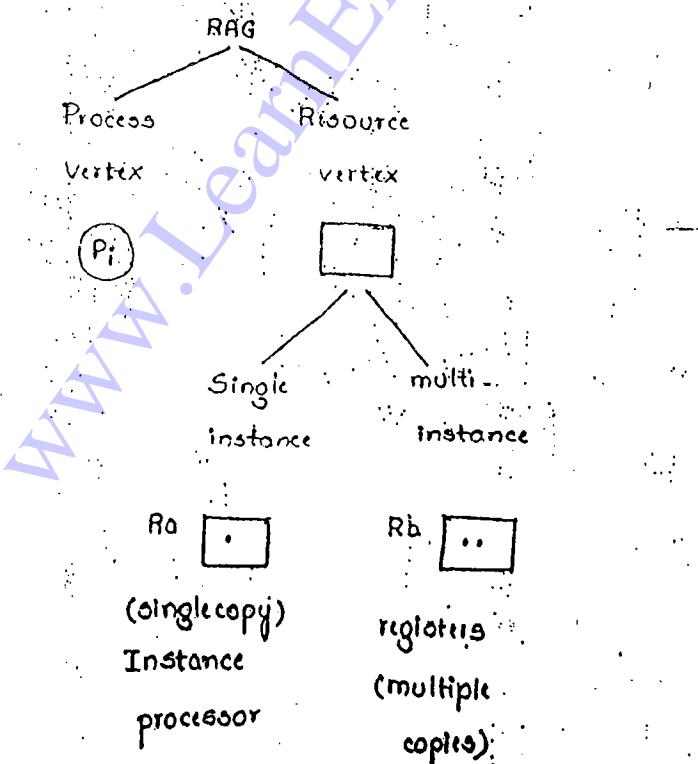
- \* Processes make a request for resources. If the resources are available, then it is granted by O.S. If it takes some time for resource allocation, then process gets blocked. Blocked process requests for the resources, after certain time, and if again not granted, then the process moves to deadlock state.

Necessary conditions for occurring deadlock :-

- (1) Failure of mutual exclusion.
- (2) Hold & wait.
- (3) No-preemption (process won't release their resources).
- (4) circular wait.



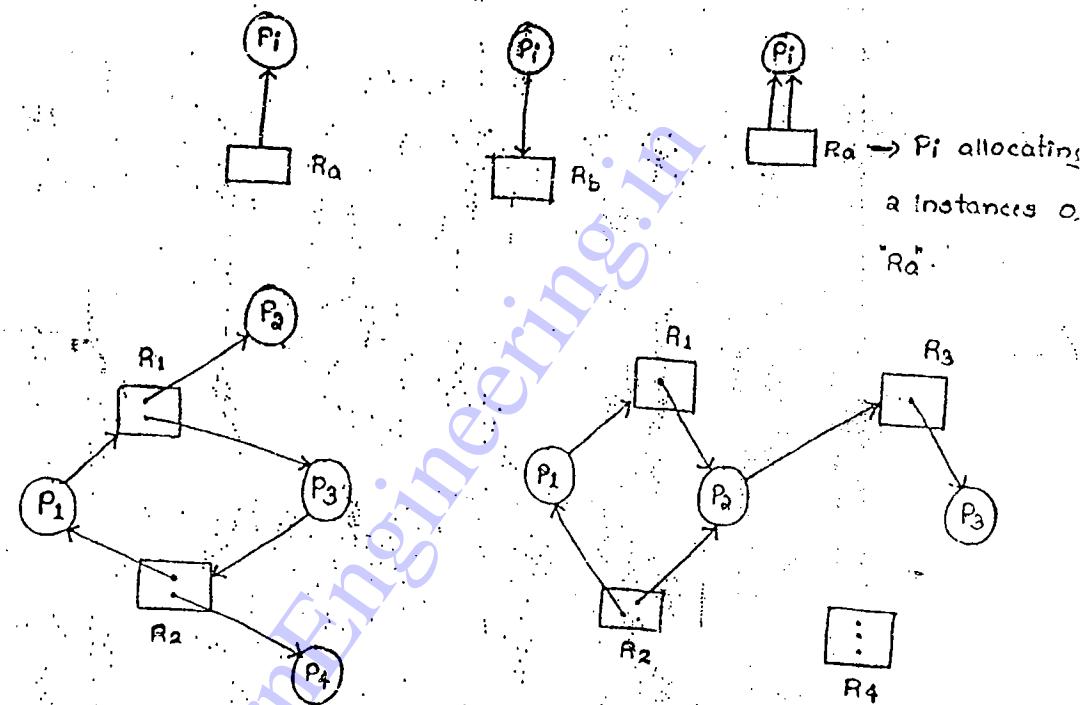
Resource Allocation Graph (RAG) / Multigraph :-



Edges :

Assign edge

Request edge



P<sub>1</sub> gets blocked, since it requests 'R<sub>1</sub>', which in turn R<sub>1</sub> gets accessed by P<sub>2</sub>. Similarly, P<sub>2</sub> also gets blocked.

- \* If RAG has single instance and multiple instance.
- \* In multiple instances, there is a chance of occurring deadlock, because they form cycles (some times).
- \* Single instances may also cause deadlock, if the processes accessed in the cycle form.

### Type - 1

- \* Deadlock prevention
- \* Deadlock avoidance  
(Banker's algorithm)
- No deadlock happens  
occurs.

### Type - 2

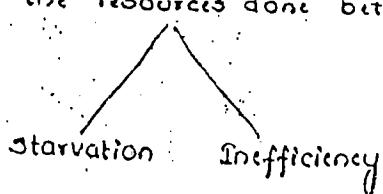
- \* Deadlock detection and recovery
- Deadlock happens

### Type - 3

- \* Deadlock Ignorance  
(Ostrich Algorithm)  
↓  
(Deserts)

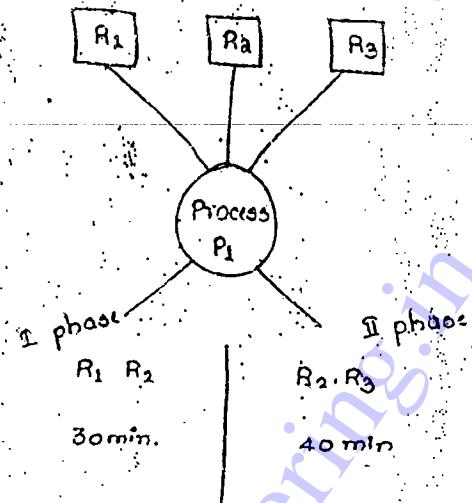
#### Deadlocks prevention :-

- \* Try to dissatisfy one/more of the four necessary conditions.
  - (1) mutual exclusion
  - (2) ! ( Hold & wait ) ;  
Either "hold" (or) "wait" (but not both)
- \* Requests & de-allocation of all the resources done between the process commencements;



#### mutual Exclusion :-

- \* Holding only one process within the critical section.



Consider,

- \* A process 'p<sub>1</sub>' which needs to access the resources 'R<sub>1</sub>', 'R<sub>2</sub>' and 'R<sub>3</sub>'. If a resource 'R<sub>1</sub>' and 'R<sub>2</sub>' are accessed and another time, again 'R<sub>2</sub>' needs to be accessed. So, the following situation holds :-
- Hold & wait :-
- \* Process 'p<sub>1</sub>' in first phase, requires R<sub>1</sub> and R<sub>2</sub> resources, and the time span allotted to phase-I is 30min to complete.
- \* After the completion of phase-I, process 'p<sub>1</sub>' releases the resource 'R<sub>1</sub>' but holds the resource 'R<sub>2</sub>' (because it again requires the same resource at phase-II also), and requests for the resource "R<sub>3</sub>". This situation is considered as "Hold & wait".
- \* In order to avoid this problem, we have two types of solutions :-

### I<sup>st</sup> Approach :-

- \* Process, in phase-I, requests all the resources whatever it needs.

$$\text{Eg.:- } P_1(R_1, R_2, R_3)$$

- \* They doesn't wait for all the resources to be accessed.

### II<sup>nd</sup> Approach :-

- \* Release all the resources before making a new request.

$$\text{Phase - I} \Rightarrow P_1(R_1, R_2)$$

$$\text{Phase - II} \Rightarrow P_1(R_3)$$

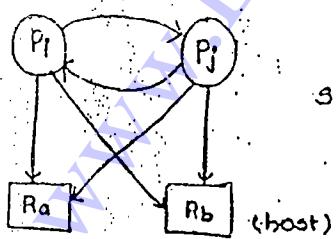
- \* After the completion of phase-I, release all the resources even though, it needs the same resource at phase-II.

- \* Then, again in phase-II, the resources are requested newly (if already used in phase-I).

### (c) No pre-emption :-

- \* Allocates the processes to get preempted.

Self forcefully



Self release  $\Rightarrow$  It requests the resources that are not available, so that process releases other resources that are already available

Consider, P1, which already accessed 'Ra', it requests for Rb, but Rb is not available, at that time. So, 'P1' releases 'Ra' resource (in the sense, it may be useful for others) after completion of other process, it will execute.

22/07/2010

The today

F Prevention (restrictive):

(d) circular wait:

"Total order relation on all processes for requesting of resources."

Incl dec.

P<sub>1</sub>-8

P<sub>2</sub>-4

P<sub>3</sub>-12

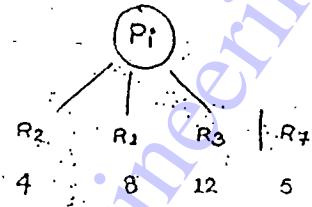
P<sub>4</sub>-18

P<sub>5</sub>-25

P<sub>6</sub>-9

P<sub>7</sub>-5

f(R) → I'



Starvation occurs when there  
is a release on."

P<sub>i</sub> requests in any order

(i.e., either in incl dec).

If no order is followed,

any sequence is followed, P<sub>i</sub>  
must release all resources.

Avoidance :-

(Banker's Algorithm):

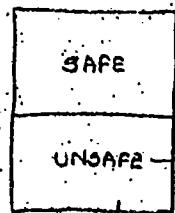
\* Safety Algorithm

\* Resource - Request

Objective :-

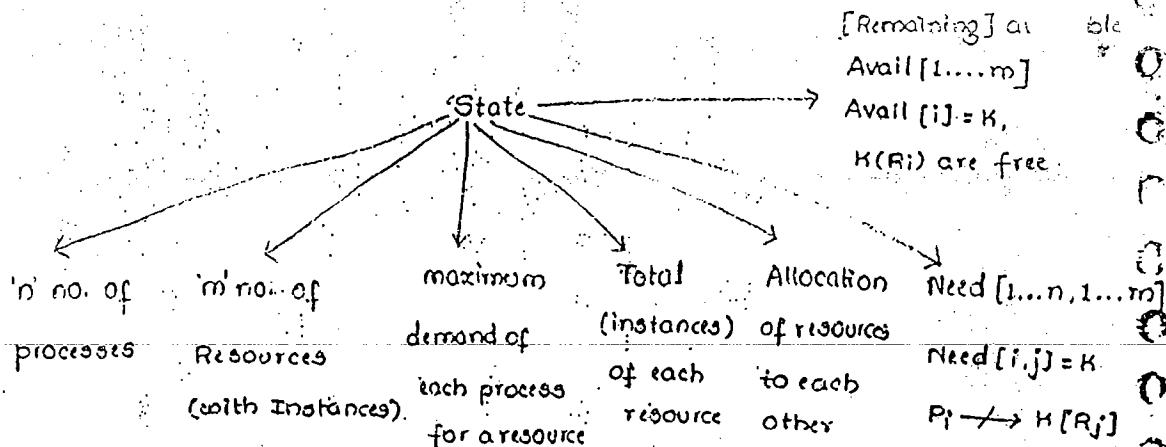
\* Obtain a system always in a SAFE  
mode.

+ It is an "A priori Info" algorithm; It is based on state of the system.



no deadlock

occurs.



$\text{Alloc}[1:n, 1:m]$

$\text{Max}(1:n, 1:m)_{\text{max}}$

$\text{Alloc}[1,j] = K$

$\text{Max}(i,j) = k$

$P_i \leftarrow K(R_j)$

$P_i \xrightarrow{\text{max}} K(R_j)$

Eg :  $n=5$  ( $P_1$  to  $P_5$ ).

$m=1, R = 28$  (Total)

$\text{Need} = \text{max} - \text{Alloc}$ .

|       | R<br>(max) | R<br>(Alloc) | (Need)         |                | (Avail)<br>R |
|-------|------------|--------------|----------------|----------------|--------------|
|       |            |              | R <sub>i</sub> | R <sub>j</sub> |              |
| $P_1$ | 10         | 5            | 5              | 3              |              |
| $P_2$ | 20         | 10           | 10             | 4              |              |
| $P_3$ | 3          | 1            | 2              | 8              |              |
| $P_4$ | 8          | 4            | 4              | 13             |              |
| $P_5$ | 12         | 5            | 7              | 18             |              |
|       |            |              |                |                | 28           |

To :  $\langle P_1, P_4, P_3, P_2, P_5 \rangle$

safe-sequence

Avail = current avail + allocation.

\* System is said to be SAFE, iff the needs of all the processes satisfied with the available resources in some (or) the order of (i.e. called SAFE sequence) otherwise, the system is said to be DEADLOCK.

But not really a deadlock to indicate warning

| P              | max |   |   | Alloc |   |   | need |   |   | Avail |   |   | Alloc<br>(A+B+C) |
|----------------|-----|---|---|-------|---|---|------|---|---|-------|---|---|------------------|
|                | A   | B | C | A     | B | C | A    | B | C | A     | B | C |                  |
| P <sub>0</sub> | 5   | 3 | 0 | 0     | 1 | 0 | 4    | 3 | 2 | 3     | 2 | 2 | 10 - 7 = 3       |
| P <sub>1</sub> | 3   | 2 | 2 | 0     | 0 | 0 | 1    | 2 | 2 | 5     | 3 | 2 | 5 - 2 = 3        |
| P <sub>2</sub> | 4   | 0 | 2 | 0     | 0 | 2 | 6    | 0 | 0 | 7     | 4 | 3 | 7 - 6 = 1        |
| P <sub>3</sub> | 2   | 2 | 2 | 0     | 1 | 1 | 0    | 1 | 1 | 7     | 4 | 5 |                  |
| P <sub>4</sub> | 4   | 0 | 3 | 0     | 0 | 2 | 4    | 3 | 1 | 7     | 5 | 6 |                  |
|                |     |   |   |       |   |   |      |   |   | 10    | 5 | 7 |                  |

To :  $\langle P_1, P_3, P_4, P_0, P_2 \rangle \Rightarrow \text{safe.}$

Initial availability = (3,3,2), O.S have enough resources to grant to P<sub>1</sub>, be it first calculates any problem if user gives this resource, It runs the "safety algorithm" in this situation.

T<sub>1</sub> : Resource Request algorithm :-

"Request by the resources are granted or not" is being identified to this algorithm.

T<sub>1</sub> : P<sub>1</sub> :  $\rightarrow (1, 0, 0) \quad (\text{Req}_1) \rightarrow \langle P_1, P_3, P_4, P_0, P_2 \rangle \Rightarrow \text{safe.}$

T<sub>1</sub> : P<sub>4</sub> :  $\rightarrow (0, 0, 0) \quad (\text{Req}_4) \rightarrow \text{blocked.}$

It is blocked because it doesn't satisfy the Availability.

$T_1 : \langle 0,0,0 \rangle \rightarrow P_0$

|       | max   | Alloc | need  | Avail |
|-------|-------|-------|-------|-------|
| $P_0$ | 4 6 3 | 0 1 0 | 7 4 3 | 2 3 0 |
| $P_1$ | 3 2 2 | 3 0 2 | 0 2 0 |       |
| $P_2$ | 9 0 2 | 3 0 2 | 6 0 0 |       |
| $P_3$ | 2 2 2 | 2 1 1 | 0 1 1 |       |
| $P_4$ | 4 3 3 | 0 0 2 | 4 3 1 |       |

### Resource Request Algorithm :-

1. Request  $\leq$  Available
2. Request  $\leq$  Need
3. (Assume request is satisfied)

$$\text{Available} = \text{Available} - \text{request}$$

$P_1 \rightarrow \langle 1,0,2 \rangle$  Request

$P_4 \rightarrow \langle 3,3,0 \rangle$

$P_0 \rightarrow \langle 0,2,0 \rangle$

$$\text{Allocation} = \text{Allocation} + \text{request}$$

$$\text{Need} = \text{Need} - \text{Request}$$

4. If result is safe, allocate resource.

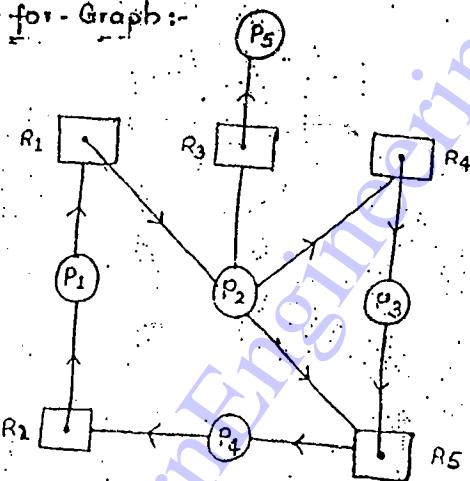
### III Deadlock Detection & Recovery :-

occurs when system performance is low.

If many blocked processes are present.

#### (a) Single Instance of a resource :-

##### wait-for-Graph :-



Among 5 processes, 4 are blocked. So, we apply deadlock detection algorithm.

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$

Cycle  $\Rightarrow$  Deadlock

Cycle  
Detection

Algorithm  
is good

#### IV multi-Instance of a resource :-

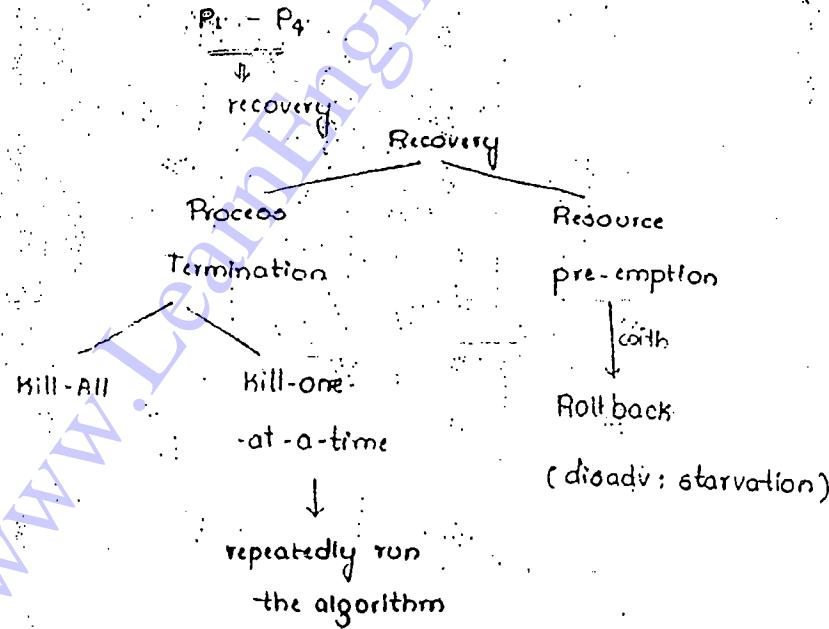
$$n=5, m=8, (A, B, C) = (7, 8, 6)$$

| To             | Alloc | Req   | Avail |           |
|----------------|-------|-------|-------|-----------|
|                | A B C | A B C | A B C |           |
| P <sub>0</sub> | 0 1 0 | 0 0 0 | 0 0 0 | 7 - 7 = 0 |
| P <sub>1</sub> | 2 0 0 | 2 0 2 | 0 1 0 | 8 - 2 = 6 |
| P <sub>2</sub> | 3 0 3 | 0 0 0 | 3 1 3 |           |
| P <sub>3</sub> | 2 1 1 | 2 0 0 | 5 0 4 |           |
| P <sub>4</sub> | 0 0 2 | 0 0 2 | 5 0 6 |           |

To check safe mode :  $\langle P_0, P_1, P_2, P_4, P_1 \rangle$

T<sub>5</sub> :  $P_0 \rightarrow \langle 0, 0, 1 \rangle$

$\langle P_0, \dots \rangle \times \text{Deadlock}$



Kill- All :-

Adv :-

\* No need for detection again.

Disadvantage :-

\* Again, it must be started from beginning.

\* To overcome, this, we have "Killing

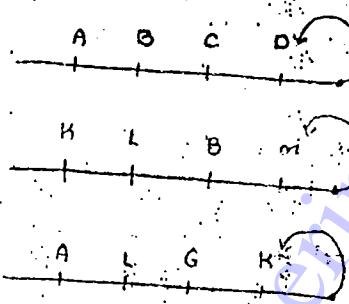
graciously".

One-at-a-time:

Adv:

- \* Save the other processes to resume.
- \* Need for detection algorithm as

Resource pre-emption:



Roll-back:

Repeatedly, a process is detected to have deadlock and re-start

then the starvation occurs

Deadlock handling strategies are not used on desktop systems and "Deadlock Ignorance" is considered :-

\* Robustness. not important tasks

\* follows oblivious algorithm (in missile, real time, Industrial OS)

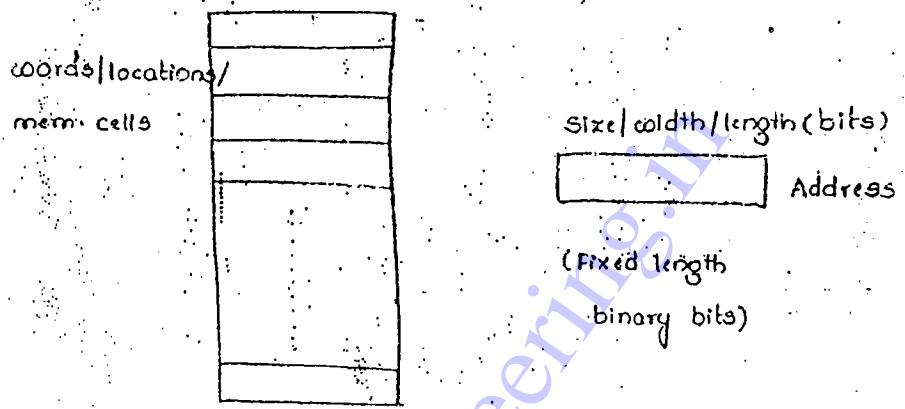
The strategies are used in real time systems where there is no single bit errors

↳ follows deadlock prevention

## Memory Management

### Abstract view of memory

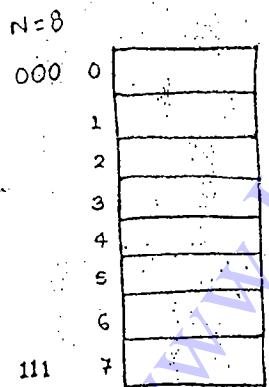
(1-D Array of words)



$N$  = Total no. of words.

$n$  = Address (in bits)

$m$  = width / size (bits) of words



$n=3$  bits

$N = 8$  words

$\downarrow$   
 $n=6$  bits ( $2^6$ )

$N = 512$  Mwords

$\downarrow$   
 $= 2^9 + 00$

$= 2^9$  bits

$N = 4$  GW

$\downarrow$   
 $n = 2^2 + 30$

$= 32$  bits  $\Rightarrow 2^{32} = 4$  GW

$$2^5 = 32 \text{ W}$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 512$$

$$2^9 = 1024$$

$$2^{10} = 1024 \approx 10^3 = 1K$$

$$2^{20} = 1024 * 1024 \approx 10^6 = 1M$$

$$2^{30} \approx 10^9 = 1G$$

$$2^{40} \approx 10^{12} = 1T$$

$$N = 2^m$$

$$512 \Rightarrow 2^9$$

$$n = \lceil \log_2 N \rceil$$

memory specification

word Byte Bit

I  
n = 23 bits.

m = 16 bits (1 word = 2 bytes)

|       | capacity                                             | Address     |                                                                                      |
|-------|------------------------------------------------------|-------------|--------------------------------------------------------------------------------------|
| words | 8 MW                                                 | n = 23 bits | $\theta = 2^3 \text{ MW}$                                                            |
| Bytes | $8M \times 2B$<br>$= 16MB$                           | n = 24 bits | $3 + 20 = 23$                                                                        |
| bits  | $16M \times 8 \text{ bits}$<br>$= 128 \text{ Mbits}$ | n = 27 bits | $16 = 2^4 \text{ MB}$<br>$4 + 20 = 24$<br>$128 = 2^7 \text{ mbits}$<br>$7 + 20 = 27$ |

II  
n = 256 MH  
m = 32 bits (1 word = 4 bytes)  $\Rightarrow 256M \times 4B = 1G$ .

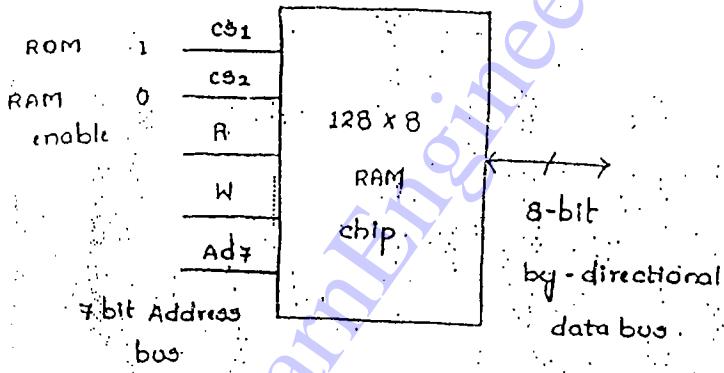
|       | Capacity                   | Address     |         |
|-------|----------------------------|-------------|---------|
| words | 256 MW                     | n = 28 bits | 32 bits |
| bytes | $256M \times 4B$<br>$= 1G$ | n = 30 bits | $2^5$   |
| bits  | 8G bits                    | n = 33 bits |         |

$N = 256 \text{ Gbits}$

$$m = 8 \text{ bits} \quad (\text{1 coord} = 8 \text{ bytes}) \Rightarrow 256 \text{ G} \times 8 \text{ B}$$

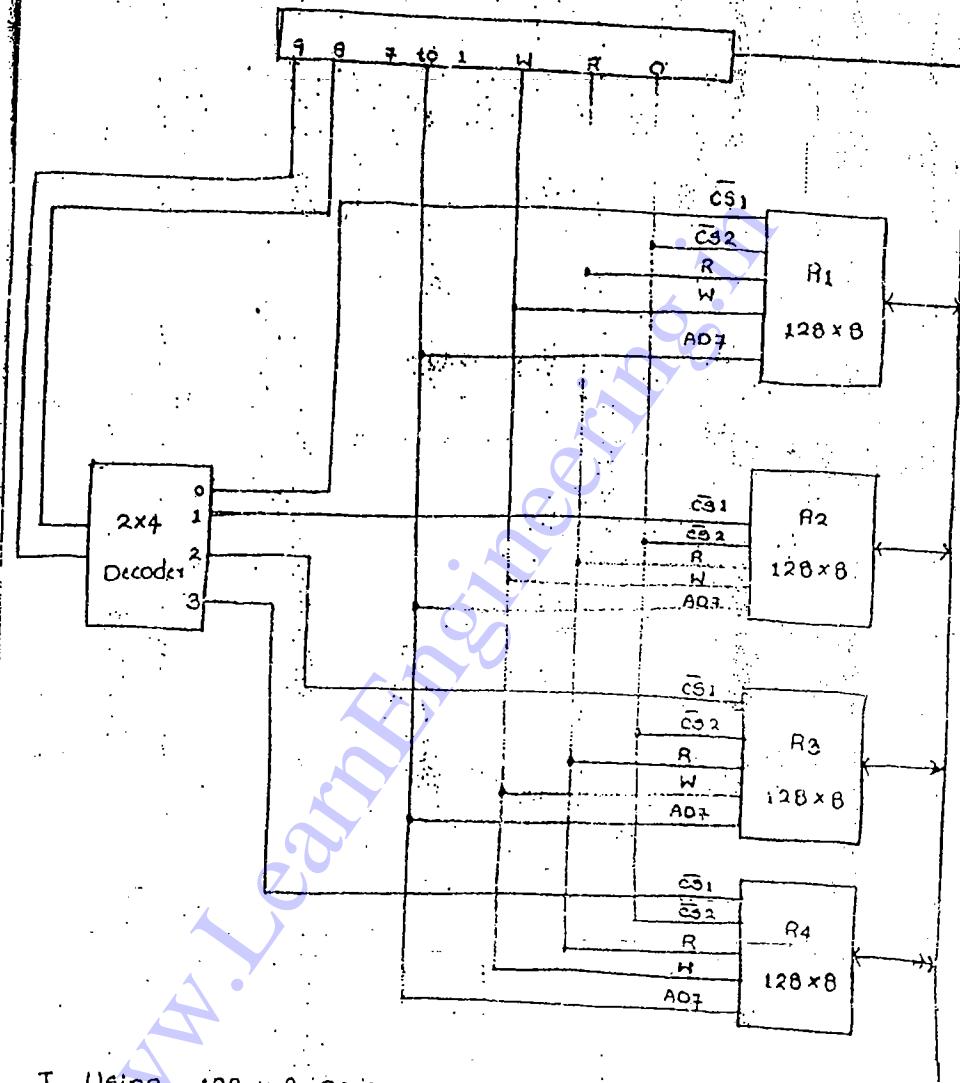
|       | capacity                                              | Address | $\frac{2^8 \times 2^{30} \times 2^6 \times 2^3}{2^6} = 2^{32} = 4 \text{ GH}$ |
|-------|-------------------------------------------------------|---------|-------------------------------------------------------------------------------|
| words | 4GW                                                   | 32 bits |                                                                               |
| Bytes | $4 \text{ G} \times 8 \text{ B}$<br>$= 32 \text{ GB}$ | 36 bits |                                                                               |
| Bits  | 256 Gbits                                             | 38 bits |                                                                               |

### RAM chip organization :-



$$\text{No. of chips} = \frac{\text{memory desired}}{\text{chip size}}$$

$$\frac{512 \text{ MB}}{128 \text{ MB}} = \frac{2^9}{2^7} = 2^2 = 4$$



I Using 128 x 8 RAM

$$16 \text{ KB} = ?$$

+ no. of words  
increasing

(i) chips = 128

(ii) n = 14 bits

(iii) Decoder = 4 x 128

$$\frac{16 \text{ KB}}{128} = \frac{2^{14}}{2^7} = 2^7 = 128$$

### III Using $128 \times 1$ RAM chip

128 bytes

(i) chips = 8

(ii)  $n = 7$  bits

16 KB

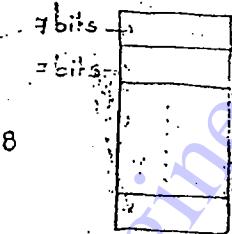
(i) chips = 1024

(ii)  $n = 14$  bits

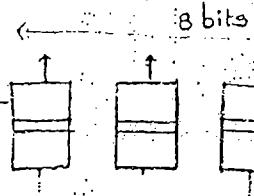
(iii) Decoder =  $7 \times 128$

$128 \times 1$  RAM

$$128 \times 8 \text{ bytes} = 1024 \text{ bits}$$



It contains, each word is 1 bit



128 B

$$\frac{128 \times 8}{128} = 1 \times 8 = 8$$

Since, memory and data size are same, there is no need of decoder.

### IV Using $128 \times 1$ RAM chip

16 KB

(i) chips = 1024

(ii)  $n = 14$  bits

(iii) Decoder =  $7 \times 128$

$128 \times 1$  RAM

$128 \times 8$

$\frac{16 \text{ KB}}{128 \times 1}$

$$\frac{2^{14} \text{ bits}}{2^7} = 2^7 = 128 \times 8 \text{ bits}$$

$$\frac{16 \text{ KB}}{128 \times 1} = \frac{2^4 \times 2^{10} \text{ B}}{2^7} = 2^7 = 1024 = 1024$$

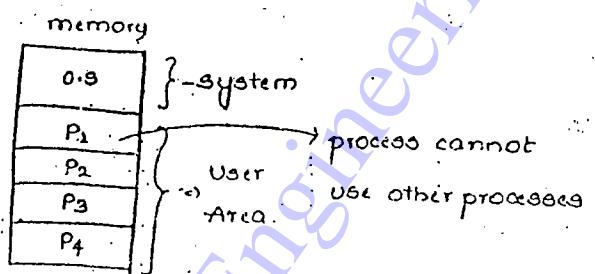
## Memory Manager

### Functions :-

- \* Allocation
- \* Protection
- \* Free space management
- \* Deallocation

### Goals :-

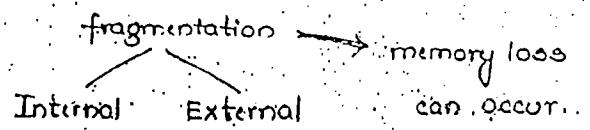
- \* Efficient utilization of space.
- \* Run larger program in smaller memory.



### Goals :-

#### (1) Space utilization:-

Keeping fragmentation at each level, so that there is a chance of effective utilization of space.



#### (2) Run larger program in smaller memory,

Program = 100 KB (block)

↓  
Using virtual memory / overlays.

Eg.:- Run the data within pendrive (eg:- 60KB)

## Memory management

### Techniques

#### Contiguous (C)

\* centralized  
program, as a unit is completely stored

\* Overlays

\* partitioning

#### Non-contiguous (NCA)

##### \* Decentralised

program portions are distributed in memory.

\* paging

\* segmentation

\* segmentation - paging

\* virtual memory.

24/07/2010

Saturday

(1) Overlays (larger programs run in smaller space):-

2 pass Assembler

pass 1 : size - 50 KB

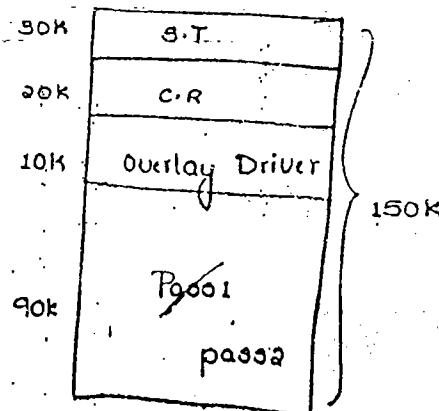
pass 2 : size - 80 KB

Symbol Table

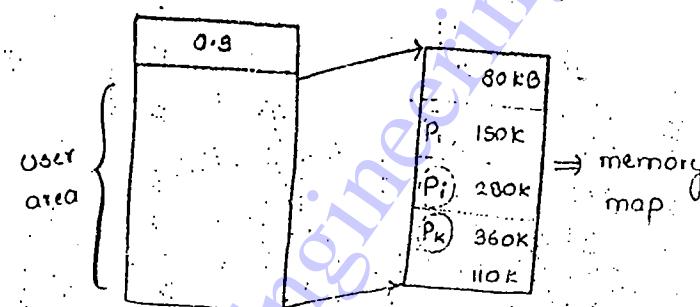
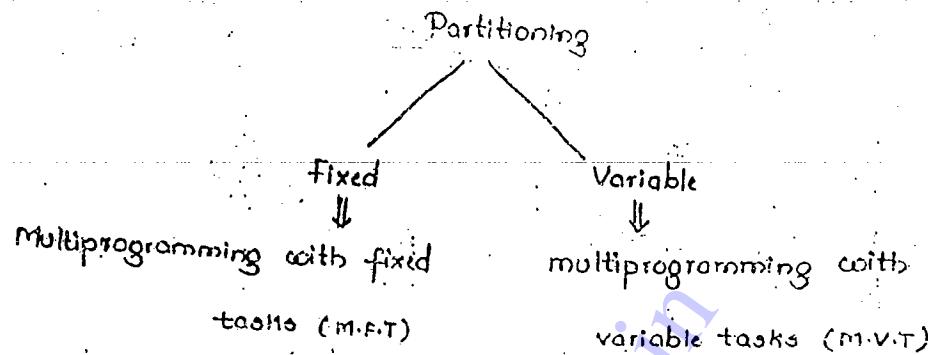
for construction of passes : 30 KB

C.R (common Rupt(s)) : 20 KB

200 KB

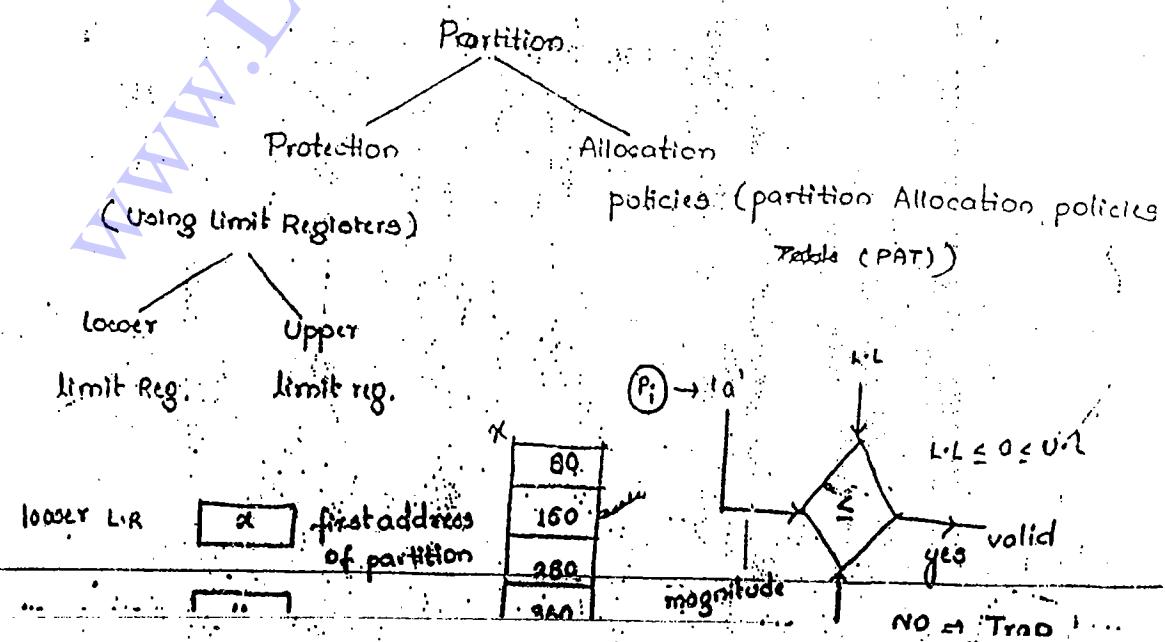


loading programs  $\rightarrow$  overlay driver

(2) partitioning :-

fixed (M.F.T)  $\Rightarrow$  no. of partitions are fixed  
and of different sizes

Since 1 partition = 1 process



FS:  $P_{\text{req}} = 100 \text{ KB}$

|     |
|-----|
| 60  |
| 150 |
| 280 |
| 360 |
| 20  |

### Partition Allocation policies :-

#### (i) First fit (F.F) :-

- \* Allocate the process in the first free begin of the partition.

To include  $P_{\text{req}} = 100 \text{ KB}$ , the partition of  $P = 150 \text{ KB}$  is selected as it is the first partition of required size.

#### i) Best fit (B.F) :-

- \* Smallest begin of free partition.
- \* Best available fit.
- \* Best fit only.

#### Best available fit :-

Among available partitions, select the smallest best available.

#### Best fit only :-

It selects for best fit only, if it is not available, it waits for that partition, and after available it takes.

#### Next fit (N.F) :-

Same as first fit, but it doesn't start from first partition, but it starts from the place where the last partition ended up (i.e., last process's partition ended point).

#### Advantage :-

- \* Searching becomes faster than first fit.

(iv) coast fit (C.F.):

Largest free partition  $\Rightarrow$  Selecting the largest available free partition, and get placed in that partition.

Performance of MFT :-

(1) fragmentation:

\* Internal

\* External  $\Rightarrow$  when unable to accommodate a process of size even though we have sufficient size of memory.

1 partition  $\rightarrow$  1 process

(contiguous allocation)

Eg:- Process = 180 KB

|        |
|--------|
| 80 KB  |
| 150 KB |
| *      |
| 280 K  |
| *      |
| 360 K  |
| 110KB  |

$$\begin{aligned} & 80 + 150 + 110 \\ & = 340 \text{ KB} \end{aligned}$$

If we have a new process of 180 KB to place into memory, but there is no certain partition to get fit into this process even though it has total of 340 KB free. So, it is called as External frag.

(2) Maximum process size: (or) maximum partition size  $\Rightarrow$  no flexibility.

(3) Degree of multiprogramming:-

Restricted to no. of partitions (can't be  $> n$ )

$\downarrow$   
no flexibility

(4) Best fit is superior to all.

Inorder to overcome all these problems, we have M.V.T.

### M.V.T (variable partitions)

- ⇒ Dynamic approach
- ⇒ No user area for partitioning

| PCM (Partition Control memory) |       |
|--------------------------------|-------|
| P <sub>1</sub>                 | 80K   |
| P <sub>2</sub>                 | 110K  |
| P <sub>3</sub>                 | 75 K  |
| P <sub>4</sub>                 | 180 K |
| P <sub>5</sub>                 | 260 K |
| P <sub>6</sub>                 | 320K  |
|                                | 300 K |

### PCM (Partition Control memory)

Used for storing the control information of particular partitions.

\* No use of limit registers in M.V.T for protection.

\* Limit value  
Size  
Status  
L.L.U.L

Eg.: If a new process of 70KB is needed to fit into memory, then if we take first fit, the 80KB partition is divided as 70KB for the new process & rest of 10KB is free, we overcome the fragmentation problem here.

Process = 70 KB  
(new)

### Performance of M.V.T

#### (1) fragmentation:-

- \* External
- \* No Internal.

#### (2) Max. process size $\Rightarrow$ flexible

no max. part. size.

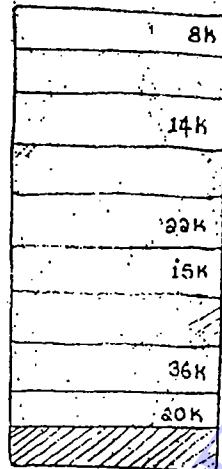
| PCM  |                      |
|------|----------------------|
| free | 400 KB               |
|      | 80KB                 |
|      | 10KB                 |
|      | P <sub>2</sub> 110K  |
|      | 75 K                 |
|      | P <sub>4</sub> 180 K |
|      | 260 K                |
|      | P <sub>6</sub> 320K  |
|      | 300K (free)          |

(3) Degree of multiprogramming:

- \* No restriction to no. of partitions.

- \* flexible.

(4) coorot fit performs better, and best-fit performs worst.



Ques:

First fit = 8K

Best fit = 8K

next fit = 36K

worst fit = 36K

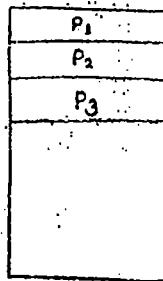
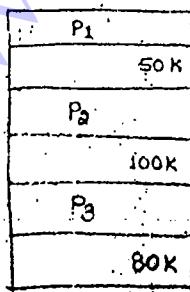
(ii) How many successive requests

of size 6K can be satisfied.

Ans: 17.

External fragmentation:

(i) Compaction (dynamic relocation of processes):



All the processes at different memory locations are compacted to one side and the free memory is now available.

\* process should be dynamically relocatable and can be processed / continued.

If cannot be run (on processes), then no compaction is followed.

### Disadvantages:-

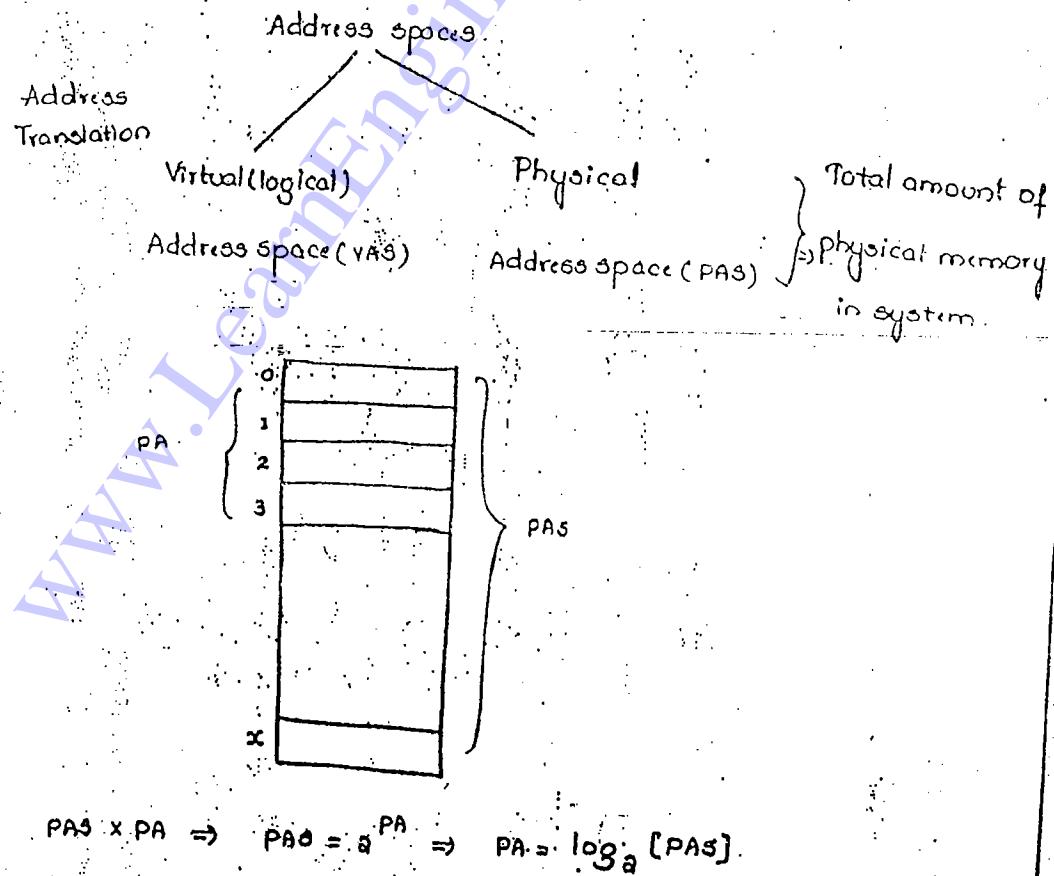
- \* longer time (Since everytime relocation is done).
- \* If done frequently, time is consumed.

### (2) Non-contiguous Allocation:-

- \* To overcome external fragmentation, we use non-contiguous allocation policies.

### Address space:-

- \* Space of coords / series of coords (act)
- \* A set of locations of memory is address space.



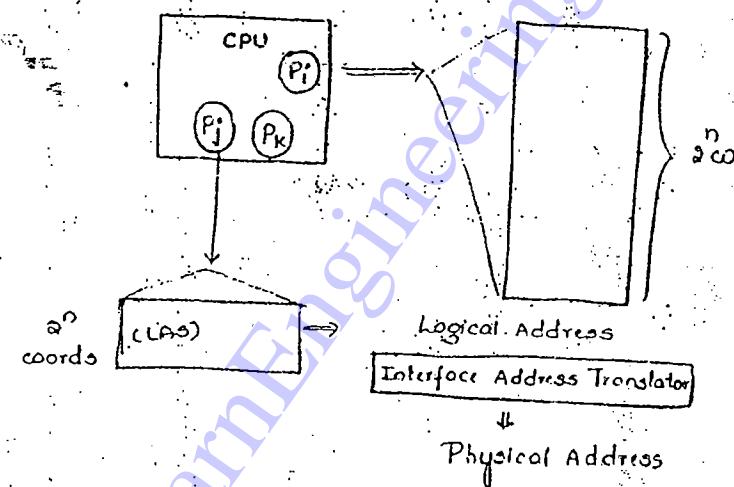
### Logical Address Space :-

\* Every process within the CPU has an assumption that it can access the whole memory available.

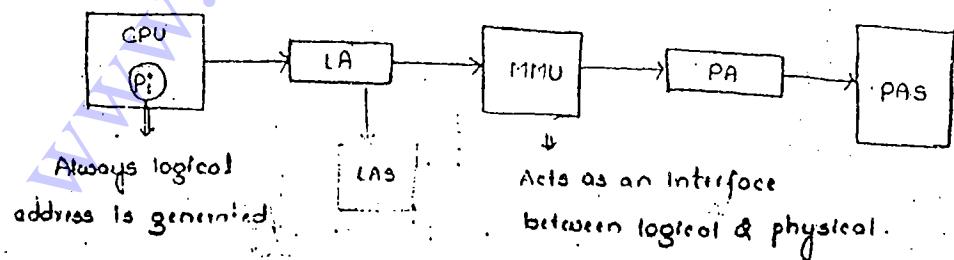
\* LAS must be translated to physical for instruction/data fetch.

$$\text{If } LA = 13 \text{ bits } (2^{13} \text{ K}) \quad LA = \log_2(LAS)$$

$$\text{Then, } LAS = 2^{LA}$$



### Architecture of non-contiguous Allocation policies :-

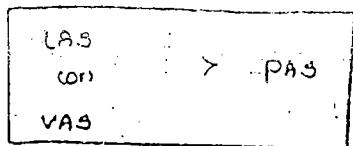


(1) Organisation of LAS

(2) Organisation of PAS

(3) Organisation of MMU (mem mgmt unit)

(4) Translation process



Because, it must have to support the concept of larger program storage in shorter memory.

Theoretically, all of them have different relation.

### Paging:-

#### Simple paging:-

$$LAS = 8KB, \quad PAS = 4KB, \quad LA = 13 \text{ bits}, \quad PA = 12 \text{ bits}.$$

#### (1) organisation of LAS:-

(i) LAS is divided into equal size pages.

(ii) Page size is a power of 2.

$$\text{(iii) No. of pages } (N) = \frac{\text{LAS}}{\text{Page size}}$$

$$\text{(iv) page no. } (p) = [\log_2 N]$$

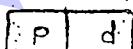
$$N = 2^p$$

$$\text{(v) page offset } (d, \text{ bits}) = [\log_2 PS]$$

$$PS = 2^d$$

(vi) Format of logical address:

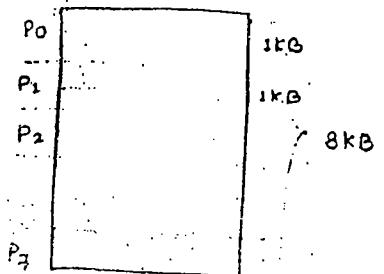
$$LA = 13 \text{ bits}$$



words

3      10

pages



### (2) Organisation of PAS:

(i) PAS is divided into equal size frames (page frames)

(ii) frame size = page size

frame size to same as page size

$$(iii) \text{No. of frames (m)} = \frac{\text{PAS}}{\text{PS}}$$

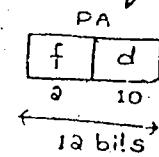
$$(iv) \text{frame no. ('f' bits)} = \lceil \log_2 N \rceil$$

$$N = 2^f$$

(v) frame offset = page offset

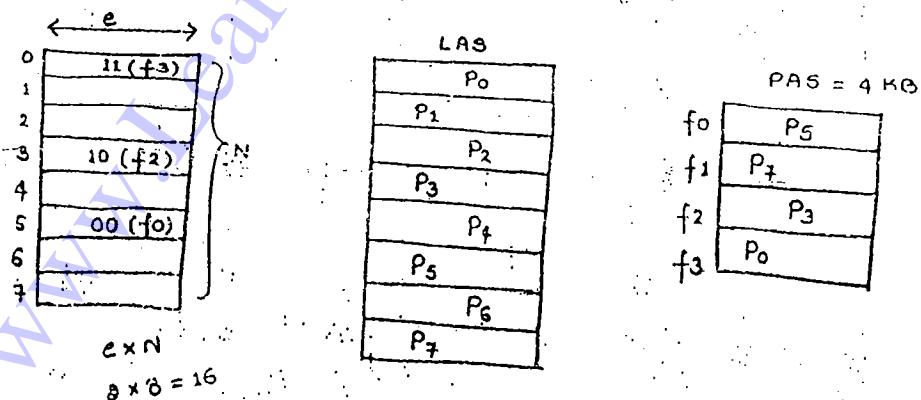
(vi) frame holds the page, therefore, frame offset is equal to page offset.

|            |
|------------|
| PAS = 5 KB |
| FA         |



### (3) Organisation of MMU (page table):

page map Table  $\Rightarrow$  Associated with processes.



\* No. of entries in P.T (P.T size) = No. of pages in LAS.

\* PT entries contains frame numbers.

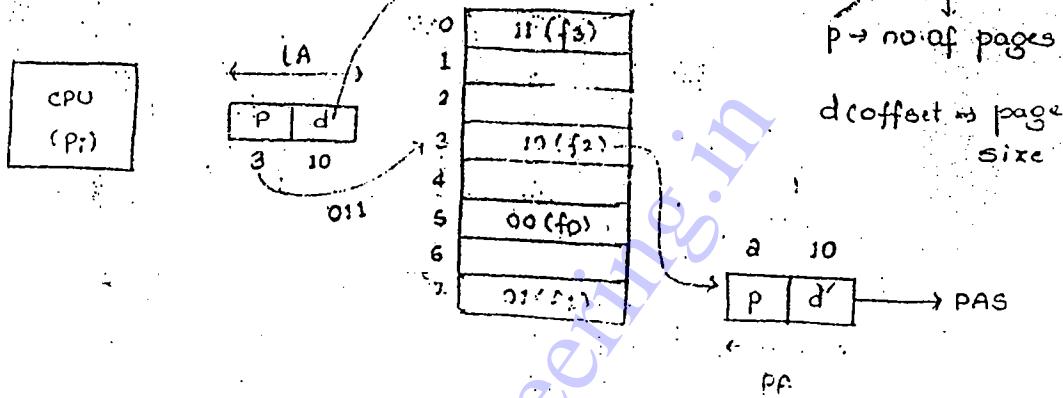
\* Associated with processes (Each process has its own page table)

\* page tables are stored in main memory.

\* P.T size (in bytes) =  $n * e$  bytes

$\rightarrow$  page table entry size in bytes

### Address Translation:



Eg 1 :-

$$LA = 32 \text{ MB}, \quad PA = 256 \text{ KB} \quad N = ?, \quad m = ?, \quad p = ?, \quad f = ?, \quad d = ?$$

$$\text{page size} = 4 \text{ KB} \quad \text{P.T size} = ?$$

Sol:-

$$N = \text{no. of pages} = \frac{32 \text{ MB}}{4 \text{ KB}} = \frac{2^{25}}{2^{12}} = 2^3 = 8 \text{ K}$$

$$m = \frac{256 \text{ KB}}{4 \text{ KB}} = \frac{2^{18}}{2^{12}} = 2^6 = 64 \text{ B}$$

$$p = \text{page no. (depends on no. of pages)} = 13 \text{ bits} \quad (N = 8 \text{ K}) \\ = 2^3 + 10 \\ = 13$$

$$f = \text{frame no. (depends on m)} = 6 \text{ bits.}$$

$$d(\text{offset}) \text{ depends on page size} = 12 \text{ bits} \quad (4 \text{ KB}) \\ = 2^2 + 10 \\ = 12$$

$$\text{P.T size} = \text{no. of pages} = 8 \text{ K} \quad = 12)$$

$$\text{If } e = 4 \text{ B} \text{ then } \text{P.T size} = e * N$$

$$= 8 \text{ K} * 4 \text{ B}$$

$$= 32 \text{ KB}$$

Eg:- 2K pages & 512 frames.

If PAS = 4MB, e = 4B, N, M, P, f, LA, PA, d, PTS.

Sol:-

$$N = 2K$$

$$M = 512$$

$$PAS = 4MB$$

$$P = 11 \text{ bits} \quad (2K \Rightarrow 2^11 + 10 = 11)$$

$$PA = (2^2 + 20)$$

$$f = 9 \text{ bits} \quad (512 \text{ frames} \Rightarrow 2^9)$$

$$= 20 \text{ bits}$$

$$LA = 24 \text{ bits} = (16 MB)$$

$$\xrightarrow{24 \text{ bits}}$$

$$PA = 29 \text{ bits}$$

$$d = 13 \text{ bits} \quad (PA = 29, f = 9)$$

$$\xrightarrow{14 = 24 \text{ bits}}$$

$$P.T.S (\text{Bytes}) = N * e$$

$$\xrightarrow{11 \quad 13}$$

$$= 2K * 4B = 8 KB$$

$$\xrightarrow{11 \quad 13}$$

$$d = PA - f = 13$$

$$P.T.S (\text{Bytes}) = N * e$$

$$= 2K * 4B = 8 KB$$

$$d = PA - f = 13$$

$$P.T.S (\text{Bytes}) = N * e$$

$$= 2K * 4B = 8 KB$$

$$d = PA - f = 13$$

$$P.T.S (\text{Bytes}) = N * e$$

$$= 2K * 4B = 8 KB$$

Performance of paging:-

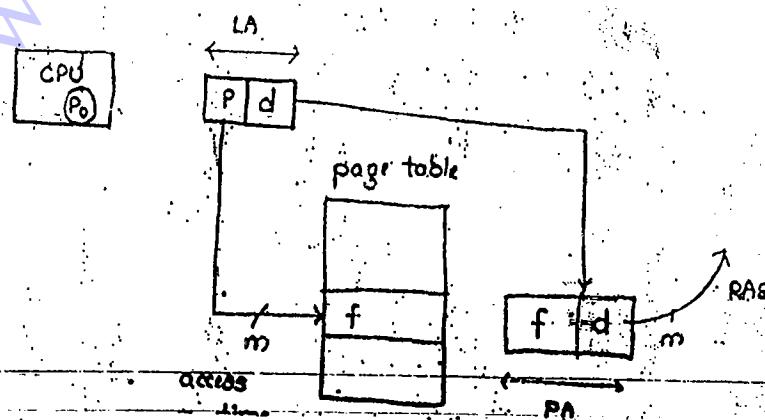
\* main memory access time (mmAT) =  $m$  ( $\mu s/n_s$ )

\* Effective memory Access Time (EMAT)

↓  
Decrease / Reduce of EMAT from  $m$  & bring closer to  $m$ .

Access Time:

= Amount of Time to read (or) write to memory.



### (iii) Paging with TLB (Translation Look-a-side Buffer):

If generated LA is not found within the cache (or TLB), it is called "TLB miss". If found, it is called "TLB hit".

$$\text{TLB Access time} = c$$

$$\text{where } [c \ll m]$$

$$\text{TLB page hit ratio } (\alpha) = \frac{\text{No. of hits}}{\text{Total references}}$$

$$\text{TLB miss ratio } (\gamma) = 1 - \alpha$$

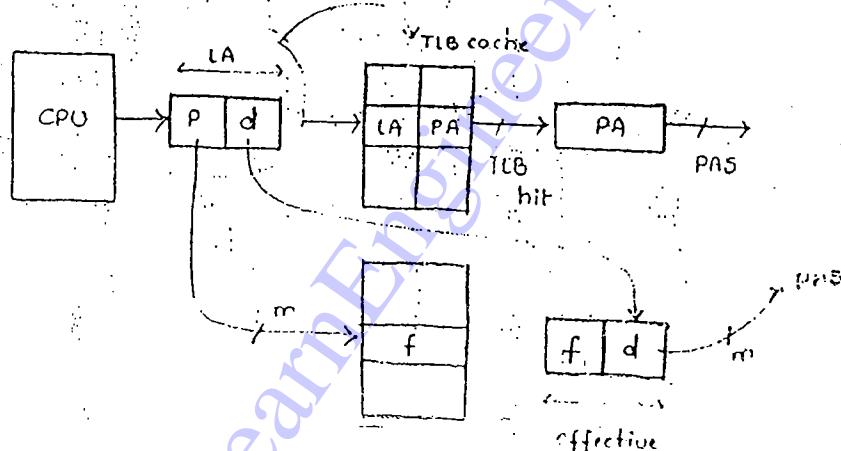
TLB is a cache contains

Set of physical & logical address

If generated LA is present, then

It is TLB hit, (or) it goes to page

table & generates physical address



$$\text{EMAT}_{(\text{SP} + \text{TLB})} = \alpha(c + m) + (1 - \alpha)(c + \alpha m)$$

↓                  ↓  
Success      failure.

Eg:- (i)  $m = 100 \text{ ns}$      $c = 20 \text{ ns}$      $\alpha = 90\%$

$$\begin{aligned}\text{EMAT} &= 0.90(20 + 100) + 0.1(20 + 100) \\ &= 9(120) + 10 = 130 \text{ ns.}\end{aligned}$$

(ii)  $\alpha = 10\%$

$$\begin{aligned}\text{EMAT} &= 0.10(20 + 100) + 0.9(20 + 100) \\ &= 10 + 9(10) = 100 \text{ ns}\end{aligned}$$

(additional time)  
Effective memory overhead :  $= m$ .

(simple paging)

SP - Simple  
paging

$$\boxed{\text{EMOH}} \quad (\text{s.P} + \text{TLB}) = \alpha(c) + (1-\alpha)(c+m)$$

\* By using page table, every time, we require "m" memory access time is overhead for minimising. This introduce TLB that access time is 'c' which is less than 'm' overhead.

\* In hit rate, Translation require "c+m", if miss occurs it takes  $c+m + c+m$

### \* Multi-level paging:

$$\text{LA/VA} = 32 \text{ bits}, \quad \text{P.S} = 4 \text{ KB}$$

$$\text{PT size} = 1 \text{ m} \quad B = 4 \text{ MB}$$

$$N = \frac{32}{2^{12}} = 2^{20} \\ = 1 \text{ M}$$



Larger page tables

$2^m$  are not desired.

Reduction of page table

Increasing

Multi-level

page size

Drawback :-

Increase Internal fragmentation.

(wastage of space)

Large : P.S = 1 KB

= 8 pages

Small : P.S = 8 B

= 512

Program **1024**

=  $1024 \approx 1 \text{ page}$

= 512

= 8 B

External fragmentation must be minimum.  
 P.T overhead } minimum.

\* LAS IVAS = 's' words

\* page Table entry = 'e' bytes

\* page size 'p' = ?

$$\text{P.T overhead (bytes)} = \left( \frac{s}{p} \right) * e$$

Internal fragmentation =  $\frac{p}{a}$

$$\text{Total overhead} = \left( \frac{p}{a} + \frac{s}{p} * e \right)$$

$$\text{To minimize/optimize} \Rightarrow \frac{d}{dp} = \left( \frac{p}{a} + \frac{s}{p} * e \right)$$

$$(min.) \Rightarrow 0 = \left( \frac{1}{a} - \frac{1}{p^2} s * e \right)$$

$$p^2 = ase$$

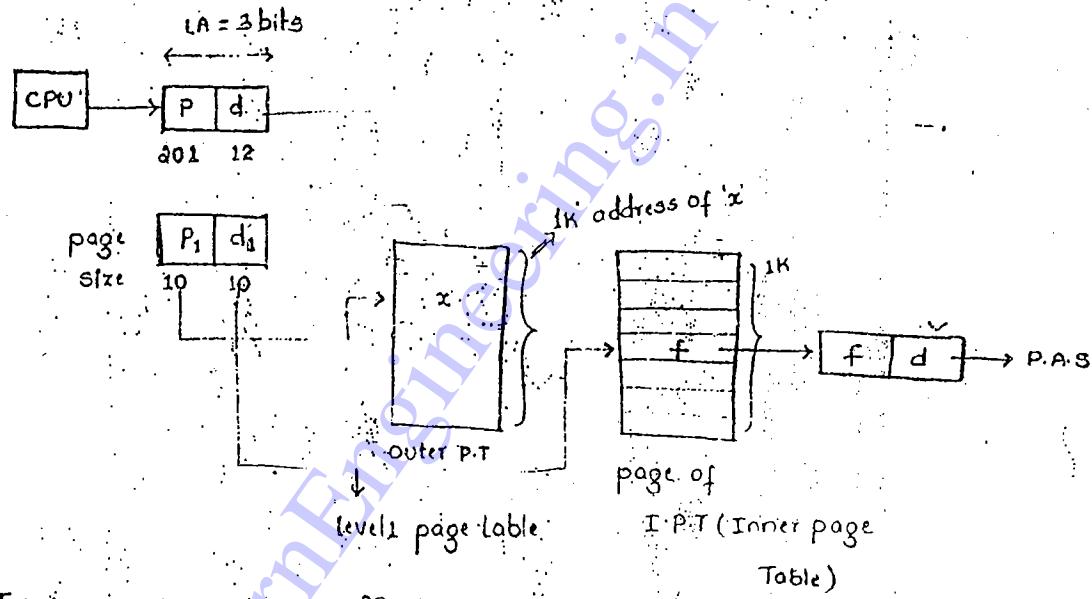
$$p = \sqrt{ase}$$

28/07/2016

wednesday  
=

Paging :

\* Dividing any address space into equal size units (pages)



$$\text{Total virtual memory} = 2^{32}$$

Each page size = 4 KB

$$\text{Total pages are } \frac{2^{32}}{4 \text{ KB}} = 2^{20} \Rightarrow 1M \text{ pages.}$$

Each page size is 4 KB. So, again apply paging on page table of each size.

$$\text{Total no. of entries} = \frac{1M}{1K} = 1K$$

Two level addresses = 20  $\rightarrow$  no. of pages

12  $\rightarrow$  offset.

Divide again pages of each size = 1K  $\Rightarrow$  It stores address of outer page for storing address it require 10 bits.

### Performance :-

\* Main memory access time = "m"

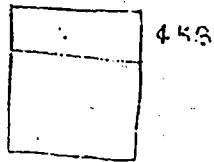
\* Effective access time =  $m + m + m = 3m$   
 ↓      ↓  
 Inner   Outer

\* Effective access time with 'n' level paging =  $(n+1)m$ .

For reducing access time Using TLB :

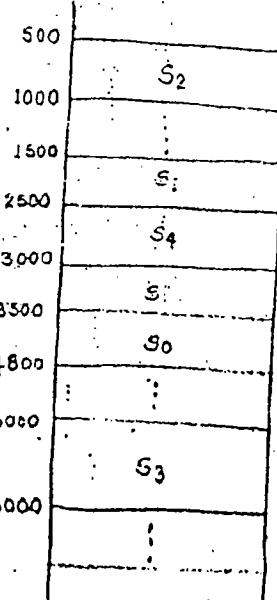
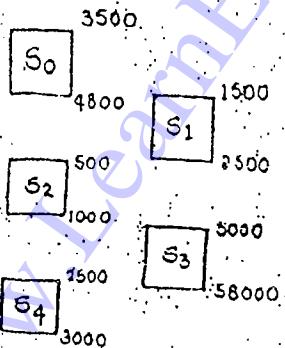
$$\text{TLB hit ratio} = \alpha \\ = \alpha(c+m) + (1-\alpha)(c+3m)$$

$$\text{For } n\text{-level pages} = \alpha(c+m) + (1-\alpha)(c+(n+1)m)$$



### Segmentation

Paging doesn't preserve user's view of memory allocation.

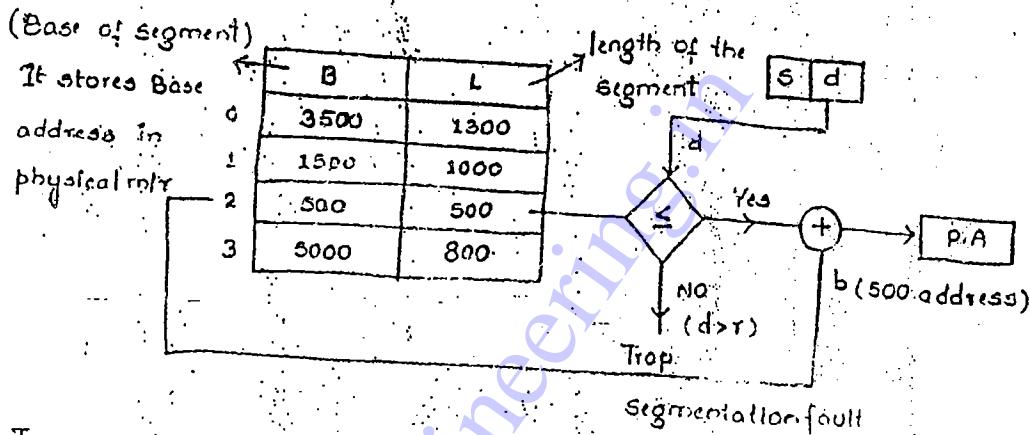


Segment is a piece of code.

(program, divided into unequal sizes

is called Segment (for equal size is  
called page(s)).

- (1) Divide the program into segments (VAM organisation).
- (2) Using policies place the segments.



In page table, all are equal sizes. So, no need to compare the required word present within limit or not. Here, unequal no. of partitions for checking the offset present in within the limit use magnitude comparator.

### Segmented paging :-

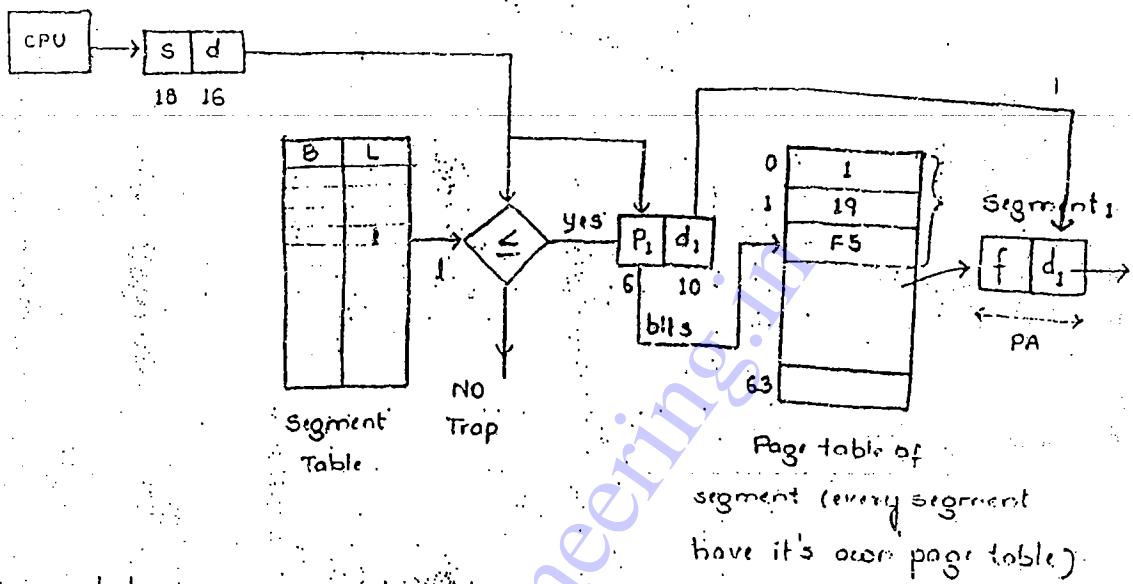
Introducing paging (on segmentation)

Virtual size = 34 bits,  $\langle s,d \rangle = \langle 18,16 \rangle$  bits

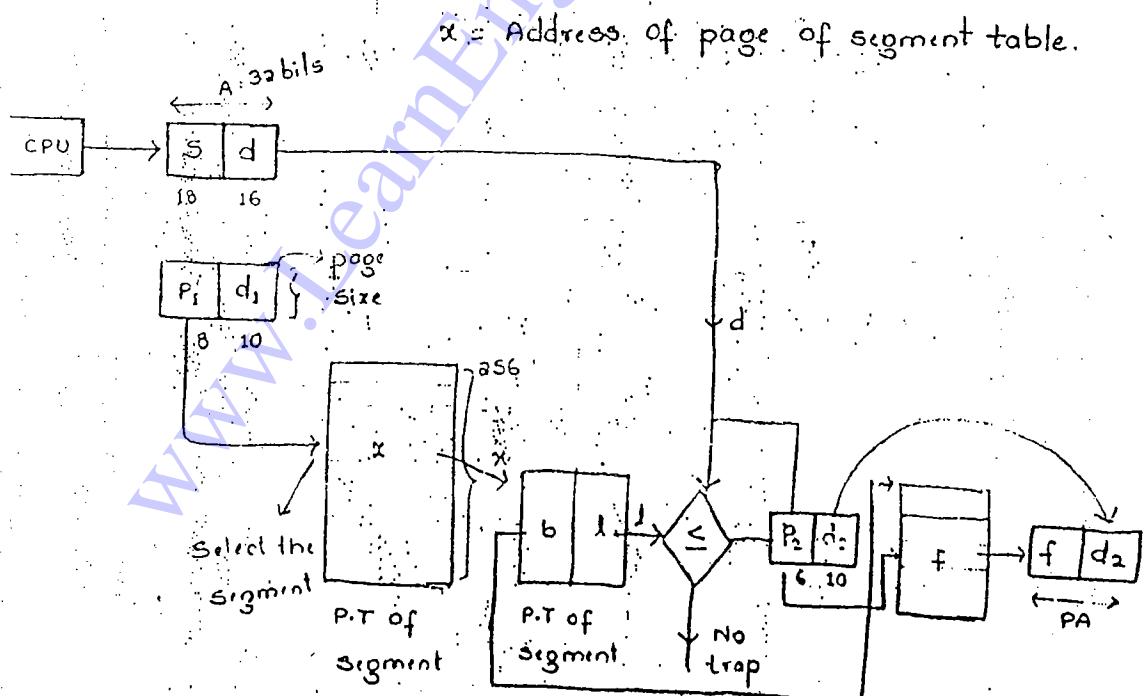
Max. size of seg. size  $\downarrow 2^{16} = 64\text{ KB}$   
offset

No. of segments =  $2^{18} = 262\text{ K}$

### Segmentation of Segment Six :-



### Segmentation of Segment tables



## Paging Vs Segmentation

|              | Internal<br>Fragmentation | External<br>Fragmentation |
|--------------|---------------------------|---------------------------|
| Paging       | ✓                         | ✗                         |
| Segmentation | ✗                         | ✓                         |

## Virtual Memory

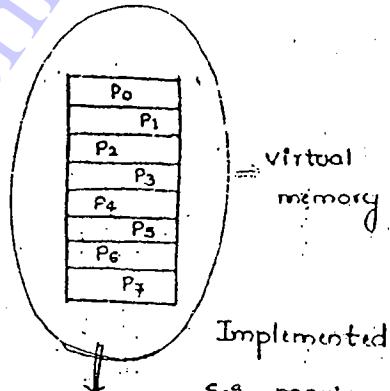
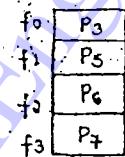
Virtual memory gives an illusion to the programmer that a huge amount of memory available for writing programs greater than the size of available physical memory.

Virtual memory implemented in secondary memory.

Main memory < Virtual memory < Secondary memory

### Demand paging :

Virtual memory is implemented with demand paging and demand segmentation.



Implemented in S.S. maximum virtual memory is limited by capacity of secondary storage

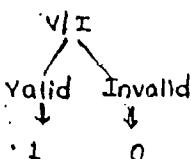
### Demand paging

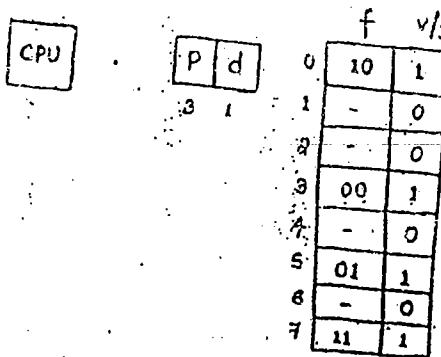
#### Pure Demand paging

Started with all the frames empty right from the first demand for page

#### Protected Demand paging.

prefetch load is now

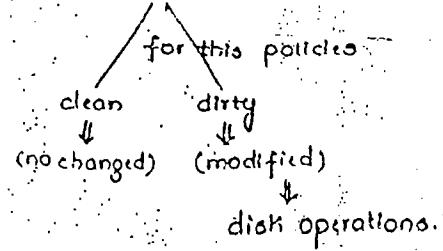




when page fault occurs, duty of virtual handlers:

- \* Process is blocked
- \* Virtual handler  $\rightarrow$  CPU
- \* V.M. handler  $\rightarrow$  Device manager
- \* Device manager  $\rightarrow$  Device controller
- \* Device controller  $\rightarrow$  Device buffer
- \* D.B.  $\rightarrow$  M.M. (DMA) transfer the page.
- \* (i) Empty free is available  $\Rightarrow$  (ii) disk operation is enough.
- (iii) No empty free  $\Rightarrow$  page replacement.

page fault service time  
 $\Rightarrow$  generally  $10^{-3}$  sec. for  
 memory accessing, it  
 takes  $\frac{10^{-6}}{10^{-9}}$  sec.



- \* Update page table
- \* Unblock the process

address

### Performance

Main memory Access Time (M.M.A.T) = 'm'

P.E.S.T = 's' ( $s \gg m$ )

Page fault rate = 'p'

Page hit ratio =  $1-p$

$$EAT = (1-p)m + p * s$$

$$E.A.T = (1-p)m + p(m+s)$$

Hit ratio is more  $\Rightarrow$  follow the units of main memory access times.  
 $\Rightarrow 1.9999 \mu s$

$$\begin{aligned} E.T.T &= \frac{1}{K} (i+j) + \left(1 - \frac{1}{K}\right) * i \\ &= \frac{i}{K} + \frac{j}{K} + i - \frac{i}{K} \end{aligned}$$

### Page replacement :-

#### frame Allocation policies:

total no. of frames = M

no. of processes = 'n'

demand of each processes for

frames =  $S_i$  (it depends on

total no. of pages of program)

total demand of process :

$$S = \sum_{i=1}^n S_i$$

(Allocation policies)  
 $S_i$ : Equal proportionate 50%

P<sub>1</sub> 10 10 5

P<sub>2</sub> 25 10 12 only given to

P<sub>3</sub> 3 → 10 1 justify 1 50% of resource

P<sub>4</sub> 12 10 6

P<sub>5</sub> 20 10 10

↓

It is applicable for all, which have equal demand.

If  $M = 50, n = 5$

$$\frac{M}{n} = \frac{50}{5} = 10 \Rightarrow \frac{n}{S} \times m \text{ (and policy)}$$

$$P_1 = \frac{10}{70} \times 50 =$$

$$P_2 = \frac{86}{70} \times 50 =$$

(b) Page Reference string :-

Set of successively unique pages referred in the given list of logical addresses :-

463, 182, 184, 195, 435, 969, 967, 834, 128, 534, 765, 784, 018, 684, 686  
page size = 1000 (consider).

463  $\Rightarrow$  It belongs to  
page address

$$\frac{463}{100} = 4 \text{ (every page contains 100 words)}$$

$$463 \% 100 = 63 \quad 63 \rightarrow \text{offset}$$

$\{4, 1, 1, 1, 4, 9, 9, 8, 1, 5, 7, 3, 0, 6, 6\}$  // If occurred successively, then consider  
ref. =  $\{4, 1, 4, 9, 8, 1, 5, 7, 0, 6\}$  // only once

String length = 10  
only  
given to  
5% of  
source  
phase

| each page | P <sub>0</sub> |
|-----------|----------------|
|           | 0              |
|           | 99             |
|           | 100            |
|           | 199            |

No. of unique pages referred in the given list of logical addresses :

$$n = \{0, 1, 4, 5, 6, 7, 8, 9\} = 8$$

Eg:-

reference string = 7, 0, 1, 2, 0, 3, 0, 4, 0, 3, 0, 3, 2, 1, 0, 0, 1, 7, 0, 1

length = 80, unique = {0, 1, 2, 3, 4, 7}

Frame size = 4

$$n = 6$$

page fault = 10

$$3F \rightarrow 16 \quad 4F \rightarrow 10$$

| 3 frames               | 4 frames            |
|------------------------|---------------------|
| 7 1 4 0 7<br>8 8 1 1 0 | 7 0<br>6 4 7<br>5 1 |
| 1 3 3 1 1              | ...                 |

- \* After loading page in main memory, how many times a page is referred so far.

: performance of LRU is most closest to optimal.

- ∴ O.S uses LRU (or LRU approximations).

#### LRU Approximations :-

\* works like LRU.

\* Approximate to the behaviour of LRU

|   |     |
|---|-----|
| F | 2   |
| 0 | → 2 |
| 1 |     |

⇒ first dropped in.

0 → 2 is replaced

1

They are not pure LRU  
but works like LRU

#### Reference bit (R) Algorithm:-

- \* Every page table contains a reference bit within it.

Page Table

| P | F | V/I | AT | R |
|---|---|-----|----|---|
| 0 | a | 1   | 3  | 1 |
| 1 | b | 1   | a  | 0 |
| 2 | c | 0   | -  | - |
| 3 | d | 1   | 0  | 1 |
| 4 | K | 1   | 1  | 0 |

R = 0 - page has not been referred so far during the present epoch.

    1 - page has been referred atleast once during the present epoch.

- \* "At the end of epoch, reference bits are cleared to zero".

Epoch ⇒ duration of time ⇒ Time divided into discrete intervals

(Time Quantum)

called epochs.

During current epoch, cobit pages are referred (or) not referred are indicated by reference bit (R).

|                | R |
|----------------|---|
| P <sub>1</sub> | 0 |
| P <sub>2</sub> | 0 |
| P <sub>K</sub> | 0 |

Initially

|                | R |
|----------------|---|
| P <sub>1</sub> | 1 |
| P <sub>J</sub> | 0 |
| P <sub>K</sub> | 1 |

page referred  
in next epoch.

End of old epoch &  
starting next epoch has  
initial value = 0

Start searching the page fault (victim)  
from initial, using ref. bit of LRU approx  
imation is Reference bit algorithm.

If all reference bit values = 1, then this algorithm fails.  
So, we introduce a new algorithm called Second chance.

### (2) Second chance / clock Algorithm :-

Criteria : Arrival time + reference bit  
 $A.T + R$

FIFO based algorithm.

Start the search of pages from the A.T. and if it is already referred,  
then give a second chance to it (i.e., value changed from 1 to 0).

| P | f | V/I | AT | R   |
|---|---|-----|----|-----|
| 0 | a | 1   | 3  | X 0 |
| 1 | b | 1   | 2  | X 0 |
| 2 | - | 0   | -  | -   |
| 3 | d | 1   | 0  | X 0 |
| 4 | K | 1   | 1  | X 0 |

(3) Enhanced Second chance: Avoids unnecessary page back.  
(not-recently used)

Criterio :  $(R) + (m)$  (modified bit)  
+  
(dirty bit)

modified bit  $\Rightarrow$  checks whether the contents of page are modified  
(or not).

0  $\rightarrow$   
1  $\rightarrow$  write back.

R = 0, M = 1 :-

| R . m |                           |
|-------|---------------------------|
| 0 . 0 | not R, not m              |
| x . 1 | not referred but modified |
| 1 . 0 | Referred but not modified |
| 1 . 1 | Referred, & modified      |

| R . m | R . m |
|-------|-------|
| x . 1 | 0 . 1 |

It is referred in old epoch and it is not referred in new epoch but modified.

when page fault occurs, consider the value of "00".

0  $\rightarrow$  LRU

we look for '01' combination next, since, it represents LRU

Then '10' is considered. & at last '11' is considered.

Ques.

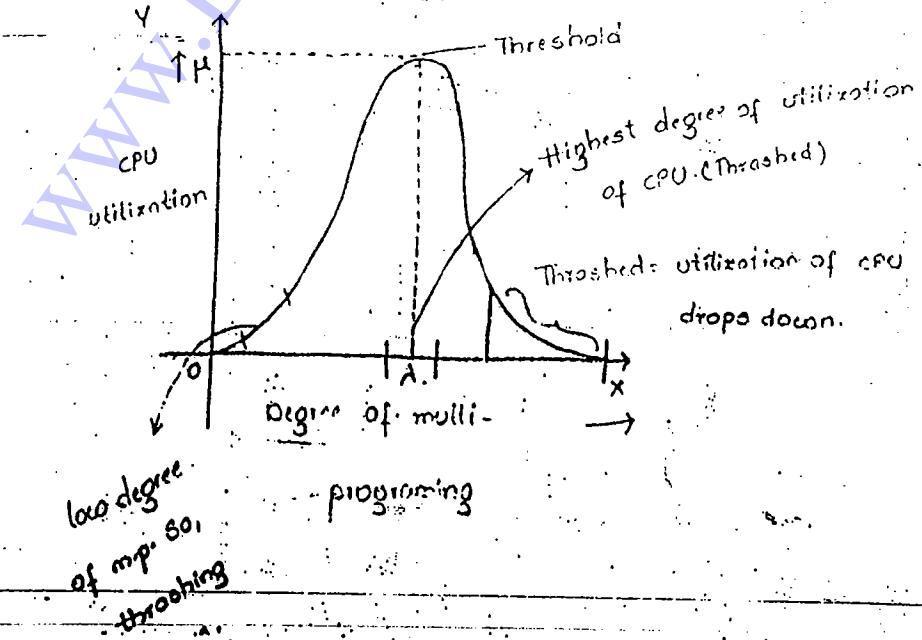
Q. Q.

Thrashing :- (High paging activity) (page fault rate).

- \* Excessive / High page fault rate is called Thrashing.
- \* It is also an undesirable state of the system like deadlock.

Reasons for Thrashing:-

| Primary                                                                            | Secondary                                                                                                                                   |
|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| (1) Lack of memory (frames)<br>i.e., if frame allocation to<br>a page get reduced. | (1) page replacement algorithm.<br><br>Drawback: fragmentation                                                                              |
| (2) High degree of multiprogramming                                                | (2) page size<br>small (1004B) $\Rightarrow$ 2 pages<br>(512 pages) $\Rightarrow$ more pages (then less page faults) when reduces thrashing |
|                                                                                    | (3) program structure.                                                                                                                      |



## Thrashing control strategies :

### I. Prevention & Avoidance

- \* controlling degree of m.p. in and around " $\mu$ "

page size:

Temporal (Time)  $\rightarrow$  Thrashing

Spatial (Space)  $\rightarrow$  Internal fragmentation

Program Structures:

int A(1...128, 1...128)

page size = 1280

i) for  $i \leftarrow 1$  to 128

    for  $j \leftarrow 1$  to 128

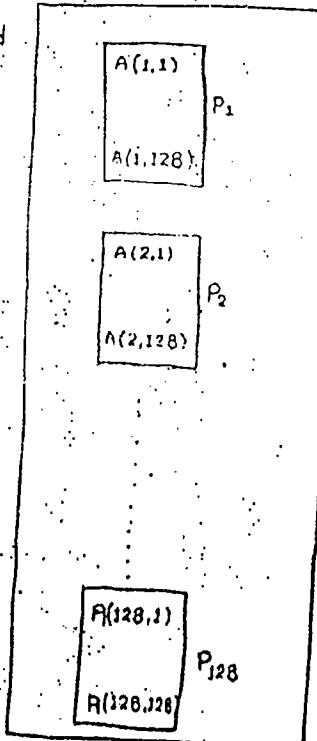
$A[i,j] = 5$

ii) for  $i \leftarrow 1$  to 128

    for  $j \leftarrow 1$  to 128

$A[i,j] = 6$

CPU Allocated



P.O.P

FIFO

(Q) If there are a set of processes (from 1 to 128) to be executed. But, If the CPU allocates only 127 processes to get executed. Then, how the CPU allocation processed for the above two conditions.

Sol:- Column  $\leftarrow A[i, i]$   
major order  $A[1, 1]$

$A(i, j) \rightarrow$  row major  
order

$P_{128}$

(1,1) not available  
(0,1) page fault

(1,2,1) go to memory  
page

$128 * 128$

each page size

is 128

working set model :-

\* Based on locality of reference.

main()

{

  f()

  {

    g()

    {

      h()

      {

        f()

        {

          g()

          {

            h()

            {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

              {

              f()

              {

              g()

              {

              h()

- \* Locality may be a function, block or class or module.
- \* Hold the pages of the locality at which the present function is performing.

Guess / Estimate : Estimating the ref. string in each function.

Reference String (pri):

| size                                                                                           |     |     |
|------------------------------------------------------------------------------------------------|-----|-----|
| 7, 6, 8, 9, 6, 8, 9, 7, 12, 16, 20, 22, 18, 20, 12, 23, 28, 29, 34, 38, 37, 33, 32, 39, 36, 34 |     |     |
| main()                                                                                         | f() | g() |

Working set window (WSW):

- \* Set of unique pages referred in the given list of reference string during the past ' $\Delta$ ' references.

↓  
Integer (Quiss)

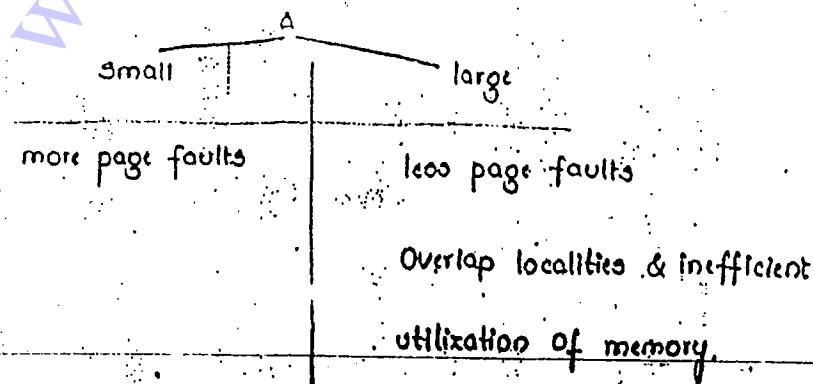
Let  $\Delta = 10$  (Guess)  $\Rightarrow$  no. of integers in  $g()$  = 10 (length)

$$\text{WSWS} = \{23, 27, 28, 29, 34, 38\} = 6.$$

Unique reference string from the

Set of  $g()$  function.

Success will depend on the value of  $\Delta$ .



\* If we guess the value of  $A$  to be small, then there is a chance occurring more page faults.

Eg: If  $A = 10$

$$\text{WSWS} = \{ 23, 28, 29, 34, 38, 44, 46, 48, 49, 58, 34 \}$$

Page faults = 6.

If  $A = 5$

$$\text{WSWS} = \{ 28, 23, 29, 38, 34 \}$$

Page faults = 5.

If  $A = 5$

$$\text{WSWS} = \{ 23, 28, 29, 34, 38 \}$$

Page faults = 5.

for the same set of reference string if  $A = 10$  (more no. of integers are consider, then Page faults = 6. other than if  $A = 5$ ; it have page faults = 10).

\* Let  $P_i = 1, 2, \dots, n$

$$1000w_5 | i \quad \text{where } i = 1, 2, \dots, n$$

$$\text{Total demand} (s) = \sum_{i=1}^n (w_5 w_3)_i$$

Total frames available = "m"

\* If  $m \geq s$  (no thrashing, system is perfectly balanced).

\* If  $m > s$  (no thrashing).

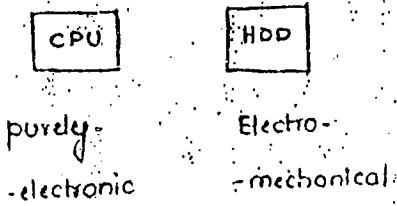
(Scope of increasing degree of multiprogramming).

\* If  $m < s$  (Thrashing)

1000 10,000

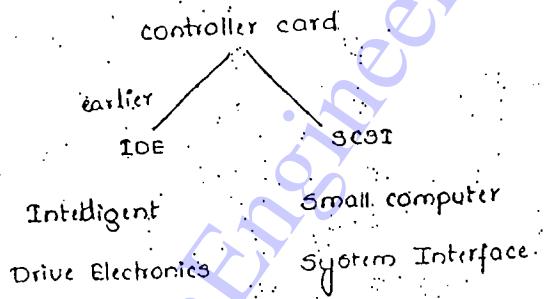
## File System & Device management

### Interface :



Every secondary / Tertiary media  
must have their own file system.

- (1) We need a controller interface as controller card / chip.  $\Rightarrow$  Hardware requirement



- (2) Software requirement :

\* Device driver.

- (3) Device Independent software :

\* It is called as file system. (Third party device drivers).



## Files & Directories

file :- Collection of logically related set of records of an entity.

\* It is considered for data structure.

\* Data Structure :-

(1) Definition.

(2) Representation / Implementation:

\* Flat structure

\* Record structure

\* Tree structure

} file structures.

(3) Operations:

\* Create a file

\* Open a file

\* Read / write / append / seek / modify / Truncate

\* Close a file

\* Delete a file

(4) Attributes:

\* name, extension of file

\* Type of file

\* Size of file

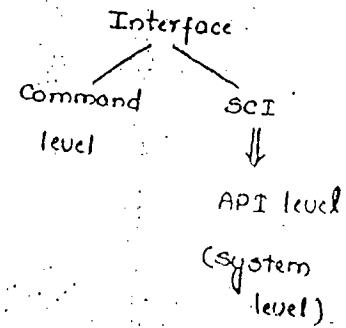
\* owner

\* permission

\* mode of access (Sequence / random)

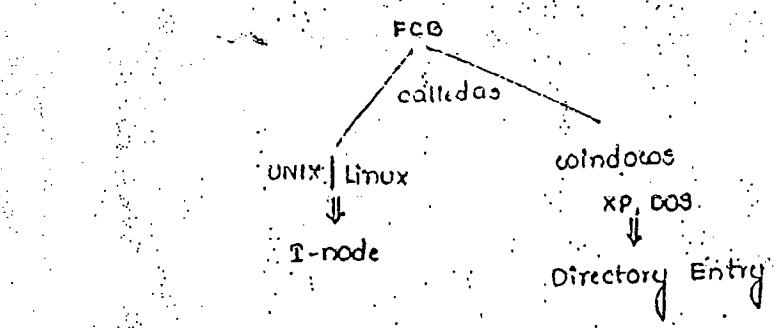
\* Time & Date stamps

\* Link count.



Stored in  
file control block (FCB)

- \* Attributes of a file are stored in file control block (FCB).

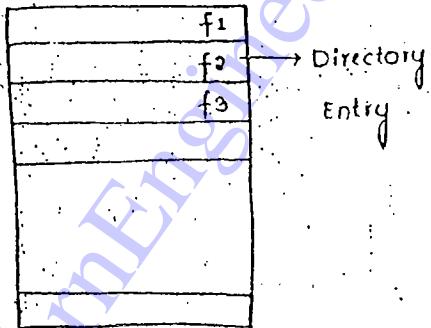


### Directory

- \* It contains information about files i.e., metadata of files.

Abstract view of  
directory (metadata of files)

1 Directory Entry  
per file.



### Organisation of Directory Structure

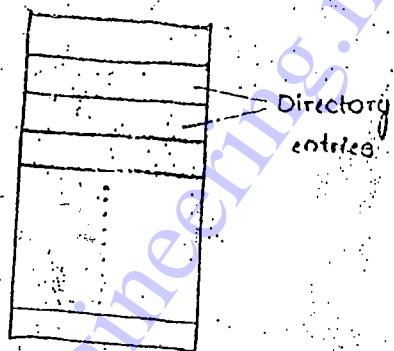
- \* Single level directory
- \* Two level directory
- \* multi level / Tree Structures
- \* Acyclic Graph.

31/07/2010

saturday

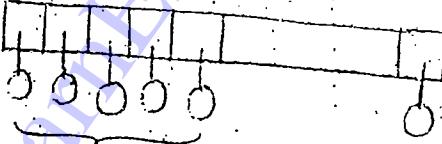
### Directory Structure :-

Abstract view of Directory



### (a) Single-level Directory structure :-

Directory



- \* No sub-directories are present. (one entry for each file)
- \* All the files are managed by one directory.

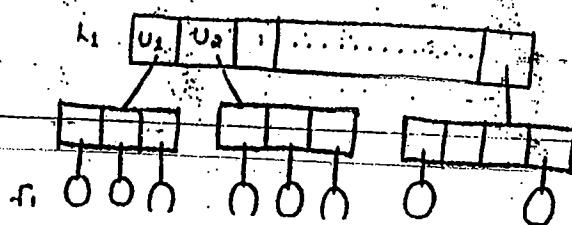
### Advantages :-

- \* Simplicity
- \* Searching
- \* Name conflicts

### Disadvantages :-

- \* Only one directory, no sub-directories.
- \* Two different users, simultaneously can have same filename.

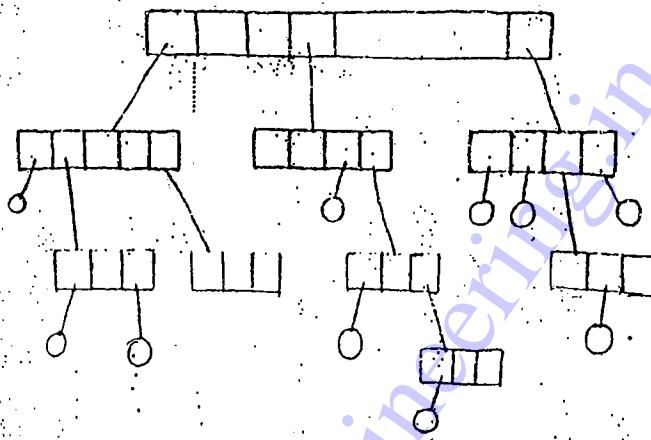
### (b) Two-level Directory structure :-



### Advantages :-

- \* Creating different files with same name. (flexibility)

### (3) Tree-structured directory :-



### Drawback :-

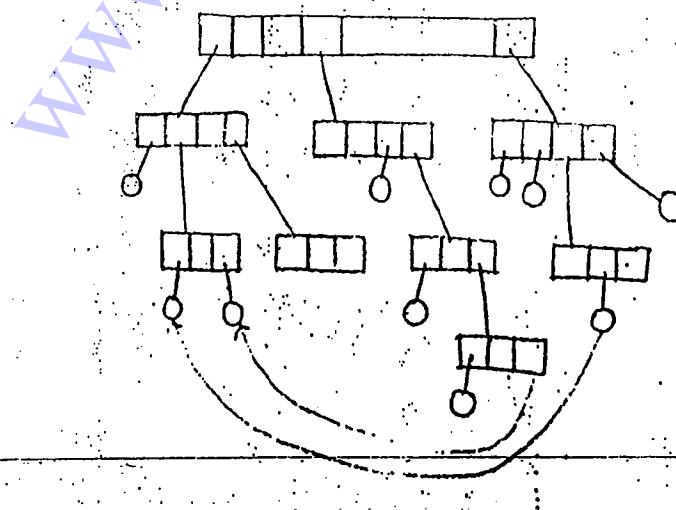
#### \* File sharing :-

- \* Sharing by duplication has drawbacks of :-

- \* Inconsistency

- \* wastage of space

To overcome duplication in file sharing, we have the concept of file sharing with links called "Directed Acyclic Graph".

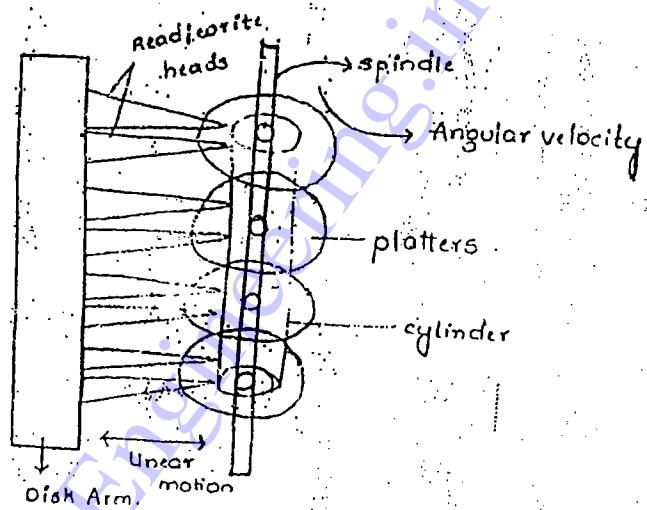


### Device characteristics:

- \* Major component in secondary storage is disk.

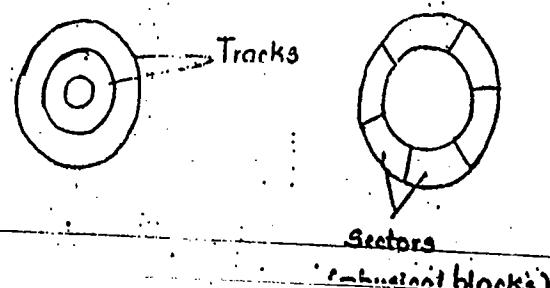
### Physical and logical structure of disk :-

#### Physical Structure / geometry of Hard disk:-

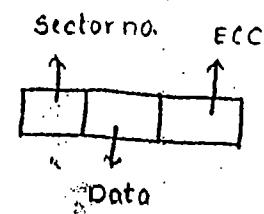


- \* Hard disk consists of set of platters, with a spindle.
- \* Each surface of platters is associated with Read/ write heads.
- \* Two types of motions are supported:
  - \* Linear velocity (move forward & backward) with disk armature.
  - \* Angular velocity with disk spindle.
- \* Tracks are divided into sectors.

### Sectors :-



(-logical blocks)



Sectors can be represented by three fields :-

- \* Sector no.
- \* Data
- \* Error correction code (ECC).

Error Correction code (ECC) :

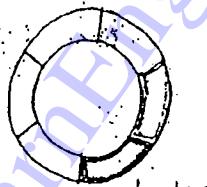
- \* To detect the deadlock. By reformatting the bad sector, some sectors can be made good sector.

Cylinder :

A group of some track number.

Cluster :

A group of one or more adjacent sectors is a cluster. A cluster is one of the unit of I/O transfer.



I/O Transfer :

The amount of time taken to a sector to make a move from present area to desired area is called seek time.

seek time:

time taken to

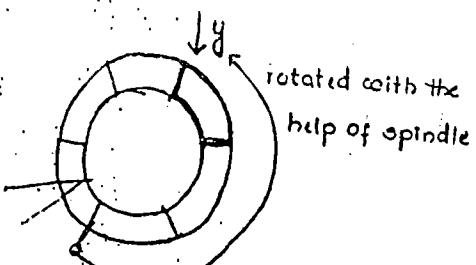
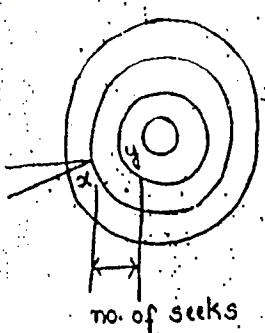
travel from 'x'

to 'y'

(or)

linear motion

from x to y



$$\text{time} = \frac{R}{2}(\text{average})$$

Seek time + (Rotational) latency time

$$\frac{R}{a} \text{ (on an average)}$$

where:  $R = \text{Time for one rotation}$ .

Transfer time (Transmission Time) :-

\* The transfer time depends on two factors:

\* Track size ( $s$ ) bytes

\* Rotation rate (RPM)

Transfer time:

\* Read the data for sector

Sector size = ' $x$ ' bytes ( $x < s$ ) i.e. [sector size  $\ll$  track size]

Rotation rate = ' $r$ '.

Time for one rotation  $\approx R$

$$R = \frac{60}{r} \text{ s}$$

$$3600 r = 60 \text{ s}$$

$$1r = ?$$

$$\text{If } \frac{60}{r} \text{ s} \Rightarrow s \text{ bytes}$$

$$\Rightarrow \frac{60}{3600} = \frac{1}{60} \Rightarrow R = \frac{60}{r} \text{ seconds}$$

$$? \Rightarrow x \text{ bytes (read)}$$

$$\Rightarrow \frac{x * 60}{r * s} \times \frac{\text{sec}}{\text{s}} = \frac{x * 60}{r} \text{ bytes}$$

DISK I/O Time :-

= (Seek Time) + (Latency Time) + (Transfer time)

$$= S.T + \frac{R}{a} + \frac{x * 60}{r * s} (\text{sec})$$

$$\frac{60}{r} \rightarrow s$$

$$? \rightarrow x$$

$$x * 60$$

$$r * s$$

Effective Data Transfer Rate (DTR) (Bytes/sec) = no. of bytes in 1 sec.

Assume Track size = 's' bytes

$$\text{RPM} = \frac{1}{T}$$

$$\text{Rotation time} = \frac{60}{T} \text{ sec}$$

$$\frac{60 \text{ sec}}{T} = s \text{ Bytes}$$

$$1 \text{ sec} = ?$$

$$\boxed{\text{D.T.R} = \frac{T * s}{60} \text{ bytes/sec}}$$

Eg:- consider a hypothetical disk with :-

16 platters

↳ 2 surfaces

↳ 2 tracks

↳ 512 sectors

↳ 2KB

Disk seek time = 30 ms

RPM = 4000

Then calculate (i) capacity (ii) I/O time (iii) DTR

Sol:-

$$(i) \text{Total no. of sectors} = 2K \times 512$$

$$\text{Capacity of one surface} = 2K \times 512 \times 2KB$$

$$= 2^4 \times 2^{11} \times 2^9 \times 2^{11}$$

$$\text{Capacity of two surfaces} = 2^5 \times 2^{11} \times 2^9 \times 2^{11} \times 2^{36}$$

$$= 64 \text{ GB}$$

$$(ii) \text{I/O time} = \underbrace{\text{seektime} + \text{latency time} + \text{Transfer time}}_{\frac{R}{a} + \frac{\alpha * 60}{T * s} (\text{s})}$$

$$R = \frac{60}{4000} \text{ s} \Rightarrow \frac{R}{2} (\text{latency time}) = \frac{60}{8000} \text{ s}$$

Transfer time :-  $\frac{60}{4000} \text{ s} = 1 \text{ MB}$   
 $? = 2 \text{ KB}$

$$\Rightarrow \frac{60 \times 2 \text{ KB}}{4000 \times 1 \text{ MB}} \text{ s} = x$$

I/O time =  $\underbrace{S \cdot T + L \cdot T + T \cdot T}_{\downarrow} + \frac{R}{2} + \frac{x \times 60}{r \times s} \text{ (s)}$

$$= 30 \text{ ms} + \frac{60 \times x}{8000} \text{ s} + x$$

BP  
BRPD

(ii) OTR :-

$$\frac{60}{4000} \text{ s} = 1 \text{ MB}$$

$15 = ?$

$$\frac{4000 \times 1}{60} \text{ MB/s}$$

Logical structure of Disk (formatting process) :-

Popular file systems :-

\* NTFS }

\* Solaris }

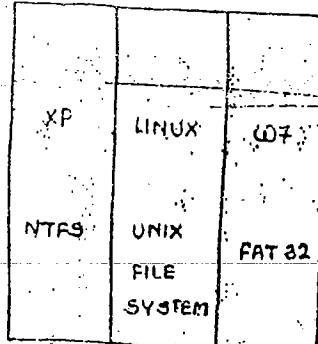
\* FAT 32 }

\* UFS }

\* ZFS }

} popular.

**fdisk** → create, delete, & edit the partitions



partitions

(volumes)

(c; d; e; )

Partitions

Primary

\* Bootable

(Stores O.S onto it)

&

userdata

Secondary

(Extended)

\* Non-bootable

↳

only userdata  
get stored.

In the partitioning of a system, there must be atleast 1 primary partition, and the rest may (or) may not be of secondary.

Computer System, which supports multiple O.S to boot is called multi-boot computer.

#### Master Boot Record

(MBR)



file format system:

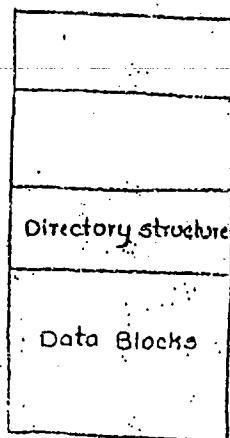
we can select any file format with different O.S's.

(Booting the selected O.S)

table (It represents the complete picture of partition)

(only one entry for the partition).

## Partition Structures :-



**Boot Control block (BCB)**

**Partition control block  
(PCB)**

Blocks containing  
data of file are  
stored.

(Eg.: user data, file data)

BCB block is empty  $\Rightarrow$  If there is non-bootable partition.

BCB can be represented in different terms :-

- \* Boot Block (In UNIX)

- \* ~~Master~~ Partition Boot Sector (In NTFS) (It contains boot of that sector).

PCB gives information about other blocks.

PCB can be represented in different terms :-

- \* Superblock (In UNIX) (gives information about others)

- \* Master file Table (M.F.T.) (In NTFS)

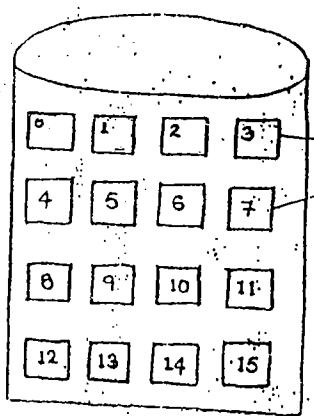
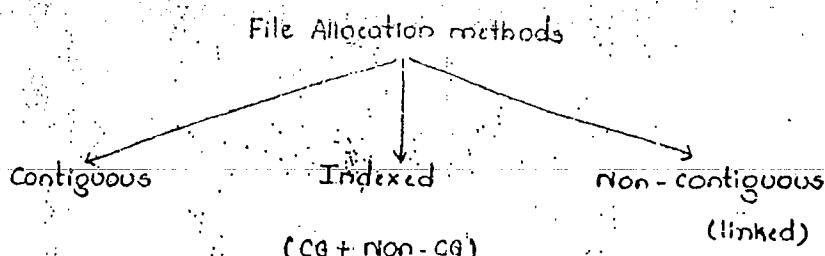
Disk space is being organised in blocks.

Block  $\Rightarrow$  unit of Allocation.

Allocating the disk space methods  $\Rightarrow$  file Allocation methods.

DISK File Allocation methods (or) Disk Space Allocation strategies :-

(1)



Disk blocks (Associated with two factors/parameters)

Disk Block Address (DBA)  
(n-bits)

Disk Block Size (bytes)  
(DBS)

Consider DBA = 16 bits

DBS = 1 KB

Then, maximum file size (in bytes) = 64 MB

depends on/limited by

disk size

$$\text{Disk Size} = 2^{16} \times 1 \text{ KB}$$

$$= 64 \text{ K} \times 1 \text{ KB}$$

$$= 64 \text{ MB}$$

### (1) Contiguous Allocation :-

first block of disk used by file  
directory entry

| filename | starting block | file size (no. of blocks) |                                 |
|----------|----------------|---------------------------|---------------------------------|
| test.c   | 5              | 4                         | (5, 6, 7, 8 blocks)             |
| temp     | 10             | 6                         | (10, 11, 12, 13, 14, 15 blocks) |

At least 'm' contiguous blocks required.

### Performance :-

### (1) Fragmentation :-

Internal      External

(not contiguous, but  
bifurcated with memory)

If last block files are  
not fully utilised, then  
it goes waste, which  
represents internal frag-  
mentation.

### (2) Increasing file size :-

May / May not be possible

Inflexible

### (3) Type of Access :-

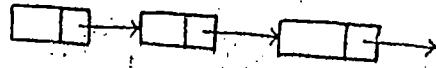
Sequential      Random

Both are  
supported

Sequential (Array)



Random (linked list)



Since, it supports random access, it is faster.

(2) Non-Contiguous / Linked Allocation :

| Filename | Starting DBA | Ending DBA |           |
|----------|--------------|------------|-----------|
| test.c   | 4            | 3          | (4,9,6,3) |
| temp     | 5            | 2          | (5,1,2)   |

Performance :-

(1) Fragmentation :

Internal

External (Any block can be utilised)



(2) Increasing file size :-

File size increment is possible, as long as free blocks are available.

(3) Type Of Access

Sequential access only possible. No random access. So, it is slow.

Drawbacks :

\* Some disk space is consumed for storing pointers; that addresses next instruction block.

\* vulnerability of links :

↓      ↓  
expose danger      breakage of links

↓  
Then, file gets truncated.

### Reason for storing ending DBA :-

- \* we store ending DBA, (along with starting DBA), because for the following :-

- \* Creating a new pointer done fastly.

- i.e., To extend a new file along with the existing, it can be made faster.

- \* Checking file system consistency. (scandisk).

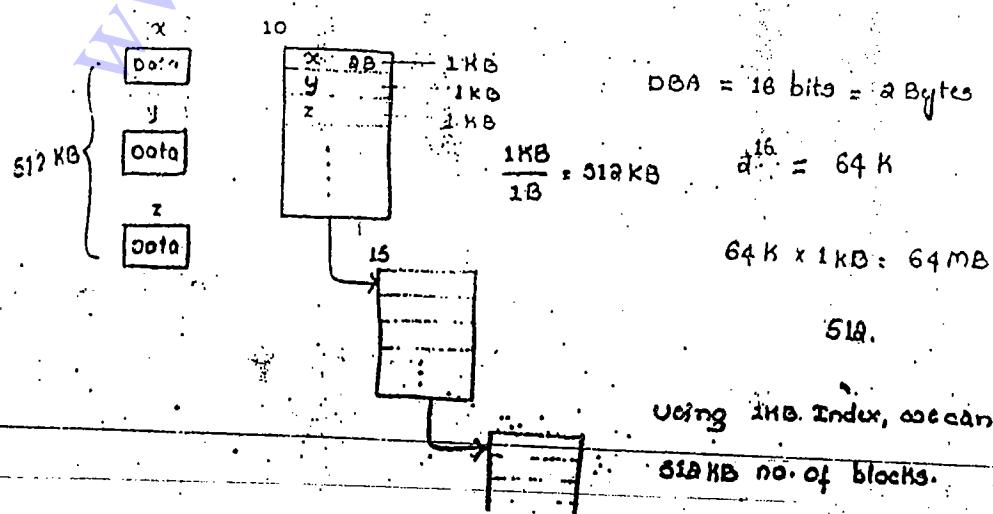
- i.e., checking whether file is fully accessed (con. note)

### (3) Indexed Allocation ( Contiguous & Non-Contiguous ) :-

| Filename | I-Block |
|----------|---------|
| test.c   | 10      |
| temp.    | 15      |

- \* A file can be allocated in both contiguous and non-contiguous order.

- \* Eg.: - DBA = 16 bits , DBB = 1 KB



04/06/2010

## UNIX - I-node structure

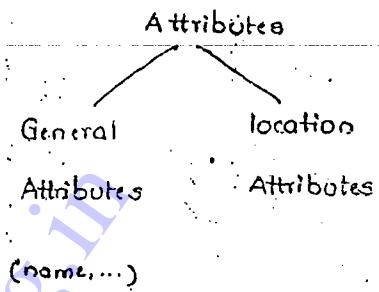
Wednesday

Directory entry consists of two fields :

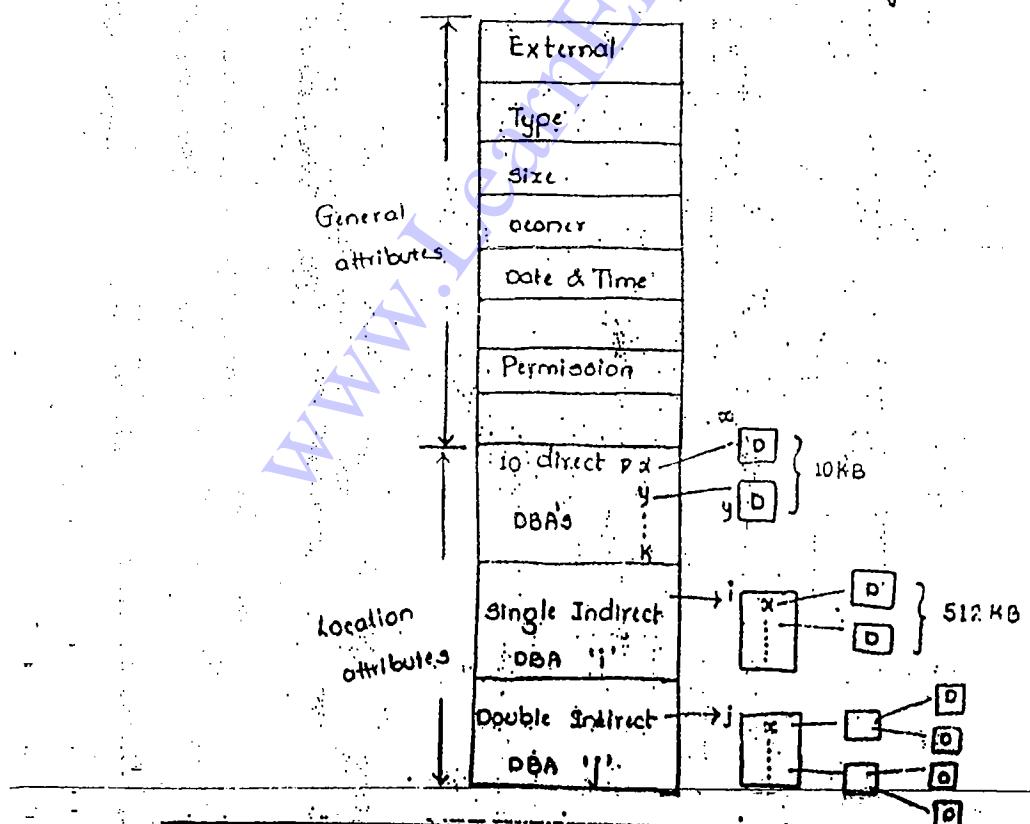
\* filename

\* I-node (contains Attributes)

| filename | I-node(43) |
|----------|------------|
| testc    | 33         |
| temp     | 40         |



I-node(43) → block of memory



\* Maximum file size is limited by the maximum disk size = 64 KB

### Single Indirect DBA :-

$$DBA = 16 \text{ bits}, DBS = 1 KB.$$

$$2^{16} = 64K.$$

$$64K * 1KB = 64MB$$

$$512 + 10 = 522 KB = 0.522 MB.$$

### Double Indirect DBA :-

$$512 * 512 KB = 2^9 * 2^9 * 2^{10}$$

$$= 2^{28} = 256 MB$$

$$\text{Total Size} = 256.522 MB \text{ (logical)} \Rightarrow \text{not considered}$$

because max. file size is only 64

### DOS directory Implementation :-

\* DOS directory implementation is complex than that of UNIX.

\* All attributes are embedded in the directory entry itself. There is no I-node in DOS implementation.

### General Attributes

| FN | EXT | Type | Size | Date & Time | ..... | first DBA(H) |
|----|-----|------|------|-------------|-------|--------------|
|----|-----|------|------|-------------|-------|--------------|

test.c ..... Directory entry

### local Attributes

$\alpha \rightarrow$  first block address  
of data.

\* For the remaining data addresses to represent, we use file Allocation Table (FAT).

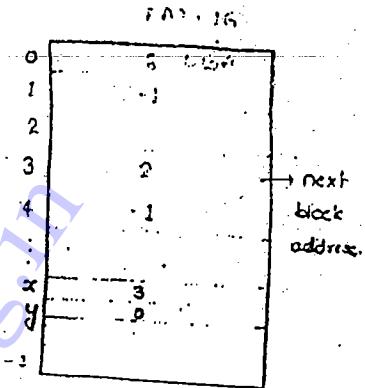
No. of Entries in FAT = No. of blocks on disk.

n blocks = 0 to

(n-1)

- \* Only the first block address of data is being indicated as a field, and the remaining data addresses are stored in FAT table, and referred whenever required.

- \* To know the (blocks) address of data, move to 'x' address on FAT. It represents the next data address. If there is last address, then it is represented by "-1" (i.e., it is last data)
- \* FAT entry represents  $\Rightarrow$  address of disk block.



Tabular linked allocation.

### DISK FREE SPACE MANAGEMENT

Eg:- consider disk = 20 MB

$$OBA = 16 \text{ bits} \quad OBS = 1 \text{ KB}$$

$$\text{Total no. of blocks} = \frac{20 \text{ MB}}{1 \text{ KB}} = 20K \text{ blocks}$$

Initial, 20K blocks are free.

Approaches to keep track of these 20K blocks :-

(1) Free linked list :-



If any file requires the block, it deletes the particular size (e.g. 10) and allocates it to the file.

Q. Q.

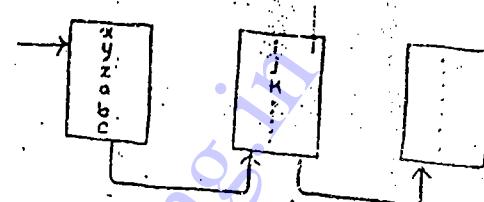
### (a) Free list :

- \* The addresses of the files are stored in linked lists.

$$1 \text{ block} \Rightarrow 512$$

$$? \Rightarrow 80K$$

$$\frac{80K}{512} = 40$$



- \* If the disk is almost free, it requires 40 blocks.

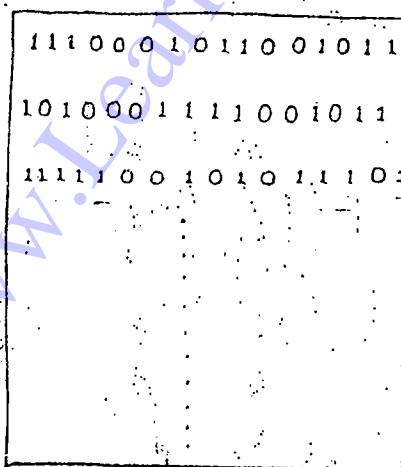
- \* If the disk is almost full, it requires 1 block.

### (b) Bit-map method :

Associate a binary bit with each block

0 - block is free

1 - block is in use.



Size of bit map for the

blocks =  $80K$  bits.

1 block can store =  $8K$  bits

? =  $80K$

$$\frac{80K}{8} = 10K$$

$\approx 3$  blocks.

It requires searching time (more) for a file.

So it is slower than free list.

Block size  $\leq 2^9$ .

S - DBS

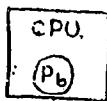
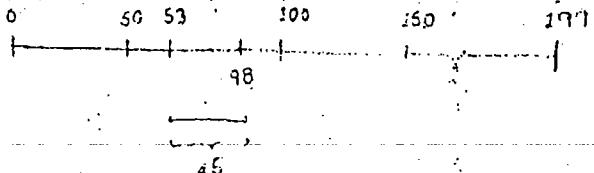
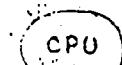
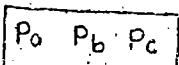
B - Blocks

D - DBA

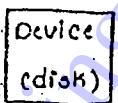
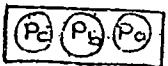
P - Freeblocks

### Disk Scheduling :

Ready Queue



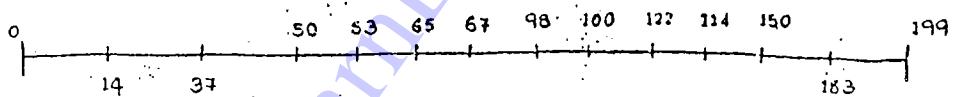
Process Queue



- Disk scheduling is used to represent which process is serviced next, when a process completes its operations.

#### (1) FCFS :

Requests : 98, 183, 37, 122, 14, 124, 65, 67



98 - read

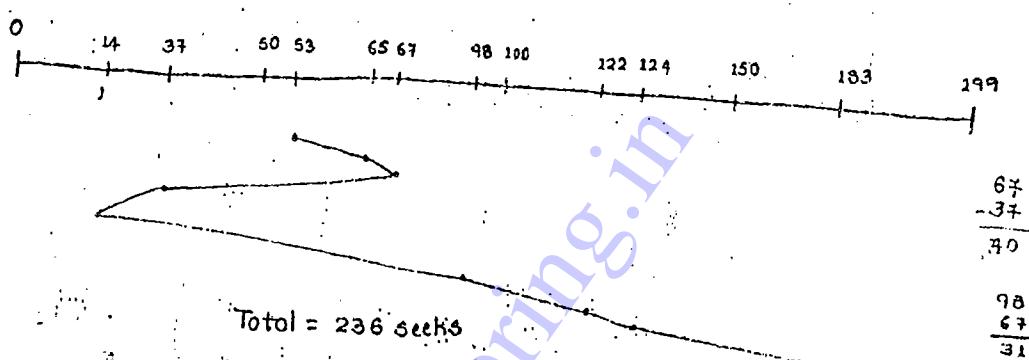
from track  
'9'

Total = 640 units

To reduce no. of seeks, we have SSTF

(a) Shortest Seek Time first (SSTF) / Nearest Track Next (NTN) :-

(Greedy method)

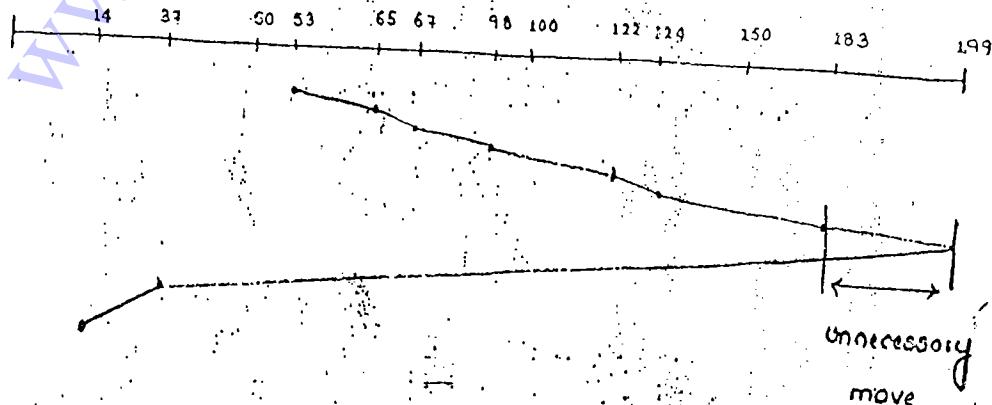


(b) SCAN / Elevator algorithm :-

- \* Read-write Head  $\Rightarrow$  represents directional scan.
- \* Scanning through the request encountered in certain direction.

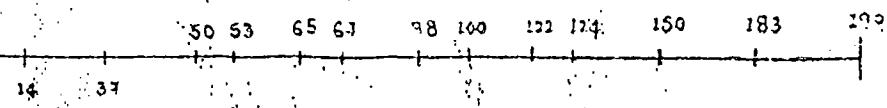
Drawback:

- \* It moves to the last track unnecessarily.
- \* Unnecessary causing starvation, by pending the other track in other direction.



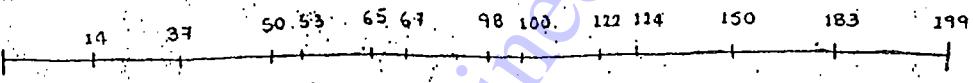
(4) Look :

Lookup to last request pending, and take reverse turn.



(5) Circular Scan (C-Scan):

Eg:- Disk

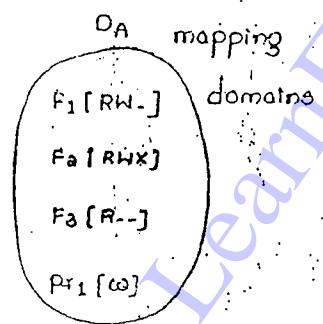


### Security vs. Protection

#### Security :-

- \* Making system secure from external threats (virus, worms, trojans etc)
- \* Security / protection has difference of authorized & unauthorized users.
- \* Protection mechanism is implemented using "domain() method"
- \* Protection Domain (Objects, Rights)

refers either user / process (with rig. permissions)

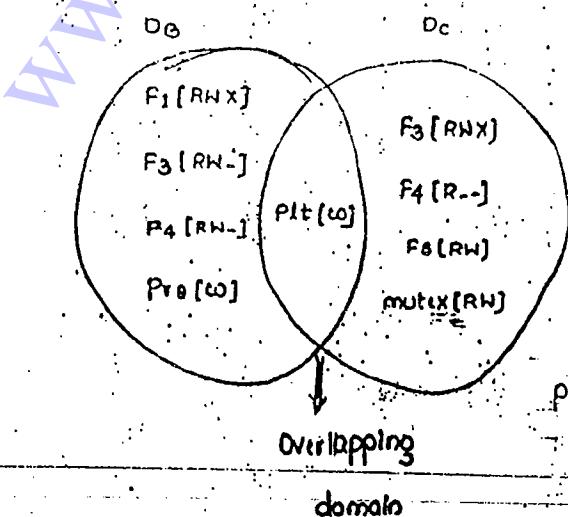


#### Protection :-

Internal threats  
(systems level)

#### Security :

External threats



### (1) Protection Domain matrix:

|       | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $P_{r_1}$ | $P_{r_2}$ | $P_{r_3}$ | $P_{l_1}$ mutex |
|-------|-------|-------|-------|-------|-------|-----------|-----------|-----------|-----------------|
| $D_A$ | RW-   | RWX   | R--   | -     | -     | W         | -         | -         | -               |
| $D_B$ | RW    |       | RW    | RW    |       | .W        |           |           |                 |
| $D_C$ |       |       | RW    | R     | RW    |           |           |           | RW              |

$n \times m$

$n \rightarrow$  no. of domains

$m \rightarrow$  no. of objects

entry  $\rightarrow$  permission.

They support direct access. (so, they are faster)

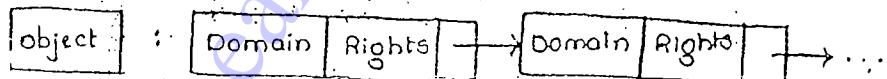
Drawback: space wastage (called sparse matrix).

$\downarrow$   
 $\downarrow$   
most are null entries.

To overcome

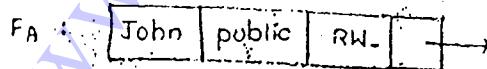
¶

### (2) Access Control List (A.C.L) mechanism:



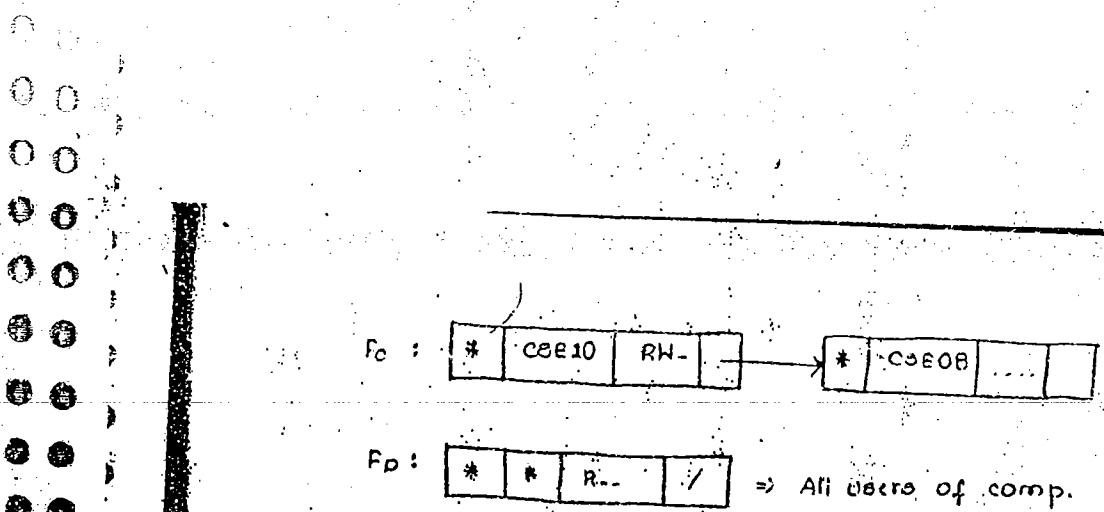
linked lists headed by objects

Used in : UNIX / LINUX



(domain) =

file used by all cae people.



### (3) Capability List (c-list):

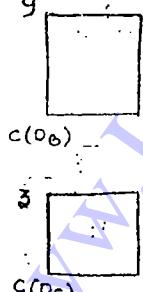
| Type    | Rights | Name/Address   | → stored in fixed place |
|---------|--------|----------------|-------------------------|
| FILE    | RW-    | F <sub>1</sub> |                         |
| FILE    | RWX    | F <sub>2</sub> |                         |
| FILE    | R--    | F <sub>3</sub> |                         |
| PRINTER | W-     | Pri            |                         |

Capability (DA)

Windows

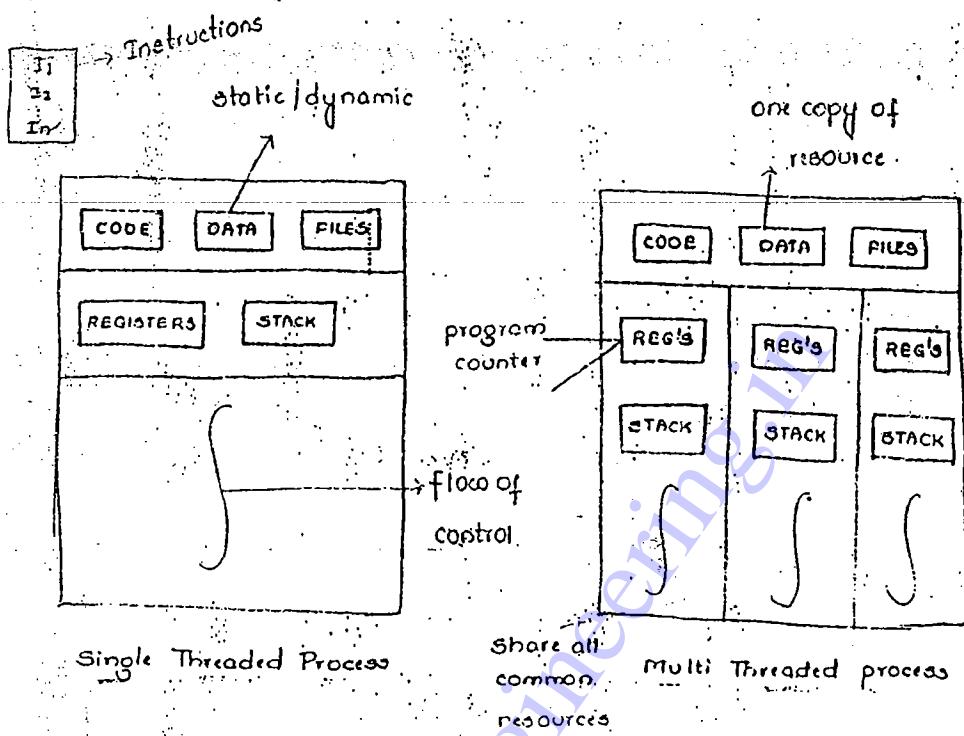
protection mgmt follows

c-list



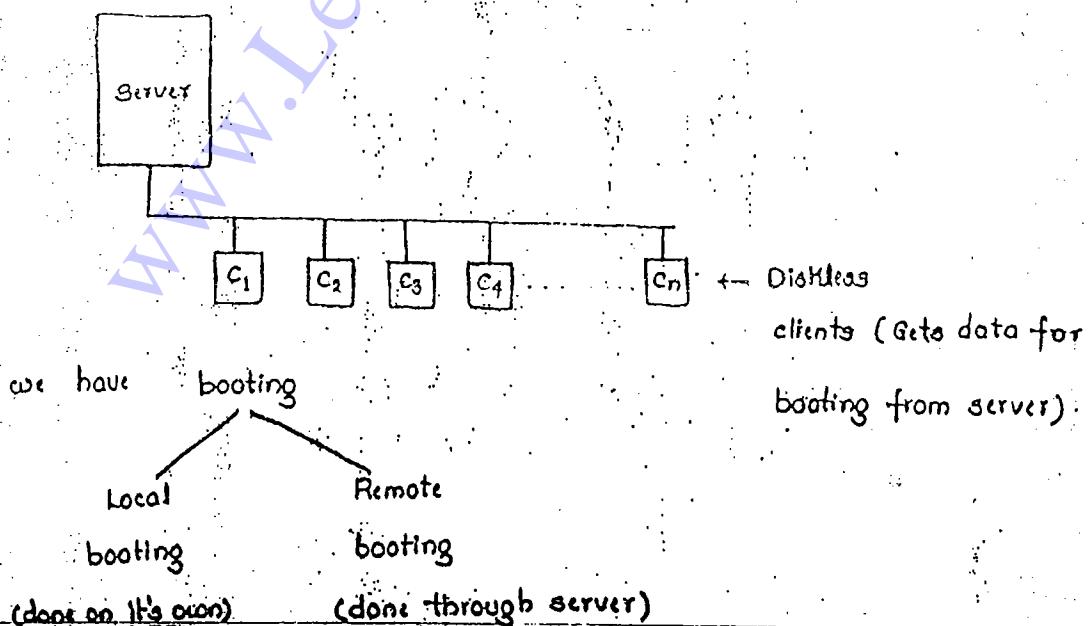
| Domain | Address (capability) |
|--------|----------------------|
| Da     | x                    |
| Dc     | y                    |
| Oc     | z                    |

## Threads & Multithreading

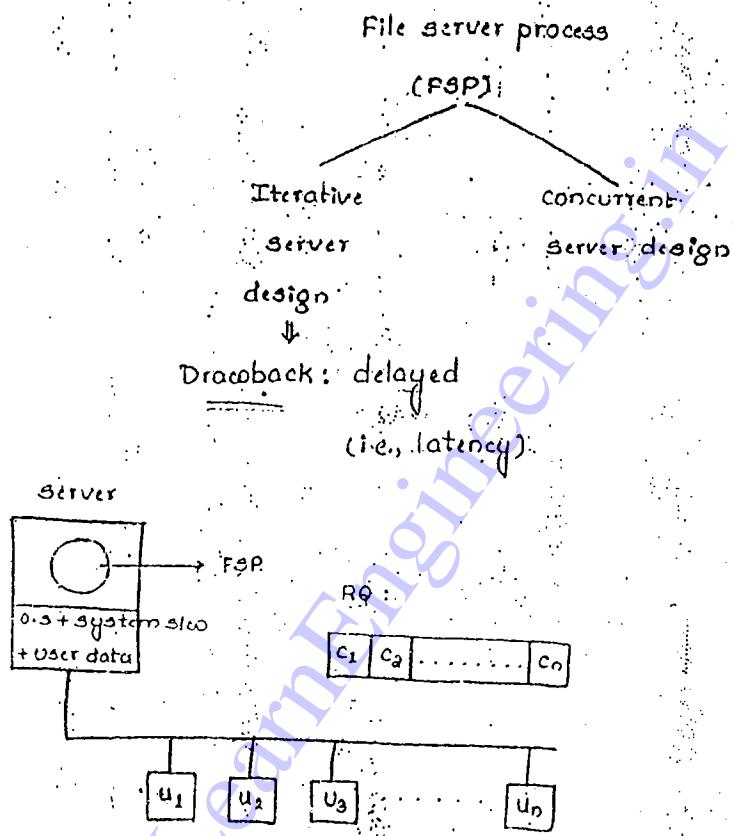


Thread - Light weight process.

LAN ( File Server process )



- \* If server is alive, it transfers the Kernel code to the client to get booted, which is called as "remote booting".



\* Concurrency is achieved through multi-process mechanism (using fork()).

\* The same code of server (parent) get copied in the childs (clients), i.e., duplication of code is done at all the clients.

Drawbacks :

\* wastage of resources :

\* Since code is duplicated (i.e., 'K' copies of code, data, files and access

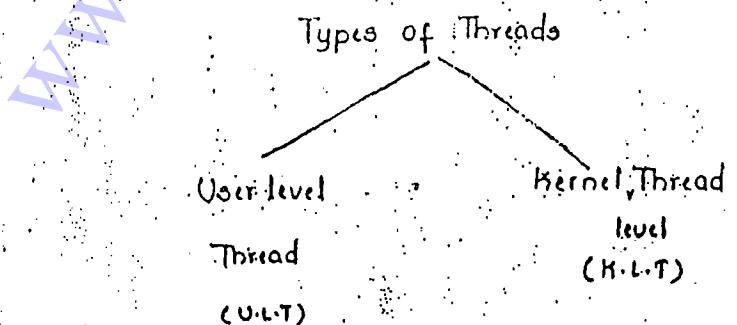
ed by all the clients. Same functionality is processed by all clients,

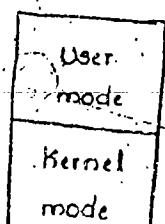
instead of single copy and get shared by them).

In non-sharing clients, every independent process must have program counter, General purpose registers and stack for each, whereas as a single copy of these can be accessed by all the clients in sharing.

Benefits :

- \* Resource sharing
- \* cost effective : Economical
- \* Improved performance.
- \* Achieve parallelism (multi-CPU).
- \*  $|TCB| < |PCB|$
- \* As processes have PCB's, Threads also have TCB's (Id, state, priority)
- \* Attributes that can't be shared are stored in TCB (i.e., Id) and which are shared are stored in PCB.
- \* Context-switch becomes fast and less context-switch overhead. So, there is improved performance.
- \* Since, it satisfies (5) & (6)th conditions, a thread is called as "Light weight process".





\* Threads, which have packages, libraries.

Eg: Java threads (It won't require o.s support)

\* O.S visualized it as a process.

\* It is transparent to the O.S, that it is only viewed as a process, but not as a thread.

\* Java Virtual machine (JVM) allocates the time to the threads.

\* User level Thread switching (U.L.T.S) is faster than Kernel level Thread switching (K.L.T.S), because for every processing, there is no need to go to Kernel always, everything is managed at user level.

#### Drawback :-

\* When a process requires I/O, the total process gets blocked instead of a single thread, because of transparency.

\* To overcome this drawback, we move to kernel level threads.

\* Threading management can be handled by Kernel level.

\* If one thread requires I/O, then that particular process only gets blocker but not the others.

\* Context switching also done through Kernel level, so, it is slightly slow compare to user level thread.

### Pthread :

- \* Portable operating system interface for UNIX  $\Rightarrow$  POSIX

### Monitors

#### Synchronization

##### mechanisms

with

Busy waiting  
(spinlock)

without

Busy waiting  
(blocking)

Spinlock  $\Rightarrow$  busy waiting.

{ livelock  $\Rightarrow$  states of process are always ready (or) running.

Deadlock  $\Rightarrow$  states of process are always blocked

### monitors :

It is a collection of procedures, variables and data structures, that are all group together in a special kind of modular package.

- Procedures running outside the monitor, cannot access the monitor's internal variables & data structures. However, they can activate monitor's internal procedures.
- Monitors have an important property that "only one process can be active at any time".

### Producer-consumer problem:

```
monitor producer.consumer;
begin
 integer count = 0;
 condition full, empty;
 Procedure Enter
 begin
 if (count = n) . . . wait (full);
 Buffer [in] = itemp;
 in = (in + 1) mod n;
 count := count + 1;
 if (count = 1) . . . signal (empty); \Rightarrow // wakeup consumer
 end;
 Procedure Remove
 begin
 if (count = 0) . . . wait (empty);
 itemc := Buffer [out];
 out = (out + 1) mod n;
 count := count - 1;
 if (count = n - 1) . . . signal (full);
 end;
 Procedure producer
 begin
 cohile (true)
 begin
 Produce-item (itemp);
 Producer.consumer:enter;
 end;
 end;
 Procedure consumer
 begin
 cohile (true)
 begin
 Producer.consumer:remove;
 process itemc;
 end;
 end;
end;
```

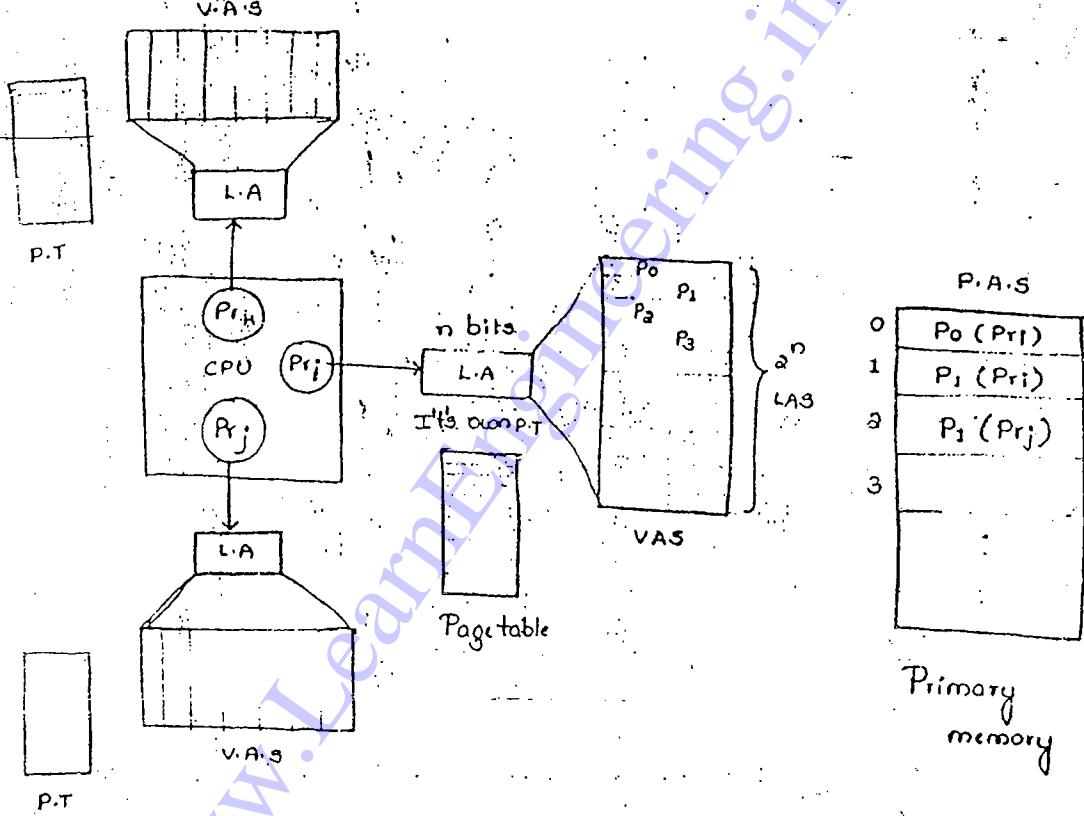
$\text{mutex} = 1 \Rightarrow$  no process is inside

$= 0 \Rightarrow$  some process is accessing

Initially  $\text{mutex} = 1$ ;

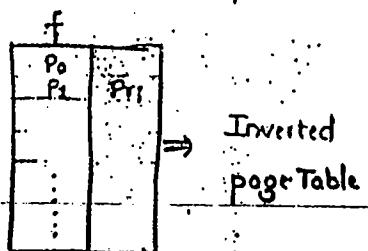
then `DOWN(mutex);`

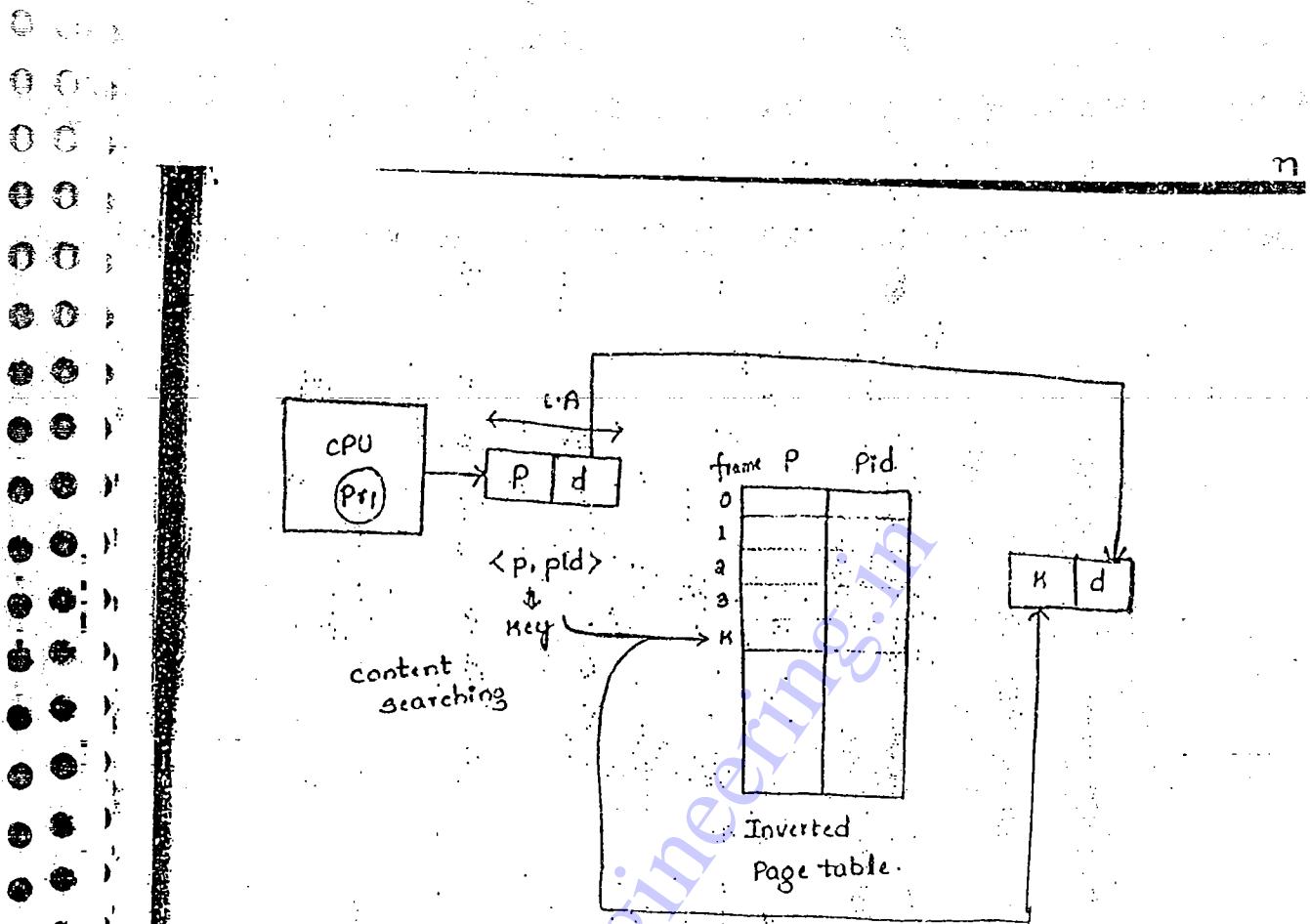
Inverted paging :-



All the processes maintain single primary memory each.

within the page tables, frames are stored within the page. The Inverted page table contains pages get stored in frames.





\* The Inverted page table maintains the "key" generated by adding  $\langle p, pid \rangle$ , thus, by referring the key, the particular page gets accessed, and the same offset is considered. This reduces the time for searching the content.



06/07/2010

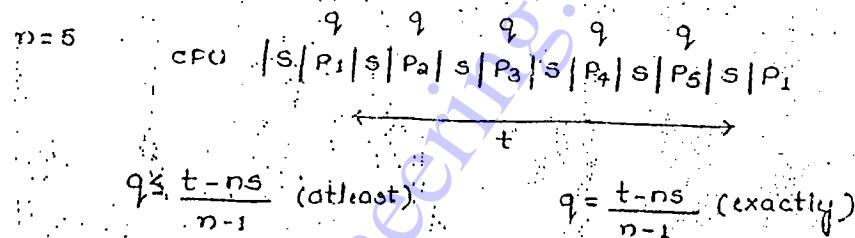
Thursday

Page no : 34

(1)  $n$  processes

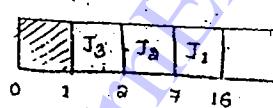
context switching = 5 seconds.

$$T.Q = q$$



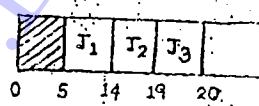
(4)

(a) {3, a, 1}, 1



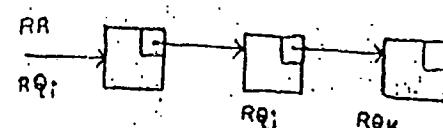
| J. | A.T | B.T |
|----|-----|-----|
| 1  | 0.0 | 9   |
| a  | 0.6 | 5   |
| 3  | 1.0 | 11  |

(d) {1, a, 3}, 5.



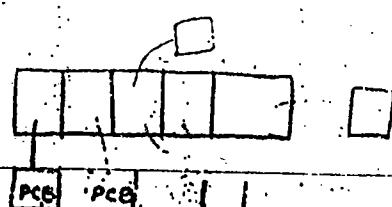
P.no: 48

(1)



priority of process increases

Draoback : starvation.



P.D.C. 144

(9)

$$(a) \text{ CPU Efficiency } (\mu) = \frac{T}{T+S} \quad (TQ = \infty)$$

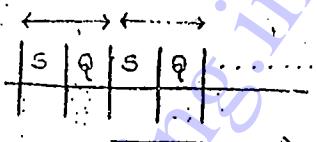
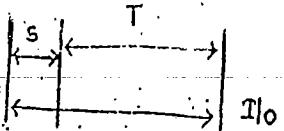
$$(b) \mu = \frac{T}{T+S} \quad (TQ > T)$$

$$(c) \mu = \frac{Q}{Q+S} \quad (S < Q < T)$$

$$(d) \mu = \frac{Q}{S} \quad (Q = S)$$

$$= .50\%$$

$$(e) \mu = 0 \quad (Q = 0)$$



(10) Trap & Interrupt :

Trap : Indicates error condition always

Interrupt : Indicates error condition (sometimes but not always)

0706176198

Thursday

Pno: 101

(6).  $n=4 \therefore m=3, (r_1, r_2, r_3) = (8, 3, 8)$

|                | n | Alloc | Request | Available                                                                   |
|----------------|---|-------|---------|-----------------------------------------------------------------------------|
| P <sub>1</sub> |   | 1 1 0 | 1 0 0   | 0 0 1                                                                       |
| P <sub>2</sub> |   | 1 0 1 | 0 1 1   | safe = {P <sub>3</sub> , P <sub>2</sub> , P <sub>1</sub> , P <sub>4</sub> } |
| P <sub>3</sub> |   | 0 1 0 | 0 0 1   |                                                                             |
| P <sub>4</sub> |   | 0 1 0 | 0 2 0   |                                                                             |

4107/0010

Saturday  
=

P.no: 140

5) min. size of partition = max. path length (in sizes of modules)

All the paths are considered as per their sizes

and the maximum one is represented.

P.no. 14a

(Q5)  $e \Rightarrow$  frame no.  $\rightarrow$  PAB = 64 MB

VAS = 8abit

Pa = 4 KB

P.T Size =  $N * e$

$$N = \frac{VAS}{Pa} = \frac{2^{32}}{2^12} = \frac{2^0}{2^1} = 1M$$

$$M = \frac{64MB}{4KB} = 2^{14} \approx 2 \text{ bytes}$$

$$\begin{aligned} m &= \frac{64MB}{4KB} \\ &= \frac{2^{26}}{2^{12}} = 2^{14} \end{aligned}$$

$$e = N * e$$

$$= 1M * 2B$$

$$= 2MB$$

P.no: 153

(13) LAS = 8KW (8 pages of 1024 words)

m) r size = 8k (8 frames of 1024 words)

$$LAS = 8 * 1024$$

$$= 8KW = 13 \text{ hits}$$

$$PAB = 8 * 1024 = 38K = 16 \text{ hits}$$

80/07/2010

friday

p.no: 150

(Q) (b)

$$IAS = PA6 = 2^{16} B$$

16 bits

|   |     |      |   |   |
|---|-----|------|---|---|
| f | V/I | P.P. | D | A |
|---|-----|------|---|---|

7 1 ... 3 .1 x

P.T entry

$$f = \frac{2^{16}}{2^9} = 2^7$$

page size = 512

$$= 2^9$$

p.no: 155

(Q6) (a) Stack:

Since, only top of the stack is accessed, so, there is a less no. of page faults.

∴ It is good.

(b) Hashed symbol table:

It distributes identifiers across the table. So, it generates more no. of page faults.

(c) Sequential search & Binary search:

↓  
likely to cause more page faults than seq. search.

(d) pure code:

Read only code (non-modified code).

∴ It is good to have non-modified in demand paged environment.

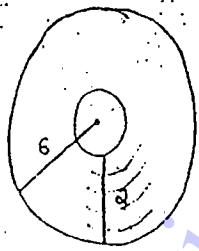
(f) Vector operations:-

Array (contiguous allocation)  $\Rightarrow$  good.

(g) Indirection:-

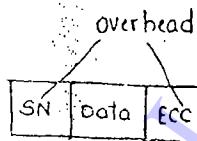
Linked lists (non-contiguous allocation)  $\Rightarrow$  bad.

(8)



$$\text{no. of tracks} = \frac{(6-a) \text{ cm}}{0.01 \text{ cm}}$$

= 400 tracks on one surface.



$$400 + 96 = 4096 = 4 \text{ KB} \quad \text{Each track has } 80 \text{ sectors}$$

$$(400 + 80 * 4 \text{ KB}) * 8 \text{ surfaces}$$

$$\Rightarrow 2^9 \times 2^5 \times 2^{12} \times 2^3 = 2^{29} = 512 \text{ MB}$$

$\approx 500 \text{ MB}$

$$(b) R = \frac{60}{3600} \text{ Sec}$$

$$\frac{60}{3600} \text{ s} = \frac{80 * 4 \text{ KB}}{3600} \text{ sec}$$

$$\frac{8000 * 80 * 4 \text{ KB}}{3600} = \frac{13.33}{60} \text{ sec}$$

P.no: 184

(4) program size = 64 KB

$$T.S = 80 \text{ ms}$$

$$R = 80 \text{ ms}$$

$$S.T = 30 \text{ ms}$$

$$20 \text{ ms} = 32 \text{ KB}$$

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

$$P.S = 2 \text{ KB}$$

$$\text{no. of pages} = \frac{64 \text{ KB}}{2 \text{ KB}} = 32 \text{ KB}$$

$$(a) \text{ Time for loading one page} = 30 \text{ ms} + 10 \text{ ms} + \frac{20 * 2}{32} \text{ ms}$$

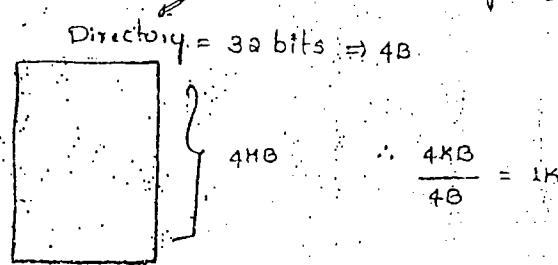
(b). page stored on track to go to that track; we have seek time

$$= 30 \text{ ms} + 10 \text{ ms} + \frac{20 * 4}{32} \text{ ms} * 16$$

04/08/2020

wednesday

(a) (a) max. no. of files  $\Rightarrow$  depends on directory



(b) DDA = 8 bits

$\therefore$  256 blocks possible for a file of 8 bits

But, there are 2 data blocks. So,  $= 256 - 2 = 254$ .

$$(254 * 4 \text{ KB}) \approx 1 \text{ MB}$$

P.no : 183

(1) B - blocks

S - DBS

D - DBA

F - free blocks

DBA = 0 bits

no. of bits = B

free = f

Bit map space consumption is 'B' bits (depends on blocks).

Size of free list (space) = F × D

$FD < B$

S - DBS

B - blocks

D - DBA

F - free blocks

Disk size = B · S

$2^D \geq B \quad (B \cdot S \leq 2^D)$

(maximum possible) 30MB 64MB

Block = 80K

OS = 80MB

DBA = 16 bits

DBS = 1KB

$2^{16} = 64KB * K = 64MB$

(max.)

P.no : 179

(1) b

(2) b (Since, it takes more access time, if placed anywhere).

(3) a

(4) b

(5) If there is only one process, then there is no matter whether to use any disk scheduling approach.

so  $\Rightarrow 0\%$

(6)

**a block**  
 In use :   
 free : 

is not so serious error

but just

reduces disk size

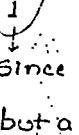
52

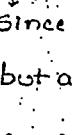
**In use :**

**b block**

**free :**







Since it is free, it may also given to other file,  
 but a file is already being accessing. So, it is  
 considered as a serious error.

To solve this error :-

copy the file to some other block and use it.

P.no: 37

(a5) (a)

(a6) (a)

6 P.no : 150

(a) VA = 48 bits

PA = 32 bits

no. of pages = BK =  $2^{13}$

no. of entries =  $\frac{2^{48}}{2^{13}} = 2^{35} = 32G$  entries

Inverted page table :-

$\frac{2^{32}}{2^{13}} = 2^{19} = 512K$  entries