Deadlock

A Process requests resources, if resources are not available at that time, the Process again unless a waiting state Sometimes, a waiting Process is never able to change state, because the resources it has requested are held by other waiting Processes. This situation is called deadlock

Necessary conditions for deadlock

Deadlock can arise if following four conditions hold simultaneously

1. Mutual Exclusion → Atleast one resource must be held in a nonsharable mode, that is only one Process at a time can use the resource. If other Process requests the resource, the requesting Process must be delayed until the resource has been released.

2 Hold and Wait

A Process must be holding atleast one resource and waiting to acquire additional resource that are currently being held by other Processes

3 No Preemption

Resources can't be preempted that is a resource can be released only voluntarily by process holding it, after that process has completed its task.

4 Circular wait →

A set $\{P_0, P_1 \text{---} P_{n3}$ of waiting Processes must exist such that P_0 is waiting for resource held by P_1, P_1 is waiting for resource held by P_2 —, P_{n-1} is waiting for resource held by P_n, P_n is waiting for resource held by P_0.

Resource Allocation graph

A graph consists of set of vertices V and set of edges E.

V is partitioned into → $P = \{P_1, P_2 - P_n\}$ (Set consisting of all active processes)

→ $R = \{R_1, R_2 - R_m\}$ Consisting of all resource types in the system

A directed edge from $P_i \rightarrow R_J$ signifies that process P_i has requested resource R_J. So it is request edge

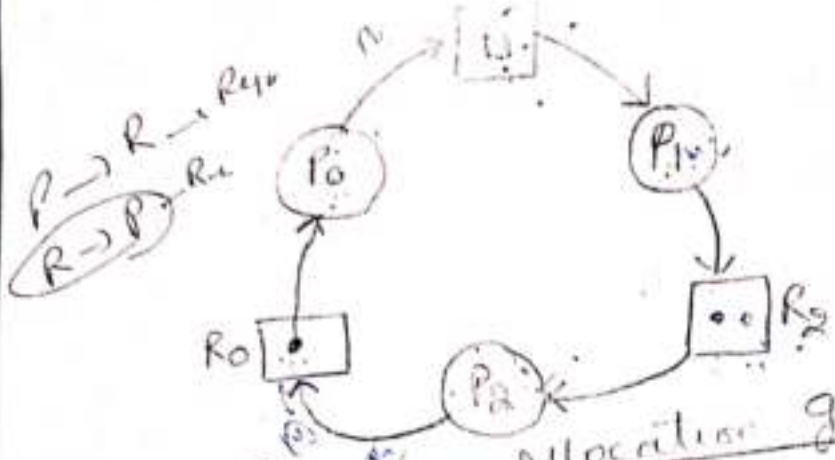
→ A directed edge $R_J \rightarrow P_i$ signifies that Resource is allocated to the process. This is called assignment edge

→ Process P_i is represented as circle

Resource R_J is represented as rectangle

Resource types

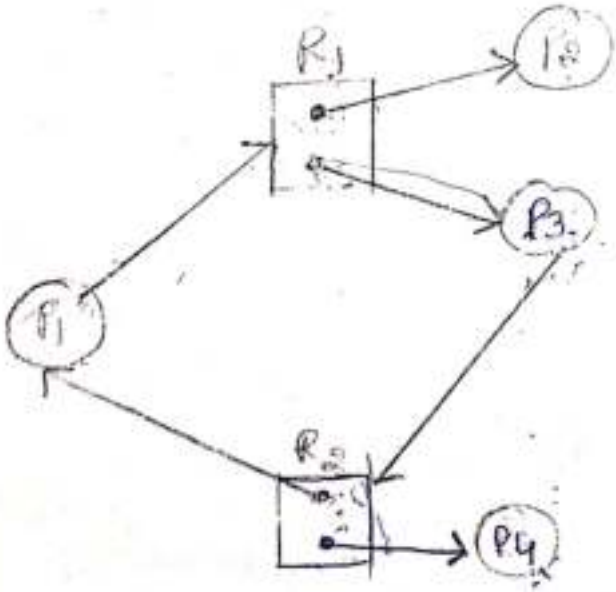
If in a Resource Allocation graph there is single instance of every resource then cycle in Resource Allocation graph is sufficient to indicate the deadlock



Cycle
$P_0 \rightarrow R_1 \rightarrow P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_0 \rightarrow P_0$
indicates deadlock for one instance of each resource.

a) Resource Allocation graph

If there are multiple instances of resources, then existence of cycle doesn't mean that there is a deadlock. In this example there is a cycle $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$ but there is



Cycle
$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
but no deadlock.

Fig
Resource allocation graph with cycle but no deadlock.

no deadlock because P_4 can release R_2 which can be granted to P_3 thus avoids deadlock.

Deadlock Prevention

Deadlock Prevention provides a set of methods for ensuring that one necessary condition for deadlock cannot hold. It can be done restraining how requests can be made.

1. Mutual Exclusion → Mutual Exclusion must hold for non-shareable resources eg Printer can't be simultaneously shared by several processes.

Sharable resources don't require mutual exclusion access eg Read only files several files can be read simultaneously. In general we cannot prevent deadlock by denying the mutual exclusion condition because some resources are non-shareable

2. Hold and wait

To ensure Hold and wait never occurs in the system, we must guarantee that, whenever a process requests a resource it doesnot hold any other resource one Protocol is that each process requests and is allocated resources Prior to its execution.

An alternative Protocol allows a process to request resources only when it has none A

Process can request additional resources, however, it must release all the resources that is currently allocated

To illustrate the difference consider a process that copies data from DVD drive to file on disk, sorts the file and then prints the result to the printer

In First Protocol all the resources must be requested at beginning of process, ie (the DVD drive, disk file and printer. It will hold the printer for its entire execution, even though it needs printer only at the end

The Second method allows process to request initially only DVD Drive and disk file. It copies DVD drive to disk & then releases both DVD drive and disk file. The process then again has to request disk file & printer.

Both Protocols have Limitation

1. Poor Resource Utilization → In first protocol printer was kept during entire process but it was needed only at the end
2. Starvation → A process that needs several resources may have to wait indefinately because atleast one of resource that it needs is always allocated to some other

3 No Preemption →

To ensure this condition doesn't hold we can use following protocol:

If a process is holding some resource and requests another resource that cannot be immediately granted than all resources that process is currently holding are preempted (released). The process is waiting for which the resources are added to the list of resources restarted only when it can regain the old resources as well as new resource it requires

4 Circular wait → To ensure this condition doesn't occur we impose a total ordering on all resource types and to require that each process requests resources in an increasing order of enumeration.

e.g Resources are tape drive, disk drive & printer

F (Tape drive) = 1
F (disk drive) = 5
F (Printer) = 12

A process can request resources in increasing order of enumeration

eg → a process wants to use tape drive and printer at same time, must request the tape drive first and then request the printer as f(tape drive) = 1, is less than f(Printer) = 12

Alternatively, we can require that a process requesting R_J, must have released any resource R_i so that $F(R_i) \geq F(R_J)$ i.e. it wants tape it must release printer. So these 2 conditions ensure (deadlock) circular wait never occurs, hence deadlock

Deadlock Avoidance ⇒

Deadlock Avoidance means deadlock can occur but we can recover from it. It requires additional information about how resources are to be (allocated) requested

eg → in a system with one tape drive and one printer, the system must need to know that process P will need tape drive first

then Prentice. With this complete knowledge of sequence of events, system can decide for each request whether or not the process should wait in order to avoid a possible deadlock

Safe State →

A state is safe, if the system can allocate resources to each process in some order and still avoid deadlock. More formally a system is in safe state only if there is a safe sequence.

A safe state is not a deadlocked state. Conversely deadlocked state is unsafe state. An unsafe state may lead to deadlock

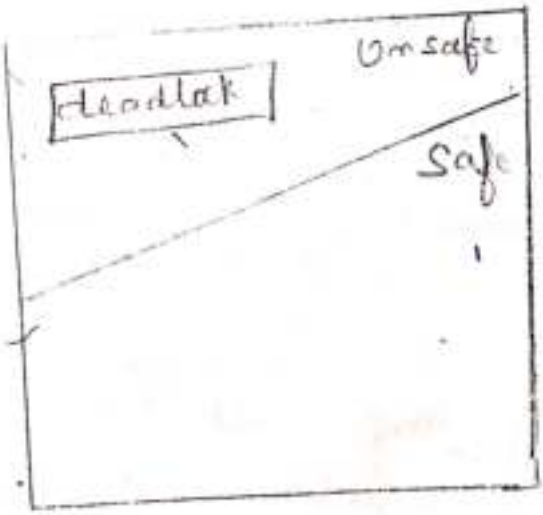


Fig: Safe, Unsafe & deadlocked state spa

Banker's Algorithm
(VVV imp)

In this algorithm, when a new process enters the system it must declare the maximum no of instances of each resource type it may need. This no must not exceed the total no of resources in the system

When user requests a set of resources the system must determine whether there is a safe sequence

Safety Algorithm

This algorithm is used for finding whether or not system is in safe state

1. Let Work and Finish be vectors of Length m and n respectively. Initialize Work = Available and Finish[i] = false for all i = 0, 1 —— n-1
2. Find an index i such that
   a) Finish[i] == false
   b) Need_i ≤ Work
   If no such i exists, goto step (4)

3 Work = Work + Allocation[i].
Finish[i] = true.
Goto step 2.

4 If Finish[i] == true for all i, System is in safe state;

Resource-Request Algorithm → Request_i be request vector for process Pi

1 If Request_i ≤ Need_i, goto step 2, otherwise error

2 If Request_i ≤ Available, goto step 3 otherwise Pi must wait, since resources are not available

3 Have the system pretend to have allocated resources to Process Pi by modifying state as follows;

Available = Available - Request_i
Allocation_i = Allocation_i + Request_i
Need_i = Need_i - Request_i

If resulting resource-allocation is 'safe', transaction is complete & Pi is allocated its resources. However, if new state is unsafe, Pi must wait for Request_i and old resource allocation state is restored

Example

Consider system with 5 Processes P_0 to P_4 & 3 resources A, B, C. A has 10 instances, B has 5, C has 7 instances. Suppose at T_0 snapshot of system is as follows:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Time Available Allocation

1) Find Need matrix
2) Is system in safe state
3) If request (1, 0, 2) by P_1 be granted immediately

Soln (a) Need = Max − Allocation

Need

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

P_1, P_3, P_4, P_0, P_2

(b) No Available must satisfy max need

Firstly Request of P_1 will be satisfied and after completion of P_1

Now :- Available = Available + Allocation 1

$$= \begin{array}{r} A\ B\ C \\ 3\ 3\ 2 \\ +\ 2\ 0\ 0 \\ \hline 5\ 3\ 2 \end{array}$$

Available = 5 3 2

2 P_2 can't be satisfied as it requires 6 instances of A but 5 are available.

3 P_3 is satisfied after P_1 and after complet of P_3

Available = Available + Allocation 3

$$\begin{array}{r} A\ B\ C \\ 5\ 3\ 2 \\ 2\ 1\ 1 \\ \hline 7\ 4\ 3 \end{array}$$

Available = 7 4 3

4 Now P_4 can be satisfied & after completion of P_4

Available = Available + Allocation 4

$$\begin{array}{r} A\ B\ C \\ 7\ 4\ 3 \\ 0\ 0\ 2 \\ \hline 7\ 4\ 5 \end{array}$$

Available = 7 4 5

5 Now P_0 can be satisfied after completion of P_0

Available = Available + Allocation$_0$

A	B	C
7	4	5
0	1	0
7	5	5

Available =

6 Now P_2 can be satisfied and after completion of P_2

Available = Available + Allocation$_2$

A	B	C
7	5	5
3	0	2
10	5	7

Available =

∴ Safe sequence = $< P_1, P_3, P_4, P_0, P_2 >$

③ P_1 requires $(A\ B\ C) = (1\ 0\ 2)$

(a) $(1, 0, 2) \leq$ Need$_1$

$(1, 0, 2) \leq (1, 2, 2)$

(b) $(1, 0, 2) \leq$ Available

$(1, 0, 2) \leq (3, 3, 2)$

Need P_1 = A B C
1 2 2
− 1 0 2
0 2 0

Need P_1 = 0 2 0

Available = A B C
3 3 2
− 1 0 2
2 3 0

Allocation P_1 = A B C
2 0 0
+ 1 0 2
3 0 2

Now new state becomes

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

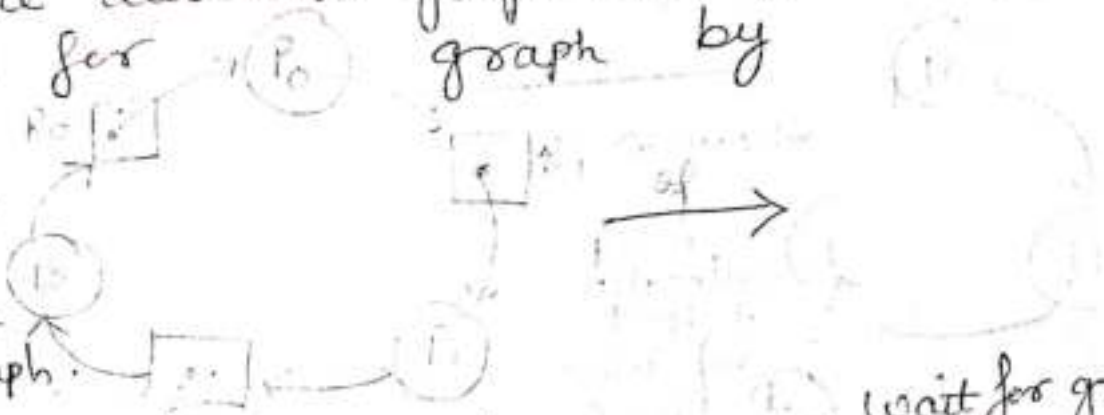
Now new safe sequence can be found = $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ ∴ request by (1, 0, 2) by P_1 can be granted

Deadlock Detection

Case I

If there is a single instance of a resource then cycle in Resource Allocation graph indicates a cycle.

A Resource allocation graph can be converted into wait for graph by removing the resources from Resource allocation graph.



Resource Allocation graph

Wait for graph

Case II → If there are multiple instances of resource then we use safety algorithm to detect deadlock

Algorithm for deadlock detection

1 Let work and finish be vectors of length m & n respectively.
Initialize work = Available for i=0, 1—n-1
If Allocation$_i$ ≠ 0 then finish[i] = false
Otherwise finish[i] = true

2 Find an index i such that both
a) Finish[i] = = false:
b) Request$_i$ ≤ work.

If no such i exists, goto step (4)

(3) Work = Work + Allocation
Finish[i] = true
goto step 2

(4) If Finish[i] == false for some i, $0 \le i \le n$
then system is in deadlock state. Moreover
if Finish[i] == false, then Process P_i is
deadlock

Recovery from Deadlock →
When an detection algorithm determines a deadlock, several alternatives are available

1 One possibility is to inform the operator that deadlock has occured and then recover manually from the deadlock

2 Second possibility is to let the system recover automatically from deadlock.

One simple alternative is to abort one or more processes to break circular wait.

Second method is Resource preemption from one or more deadlocked processes.

Process Termination

* Abort all deadlocked processes → it is an expensive method as all the processes needs to be restarted

* Abort one process at a time until deadlock cycle is eliminated

Resource Pre-emption

1) Select a victim → which resources and which processes are to be preempted? We must select order of preemption to minimize cost

2) Rollback → If we preempt a resource from the process, what should be done with that process, we must roll back to some safe state and then restart again

3) Starvation → A process can be selected victim only a finite no of times, so that process doesnot starve

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

(a) What is content of matrix need ?

b) Is system in safe state ?

c) If a request from process P_1 arrives for (0, 4, 2, 0) can the request be granted immediately ?

Need = Max − Allocation

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

Available

A	B	C	D
1	5	2	0
0	4	2	0
1	1	0	0

P_1

Safe sequence is $< P_0, P_2, P_3, P_4, P_1 >$

As need for $P_0 \leq$ Available. After completion P_0.

Available = Available + Allocation =

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

Since Need(P_1) > Available ∴ request of P_1 cannot be satisfied

As Need(P_2) < Available ∴ After completion of P_2 New Available = Available + Allocation

Available ⇒	1	5	3	2
	1	3	5	4
New Available ⇒	2	8	8	6

As Need(P_3) < Available ∴ After completion of P_3 New Available =

	2	8	8	6
	0	6	3	2
New Available ⇒	2	14	11	8

Need(P_4) < Available ∴ After completion of P_4 New Available =

	2	14	11	8
	0	0	1	4
New Available =	2	14	12	12

Need(P_1) < Available
∴ After completion of P_1 New Available =

	2	14	12	12
	1	0	0	0
	3	14	12	12

(c) If a request from Process P_1 for (0, 4, 2, 0) can be granted immediately?

Request ≤ Available

(0,4,2,0) (1,5,2,0)

New snapshot

	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	0	0	1	1	0	0
P1	1	4	2	0	0	3	3	0				
P2	1	3	5	4	1	0	0	2				
P3	0	6	3	2	0	0	2	0				
P4	0	0	1	4	0	6	4	2				

∴ with this snapshot we can get a safe sequence < P0 P2 P3 P4 , P1 >

∴ Request of P_1 for (0,4,2,0) can be granted immediately.

Deepak Singla A.P CSE PIET

(17)

UNIT-3

Memory Management

→ Memory management is the functionality of an operating system which handles or manages primary memory. C.S.E 4th DS-U-3 = (60)

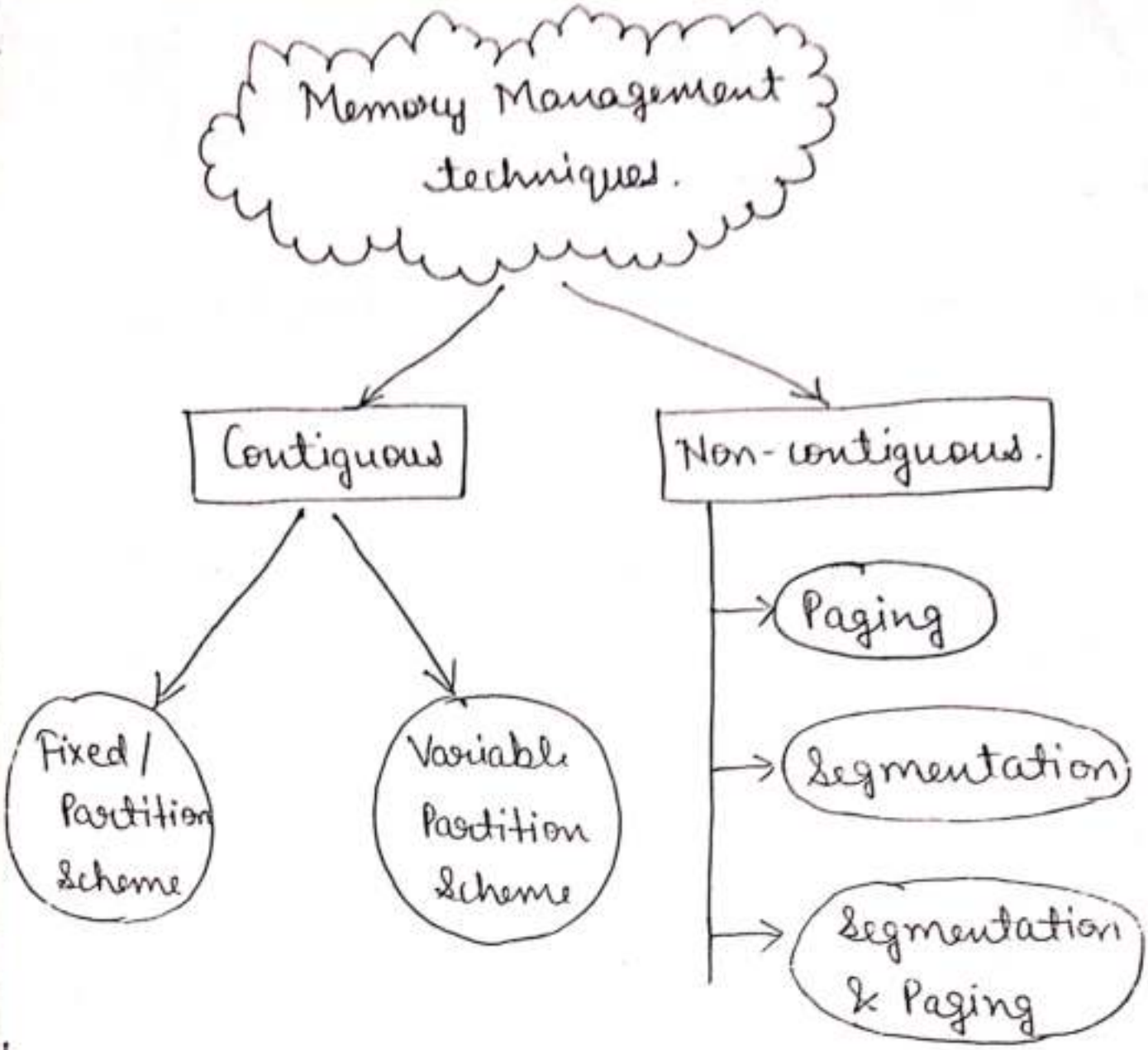
→ Memory management keeps track of each and every memory location either it is allocated to some process or it is free.

→ It checks how much memory is to be allocated to processes.

→ It decides which process will get memory at what time.

→ It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

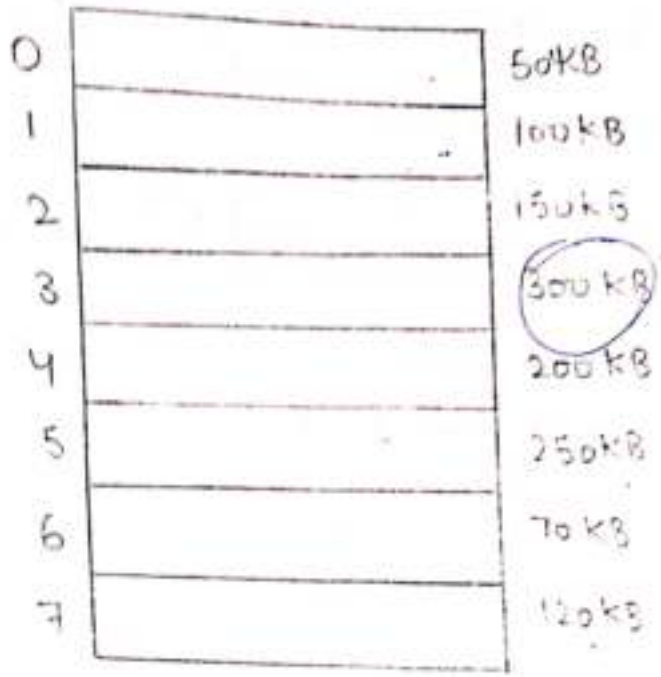
→ For the execution of a program, program together with data may access must be in the main memory during execution.



↳ Fixed Partition Scheme :-

→ Every partition is associated with limit registers.

- Lower limit :- starting address of partition.
- Upper limit :- ending address of partition.



Memory

→ Here, number of partition are fixed

→ No. of partition = 8

Maximum size of partition = 300KB

In fixed partition scheme, memory is divided into fixed number of partition.

↓

means, no. of partitions are fixed.

In one partition, only one process will be allocated.

Disadvantage :-

(1) The degree of multiprogramming (no of processes in main memory) is restricted by the no. of partition in the memory.

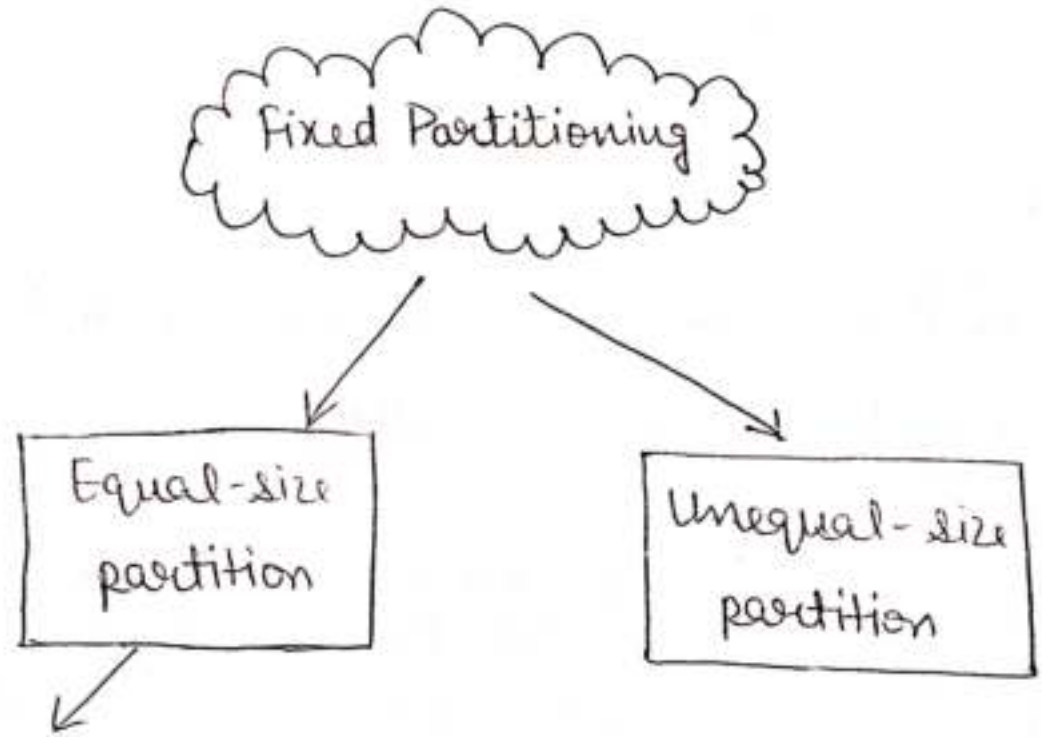
(2) The maximum size of process is restricted by maximum size of partition.

Eg Process size of 350 KB can't be placed in any partition.

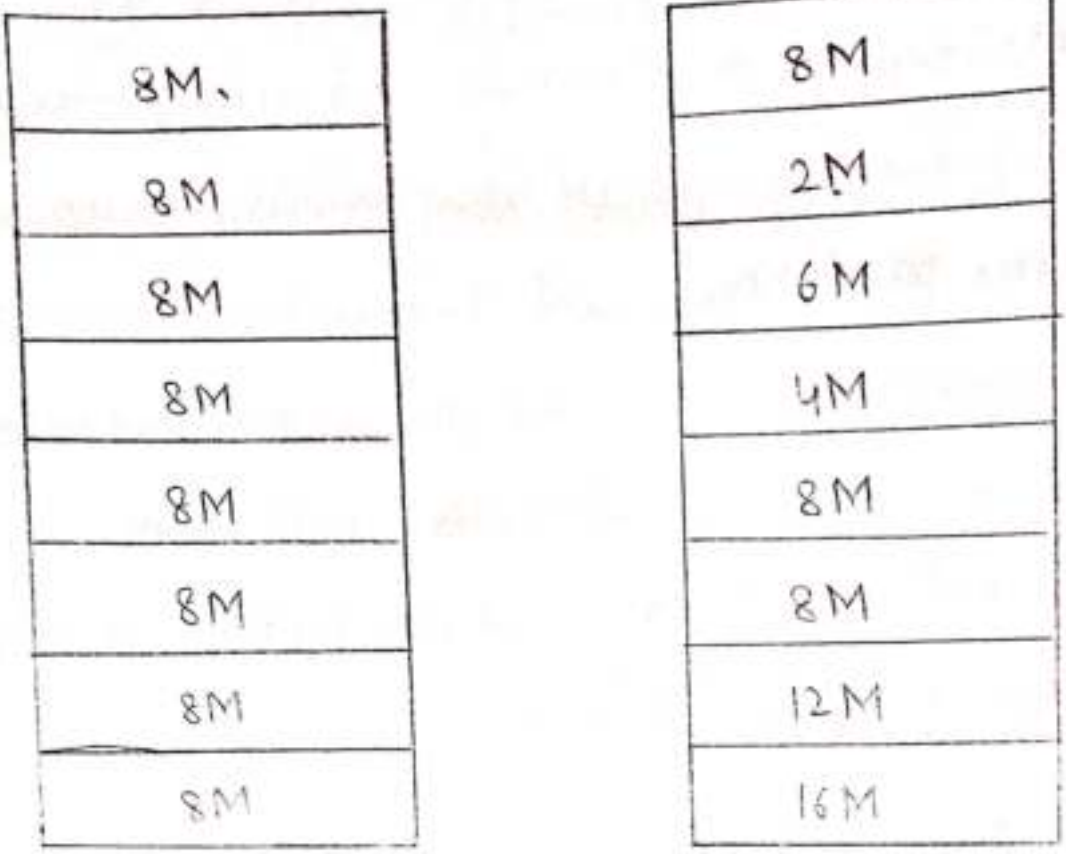
∵ max partition size = 300 KB

⇒ Two alternatives for fixed Partitioning :-

- Equal size partition
- Unequal size partition



Any process whose size is less than or equal to the partition size can be located into any available partition.



(equal-size partition) (Unequal-size partition)

(Example of fixed partitioning of a 64 Mbyte memory)

Difficulties with the use of equal size fixed Partioning scheme :-

(1) A program may be too big to fit into the partition. In this case programmer must design the program using **overlays.**

* Overlays :- When a process has size larger than the amount of memory allocated to it

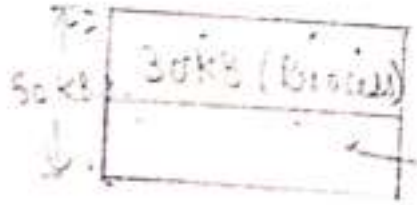
, we can use overlays.

(2) Main memory utilization is extremly inefficient. Any program, no matter how small, occupies an entire partition.

In our example, there may be a program whose length is less than 2 MByte, yet it occupies an 8-Mbyte partition. This is called as the

Internal fragmentation.

- Internal fragmentation :- If the process size is small as compared to the partition size, then memory which is wasted is called as internal fragmentation.



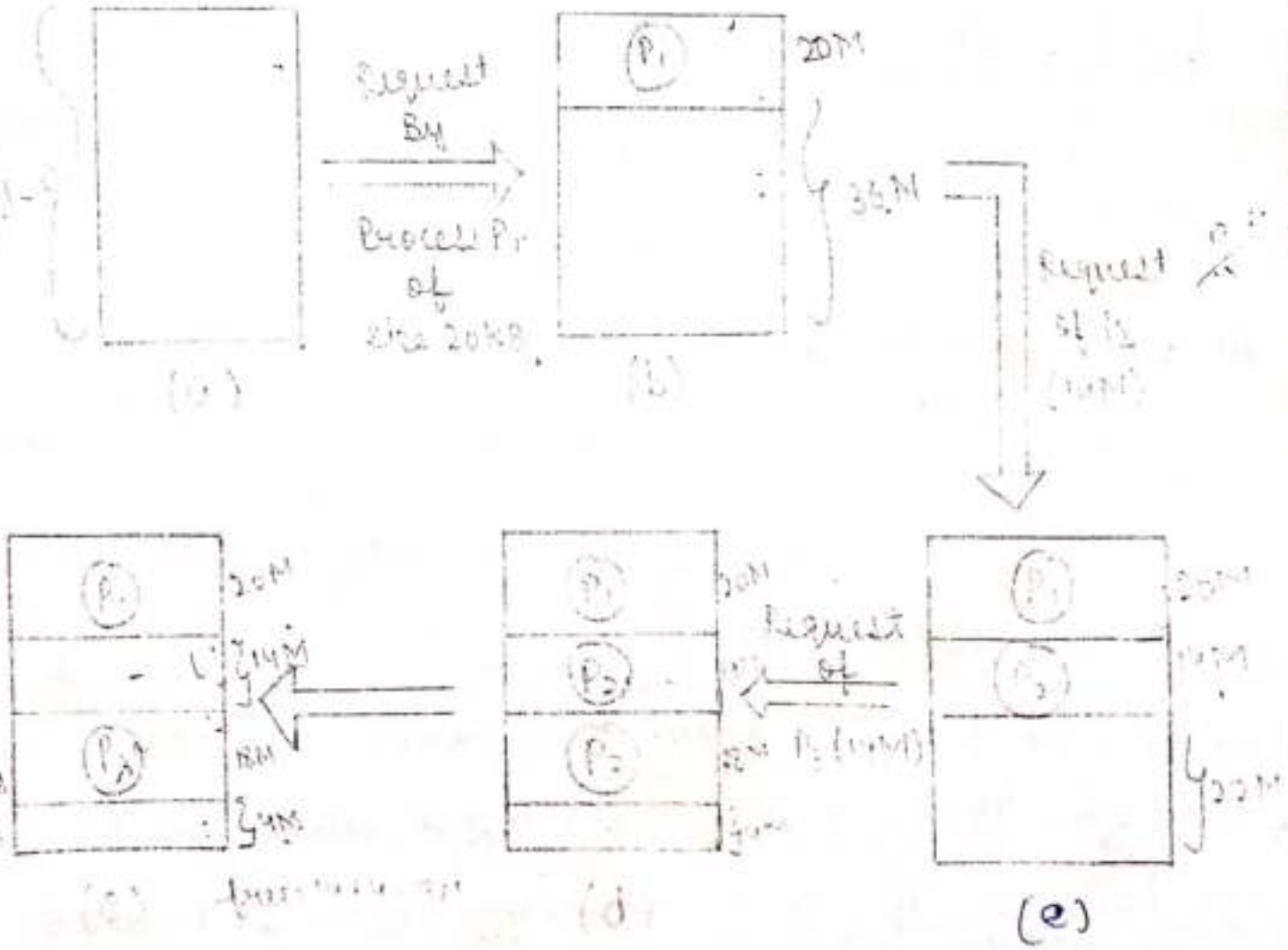
So 20KB will be wasted, this is called as internal fragmentation.

Variable Partion Scheme :-

→ In variable partition scheme, initially the memory will be single contiguous free block.

→ Whenever request by process comes in, accordin-gly partition will be made.

→ Problem of degree of multiprogramming and max size partition are removed.



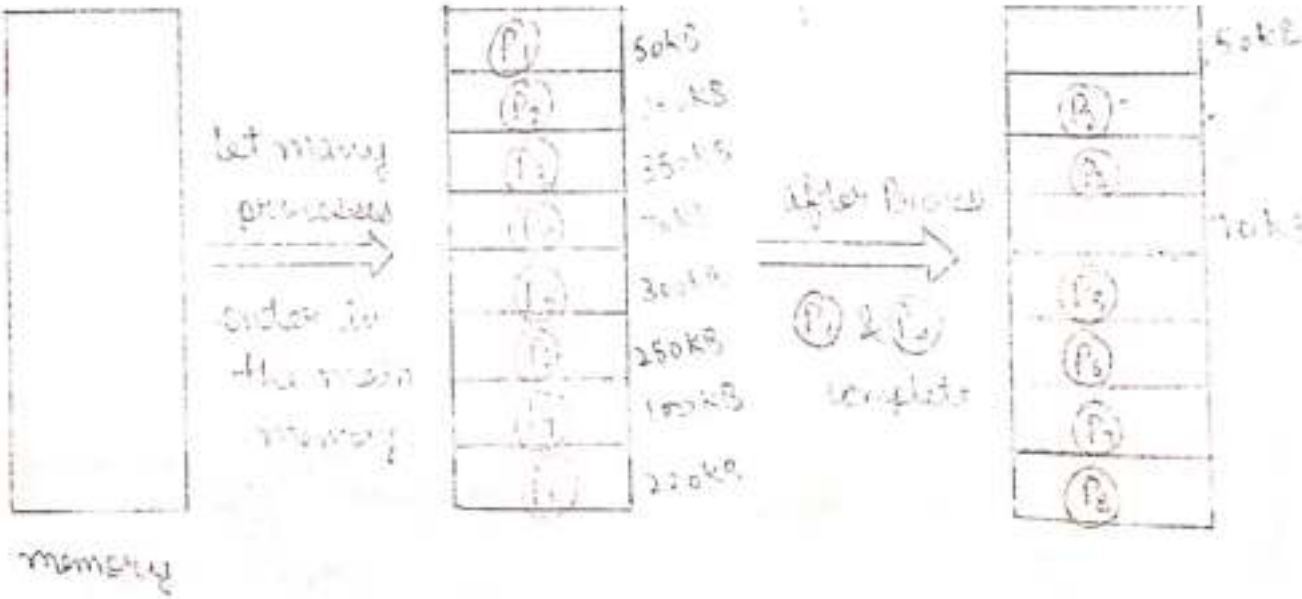
If process size = 16M i.e (P4) arrived

Available free space = 14+4
= 18M

Process (P4) cannot be allocated even if we have 18M free. This is known as the external fragmentation.

- External fragmentation happens due to contiguous memory allocation

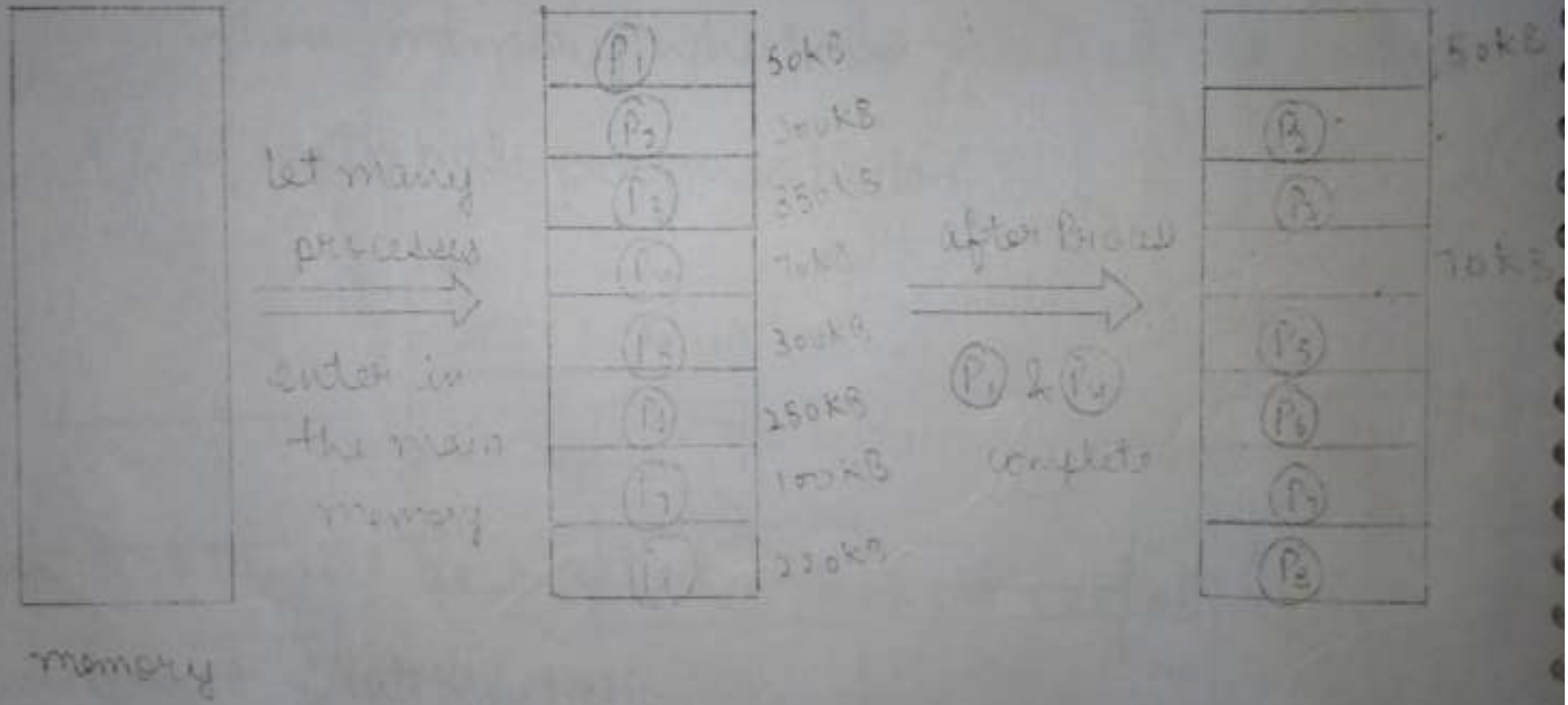
another example:-



Now, if process of size 100 kB comes even though we have 50 + 70 = 120 kB memory free. But, still we are not able to use it. This is known as

memory allocation

another example:-



Now, if process of size 100 kB comes even though

external fragmentation.

To avoid external fragmentation the below techniques are followed:-

Compaction:-

Moving all the processes towards the top or towards the bottom to make in a single contiguous place is called **compaction.**

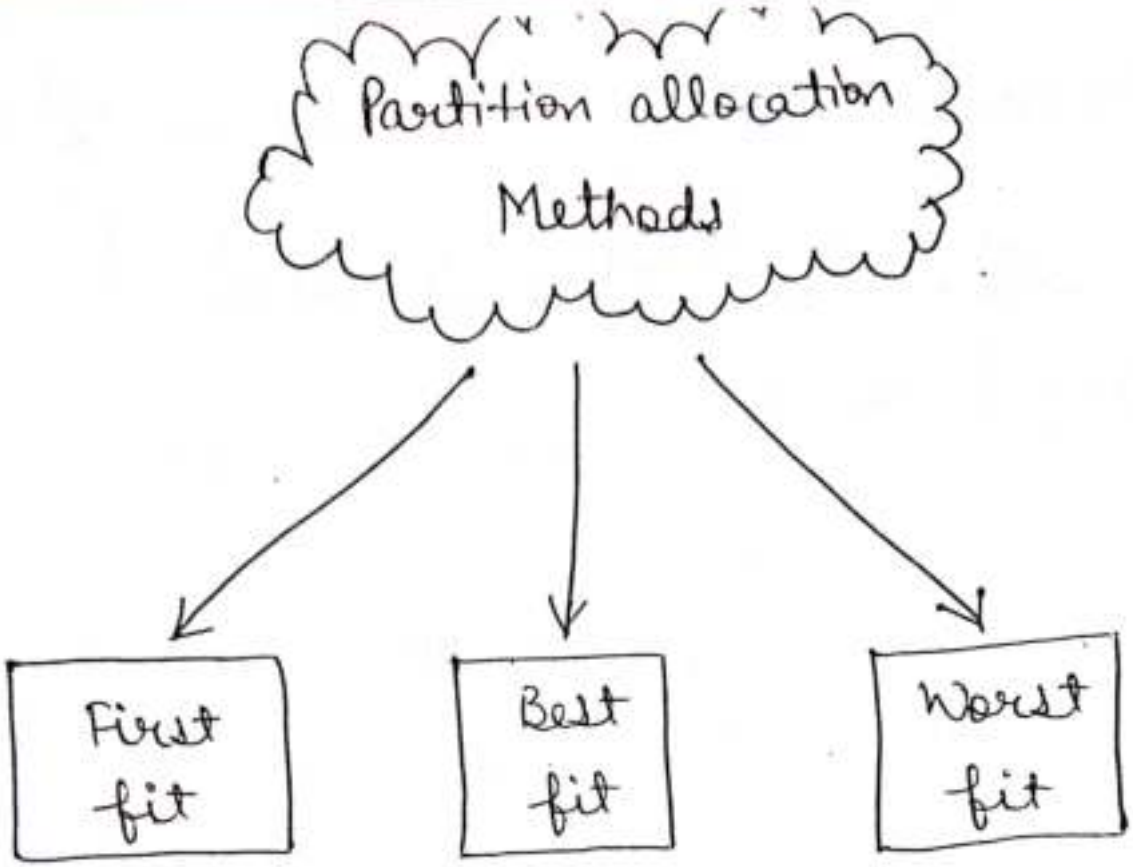
↓

The compaction idea is not practical because it disturb all the running process in the memory.

Compaction is possible only if relocation is dynamic and done at execution time.

Partition allocation method:-

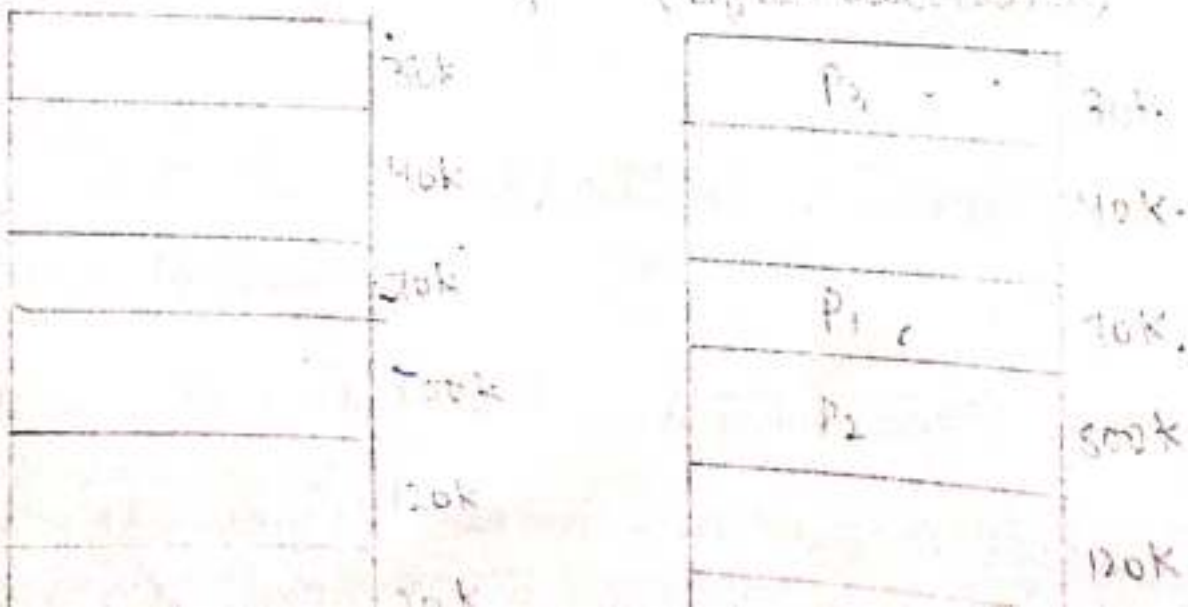
If multiple partitions are free in memory and there is some requirement then in which partition to accomodate, this decision is taken by partition allocation policies.



1) First fit :-

Allocate the process in a partition which is first sufficient partition from the top.

e.g. Let $P_1 = 65K$, $P_2 = 110K$, $P_3 = 20K$

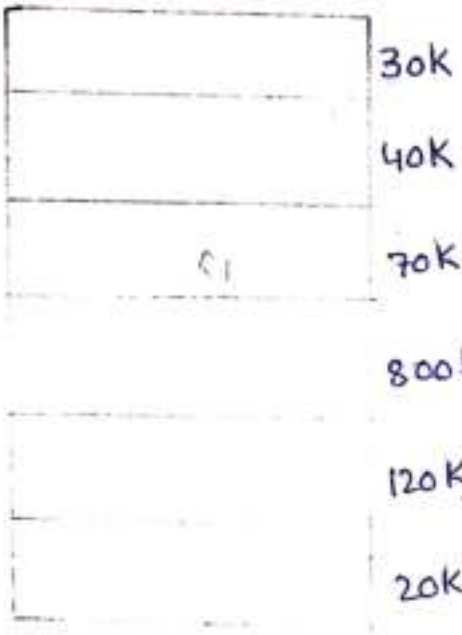


(2) **Best fit:-**

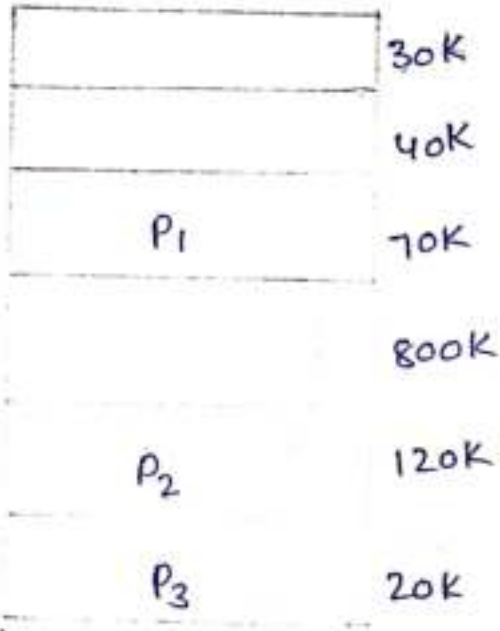
Allocate the process in partition which gave less internal fragmentation.

e.g. Let $P_1 = 65K$, $P_2 = 110K$, $P_3 = 20K$.

(Before Allocation)



(After allocation)



Wastage = 5 + 10 + 0 = 15K

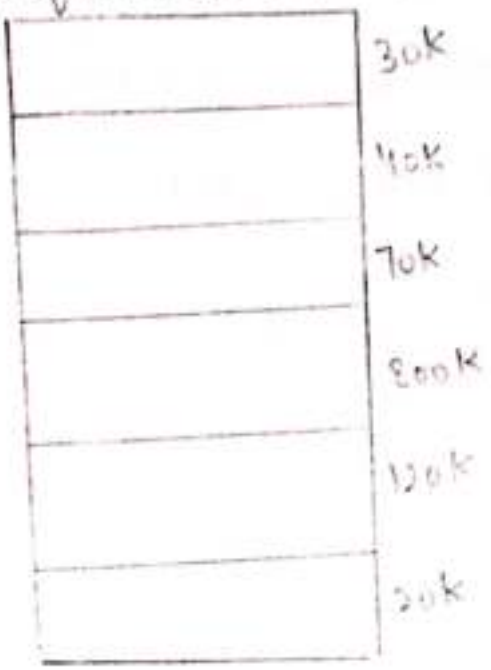
(3) **Worst fit:-**

Allocate the process in partition which gives more/max internal fragmentation.

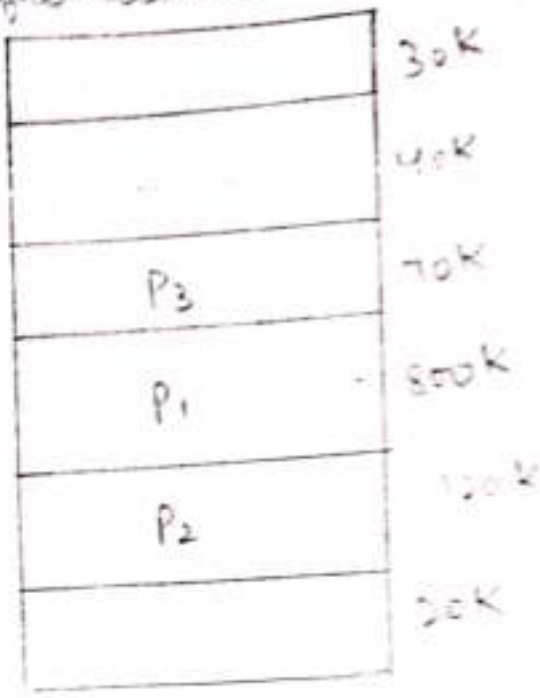
e.g.

Let $P_1 = 65K$, $P_2 = 110K$, $P_3 = 20K$

(Before allocation)



(After allocation)



MEMORY NUMERICALS

Assume each word is having size of 16 bits ~~capacity~~

Capacity of memory = No of words x word size

= 8X 16 bits

= 2X 2B = 16 Bytes.

Ques:- Consider a system where 19 bits are used to represent words of memory each word is having the size of 32 bits. Then what is capacity of memory.

Ques :- Consider a system which is having 128 K words in memory each word is having the size of 64 bits then find the capacity of memory.

Soln :- 128 K X 64 Bits

$= 128\,K \times 2^{10} \times 8$ Byte

$= 2^7 \times 2^{10} \times 2^3$ Byte

$= 2^{20}$ Byte

$= \boxed{1\,MB}$

Ques :- Consider a system where 19 bits are used to represent words of memory each word is having the size of 32 bits. Then what is the capacity of memory.

Sol :- If having 3 binary bits then $2^3 = 8$ words

So, No. of words $= 2^{19}$

Capacity of memory = No. of words x Size of each word

$= 2^{19} \times 32$ bits

$= 2^{19} \times 4$ bytes

$= 2^{19} \times 2^2$ bytes

$= 2^{21}$

= 2MB

Main memory is implemented using RAM chips.

↳ RAM chips Implementation :-

Ques RAM chip size = 128 B

Organize memory capacity of 16 KB

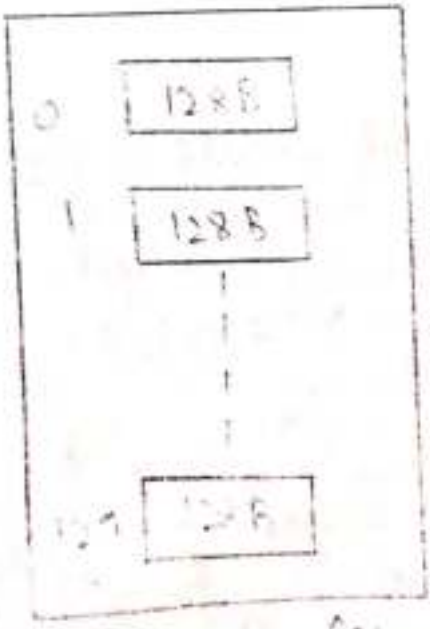
1. How many RAM chips are required.
2. Draw memory organisation map.

Sol :-

(1) $\frac{16\ KB}{128\ B} = \frac{2^4 \times 2^{10}}{2^7} = \frac{2^{14}}{2^7} = 2^7 = \boxed{128}$

means RAM chips has 128 words and each having 1B.

(2)



(Memory map)

128 B

↓ No. of words 128

↓ Size of each word

Ques:- RAM chip size = 256 x 1 bit

organize memory capacity of 32 KB

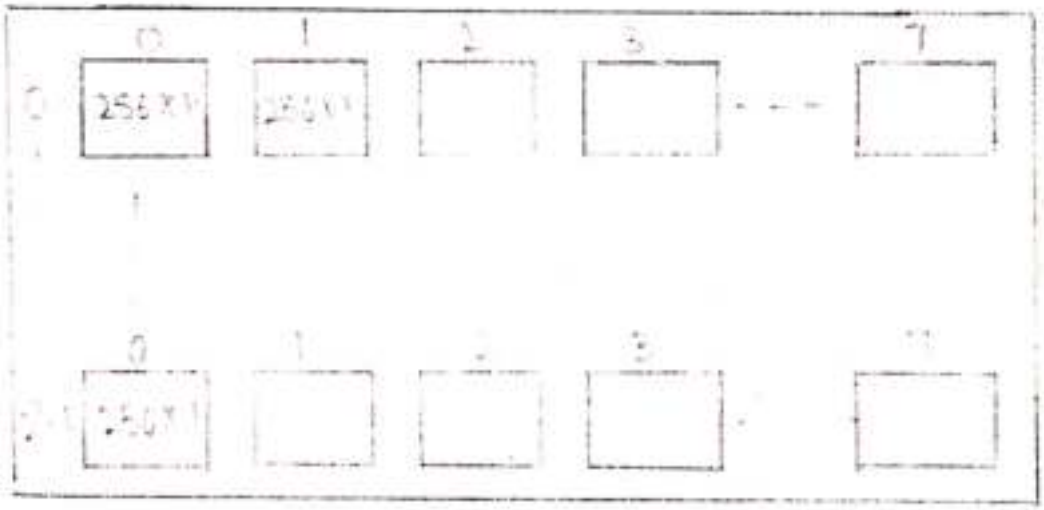
(a) How many RAM chips are required.

(b) Draw memory organisation map.

Sol^n:-

(a) $\frac{32\ KB}{256 \times 1}$ → columns, ↙ Rows $= \frac{2^5 \times 2^{10} \times B}{2^8} = \frac{2^{15}}{2^8} B$

$= \boxed{2^7 B}$



Memory representation is 1 Byte But RAM chip is giving 1 bit. So, 8 columns.

Ques:- RAM chip size = 512 x 2 bit

Organize memory capacity at 64 MB

Sol^n:-

$$\frac{64\,MB}{512 \times 2\ bits} = \frac{2^6 \times 2^{20} \times 2^3\ bits}{2^9 \times 2\ bits}$$

$$= \frac{2^{29}}{2^{10}} = 2^{19}$$

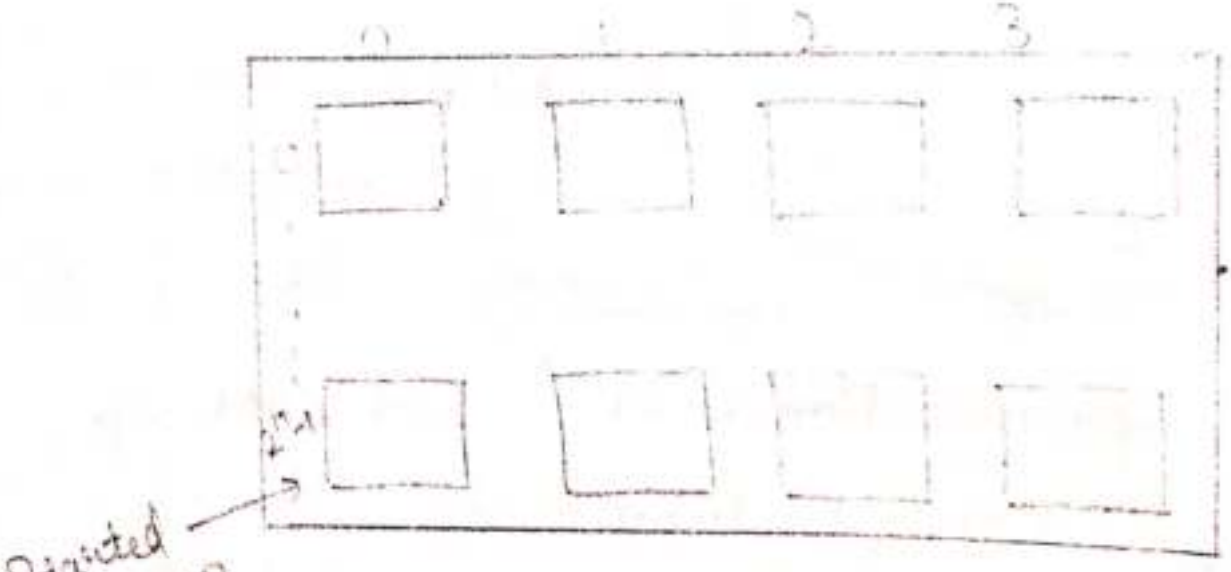
No. of rows & columns :-

$$\frac{64\,MB}{512 \times 2}$$

↙ Rows ↘ Columns. $\frac{2^3}{2} = 2^2$

$$= \frac{2^6 \times 2^{20}}{2^9}$$

$$= \frac{2^{26}}{2^9} = 2^{17}$$



Ques:- A main memory unit with a capacity of 4 Mega Byte built using 1M x 1 bit DRAM chips each DRAM is having the 1K rows of cells with 1K cells in each rows. The time taken for single refresh operation is 100 ns. The time required to perform refresh operation on all the cells in the memory unit.

Sol^{n}:- Memory capacity = 4MB

DRAM size = 1M x 1 bit

No. of rows = 1K

No. of columns = 1K

$\Rightarrow$ No. of RAM chips = $\frac{4MB}{1M \times 1 bit}$

$= 4 \times 8$

$= 2^2 \times 2^3$

$= 2^5$

$= 32$

$\Rightarrow$ Total cells = $1K \times 1K = 2^{20}$ cells.

$\Rightarrow$ 100 ns require to refresh 1 cell

required for all cells $\rightarrow 32 \times 100 \times 2^{20} \Rightarrow \boxed{3200 \times 2^{20} \text{ ns}}$

↳) Difference b/w logical address and Physical address

Logical address	Physical address
* It is a virtual address generated by CPU.	* It is located into the memory unit.
* The set of logical addresses generated by CPU in reference to a program is referred as logical address space.	* The set of all the physical address MAC to the corresponding local address is referred as physical address space.
* The user can view the logical address.	* The user can't view the physical address.
* The user use the logical address to access the physical add^{n}.	* The user can't access the physical address.
* The logical address is generated by	* The physical address is computed by a

the CPU.

* It is represented in the form of bits.
* Logical address (LA) ~~is~~ represented one of the address in LA.S
* If LA-S is given we can find out LA and vice versa.
* LA.S (logical address space) is the set of all the logical address generated by a program.

hardware device known as MMU (memory management unit)

* It is also represented in the form of bits.
* Physical address (PA) represent one of the address in P.AS
* If P.A.S is given we can find out PA and vice versa.
* P.A.S (physical address space) is the set of all the physical addresses.

Paging :-

→ Non contiguous memory allocation.

→ Solution to external fragmentation.

, Technique of mapping, the processor generated logical address to the physical address is called as **Paging**.

→ Physical memory is broken into fixed sized blocks called frames.

→ Logical memory is also broken into blocks of same size called pages.

→ When a process is to be executed, its pages are loaded into any available memory frames from Backing store

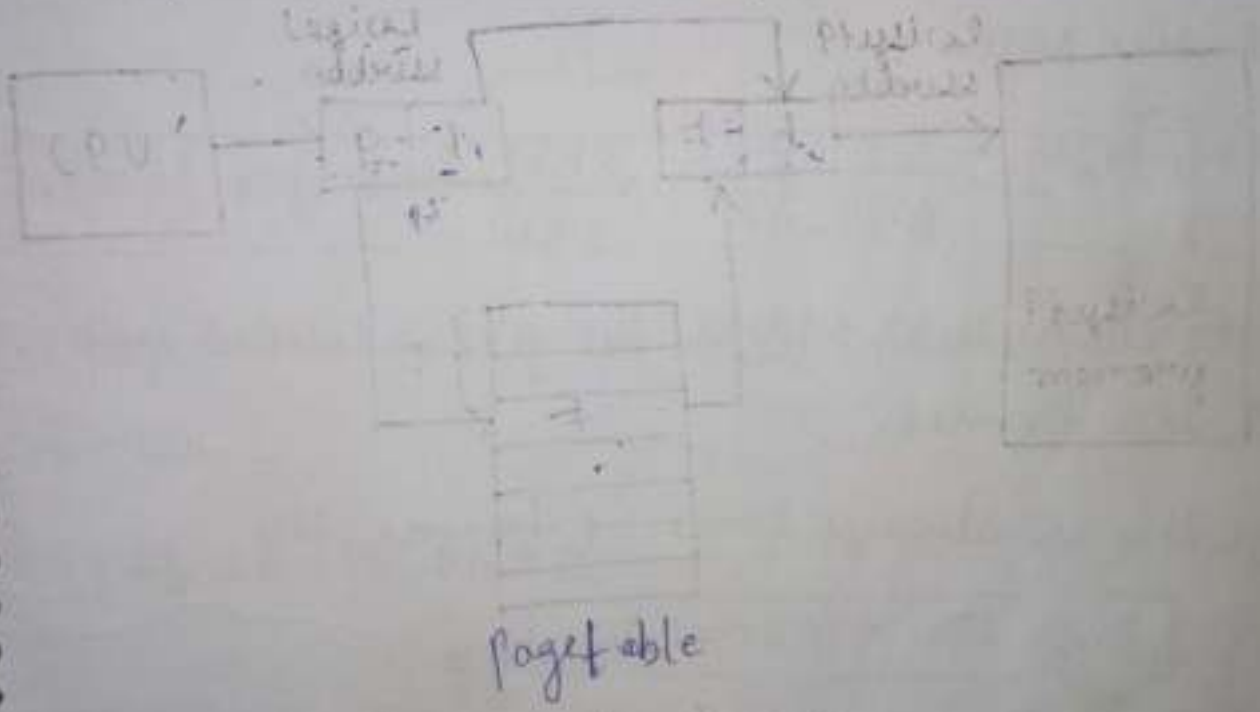
Paging hardware - Every address generated by the CPU is divided into two parts - a page number (P) and a page offset (d).

P	d

→ Page number is used as an index into a page table.

→ The page table contains Base address of each page in physical memory.

→ This page address is combined with page offset to define physical memory address.



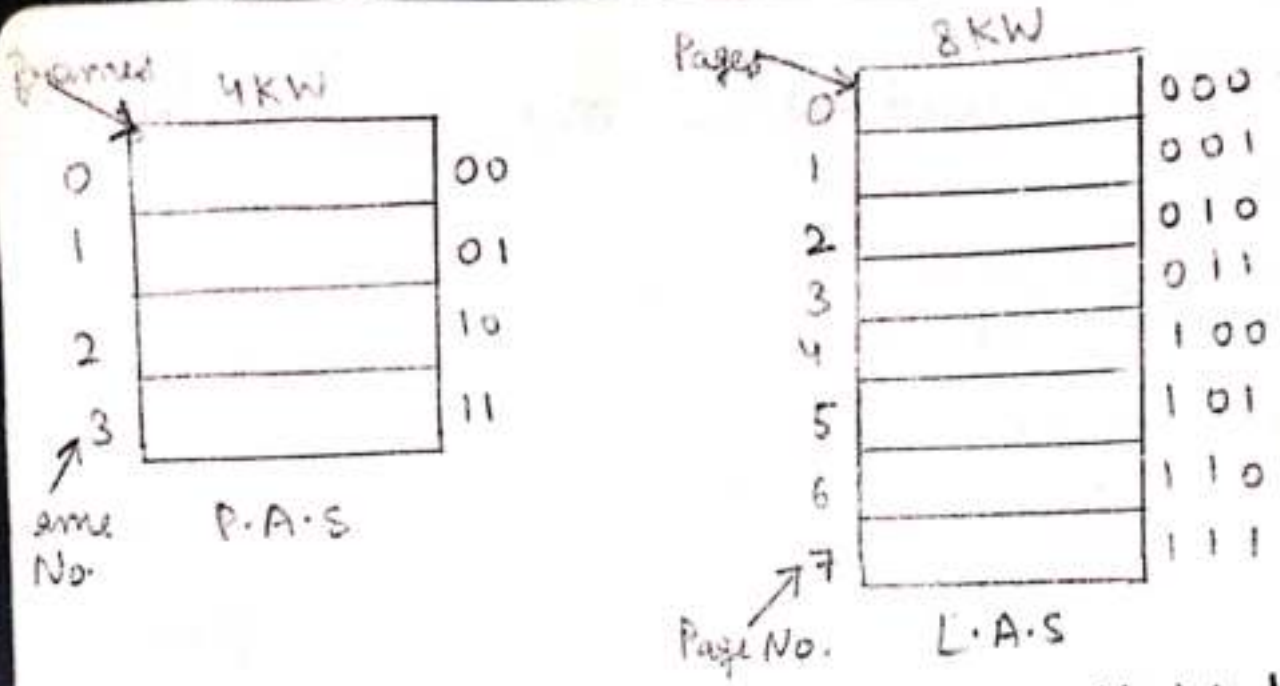
Example of paging :-

Assume Page size = 1KW

Let LA = 13 bits, L.A.S = 2^{13} = $2^3 \cdot 2^{10}$ = 8KW

P.A = 12 bits. P.A.S = 4KW

= $2^2 \cdot 2^{10}$



The logical address space (L.A.S) is divided into equal size pages.

$$\text{No. of pages} = \frac{\text{L.A.S.}}{\text{page size}} = \frac{8KW}{1KW} = 8$$

The P.A.S (physical address space) is divided into equal size frames.

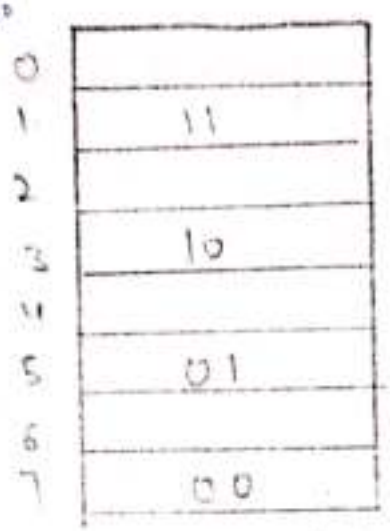
Page size is always same as frame size

$$\boxed{\text{page size} = \text{frame size}}$$

$$\text{No. of frames} = \frac{\text{P.A.S.}}{\text{frame size}} = \frac{4KW}{1KW} = 4$$

→ LAS is size of a process which is divided into various pages.

→ Whenever we apply paging page table will be maintained.

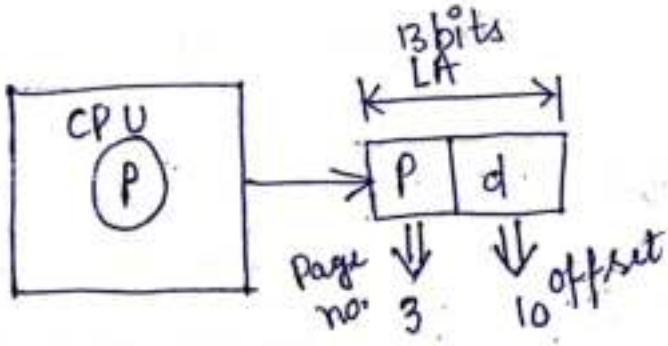


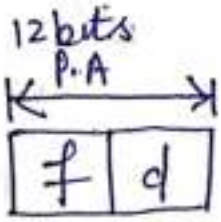
Page table

No. of page entry = no. of pages in L.A.S

The page table entry definatly contains the frame number.

So, if LA = 13, PA = 12





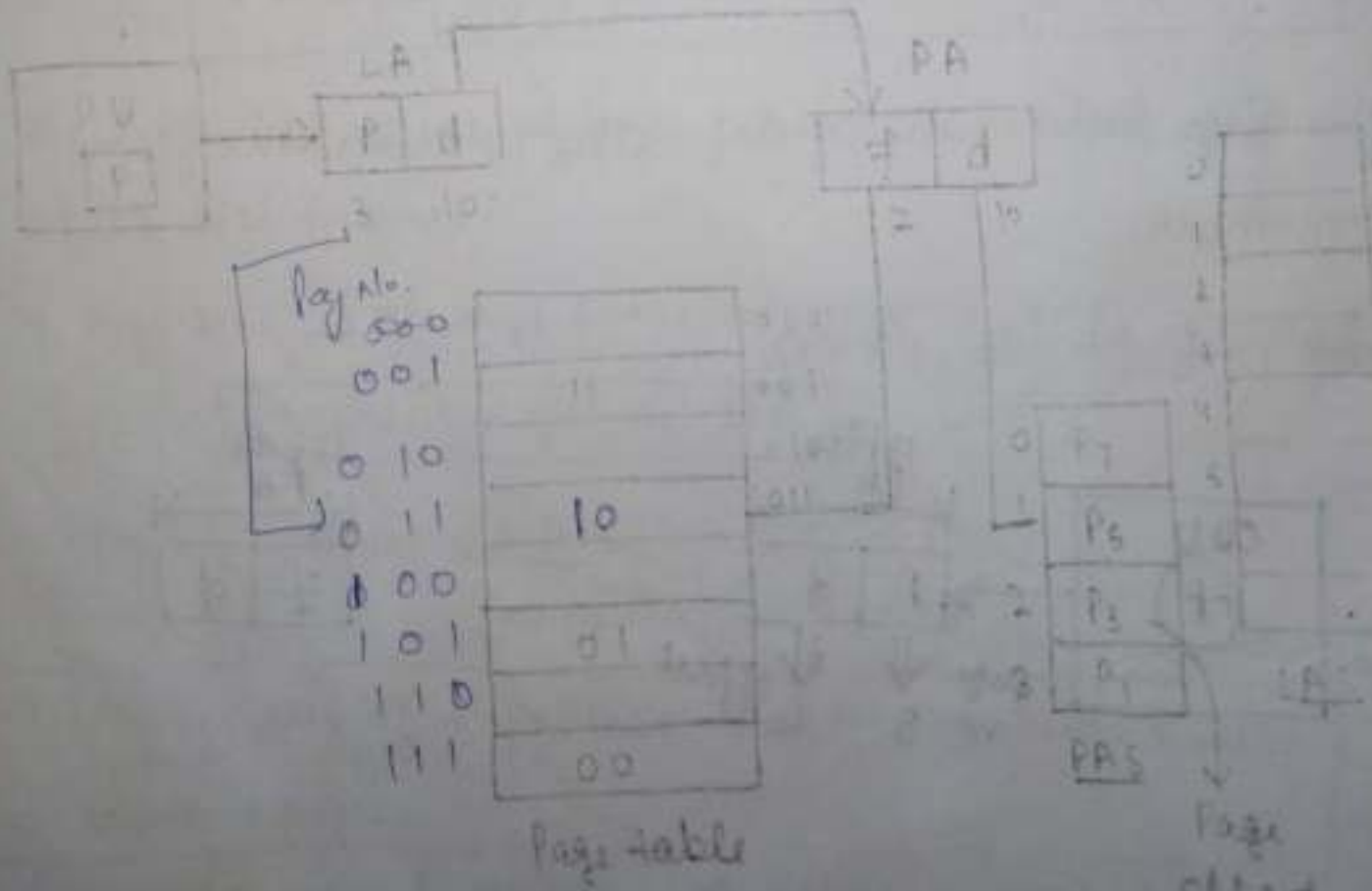
→ No. of bits require to represent pages of LAS or page number.

d → No of bits require to represent each word of page or page offset.

f → No of bits require to represent frames of P.AS or frame Number.

d → No of bits require to represent, word number of the frame or frame offset.

Page offset and frame offset are same.



Example

Let LA = 13 bit , PA = 12 bit

here firstly we have LA in which P = 3, d = 10

from P = 3, we will find Page No. and from d = 10 we can find specific word inside that page.

Important Points :-

(1) whenever the process is created, the paging will be applied on the process and base address of page table will be store in PCB.

(2) The paging is with respect to every process and every process have its own page table.

(3) The page table of processes will be store in the main memory (PAS)

(4) There is no external fragmentation in the paging.

(5) The internal fragmentation ~~exists in paging~~ exists in the last page and the internal

fragmentation is considered as $\frac{P}{2}$ where P is page size.

(6) Here external fragmentation is not happening due to **Page size = frame size**

Ques :- Consider a system with logical address LA = 27 bits, P.A = 21 bits, page size = 4KW. Calculate the No. of pages and no. of frames.

Sol^n :- LA = 27 bits $\Rightarrow$ LAS = $2^{27} \Rightarrow 2^7 \cdot 2^{20} \Rightarrow 128$ MW

PA = 21 bits $\Rightarrow$ PAS = $2^{21} \Rightarrow 2^{20} \cdot 2 \Rightarrow 2$ MW

No. of pages = $\frac{128\text{ MW}}{4\text{ KW}} = \frac{128 \times 2^{20}}{2^2 \times 2^{10}} = 2^5 \times 2^{10}$

$= 32$ k pages.

No. of frames = $\frac{2\text{ MW}}{4\text{ KW}} = \frac{2 \cdot 2^{20}}{2^2 \cdot 2^{10}} = 2^9 = 512$ frames.

Performance of paging :-

The main memory excess time = m

the page table are stored in main memory.

Two memory accesses are needed to access a byte (one for page table entry, one for the byte). Thus memory access is slowed by a factor 2.

So, effective access time = 2M (EMAT)

Because, we access page table + P.A.S
(table is stored in MM) → to access actual address

// So, we access MM two times.

So, there is a delay

// Solution

Solution to this problem is use of special, small, fast-look up hardware cache, called as associative registers or translation look aside buffers (TLBs)

→ Each register consists of two parts - a key and a value. When one item is to be searched, it is compressed with all keys simultaneously

→ Search is fast.

→ Hardware is expensive.

→ TLB improve performance of paging.

→ TLB (associative registers) access time is very - very less as compared to main memory access time.

→ TLB contain frequently referred page no. and corresponding frame numbers.

→ TLB is placed before the page table.

→ If CPU generate a logical address, the page number is presented to TLB that contains page No. and corresponding frame No. If page no is found in TLB its frame no. is immediatly available and is used to access memory.

If page number is not found in the TLB, then page no. is searched in the page table. Then frame no can be obtained from there to

access physical memory.

⇒ If TLB is already full of entries, the O.S must select one for replacement.

⇒ Everytime a new page table is selected, the TLB must be flushed (erased)

⇒ The percentage of times that a page no. is found is associative Registers (TLB's) is called the hit ratio.

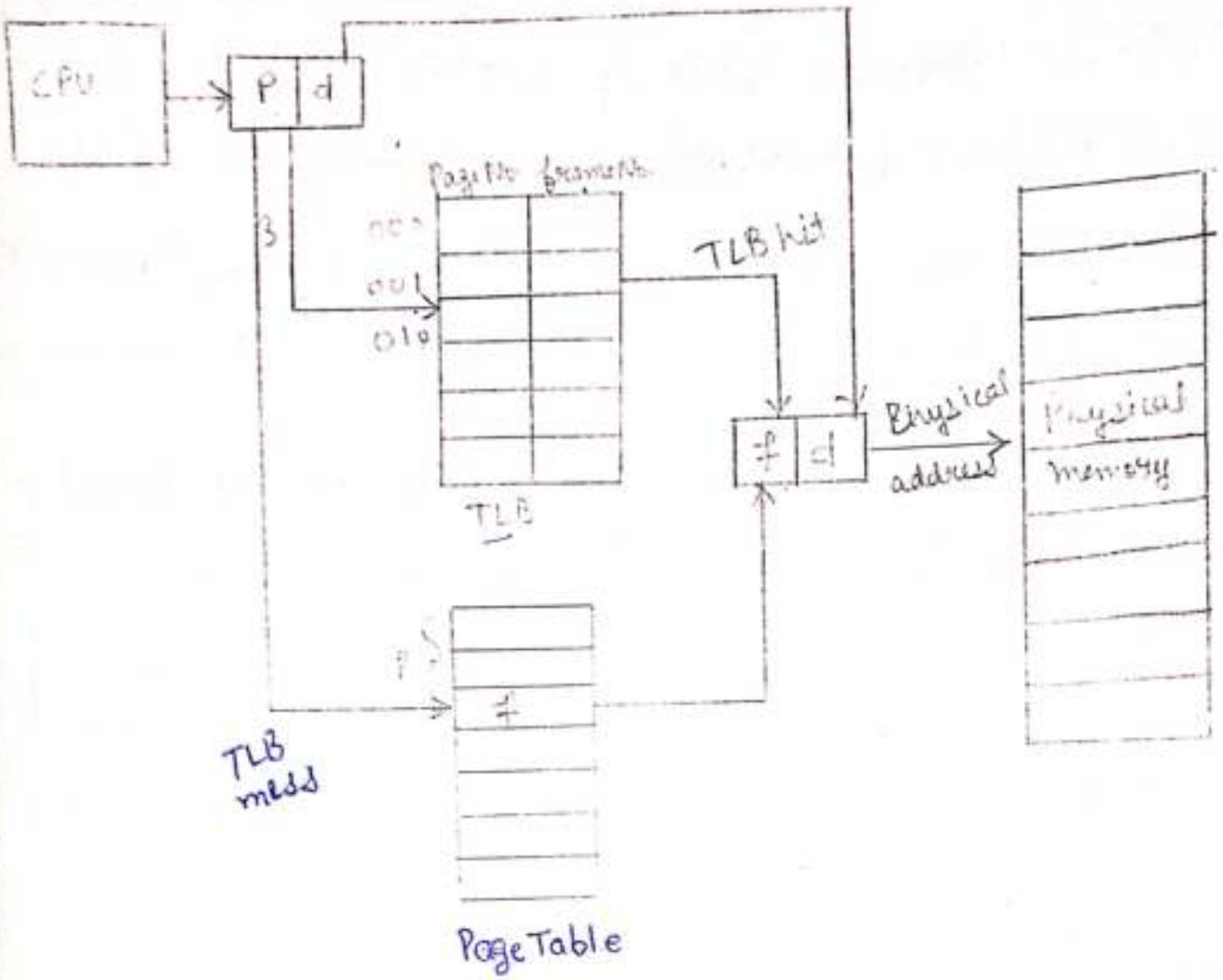
If the TLB access time = c

the TLB hit ratio = x

$$EMAT = x(c+m) + (1-x)(c+2m)$$

- EMAT: effective memory access time
- $x(c+m)$: If TLB hit
- $(1-x)(c+2m)$: If TLB miss

⇒ First CPU will check for frequently refered pages in TLB if page no. found in TLB (TLB hit), otherwise CPU check for main memory (TLB miss)



(Paging hardware with TLB)

Ques: Consider a system MM access time = 100 ns

TLB access time = 2ns

TLB hit ratio = 95%. then what is effective memory. Access time (EMAT) with TLB and without TLB.

Sol^{n}:- Without TLB = 2* Memory access time

= 2 * 100 = 200 ns

With TLB = Hit ratio → $x(c+m) + (1-x)(c+2m)$

↑ TLB access time

$$= \frac{95}{100}(2+100) + \left(1-\frac{95}{100}\right)(2+200)$$

$$= \frac{95}{100} \times 102 + \frac{5}{100} \times 202$$

$$= 107 \text{ ns}$$

Segmentation :-

→ Paging don't follow user view of memory allocation.

→ Paging don't bother, where function is starting or ending, it simply divides memory into pages.

→ To achieve user view of memory allocation, the segmentation will be implemented.

(User view of a program)

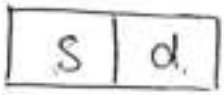
→ In segmentation, LAS will be divided into various segments.

→ The segments of LAS will vary in the size.

→ The segments of LAS will be brought into PAS.

→ The no. of entries in segment table is equal to the number of segments in LAS.

→ Logical address (LA) contains



segment no. → used as an index in segment table.

offset → must be b/w 0 and segment limit.

→ Segment table has a segment base and a segment limit.

segment base → Contains starting physical address when segment resides in memory.

segment limit → Specifies length of segment

→ d must be ≤ L

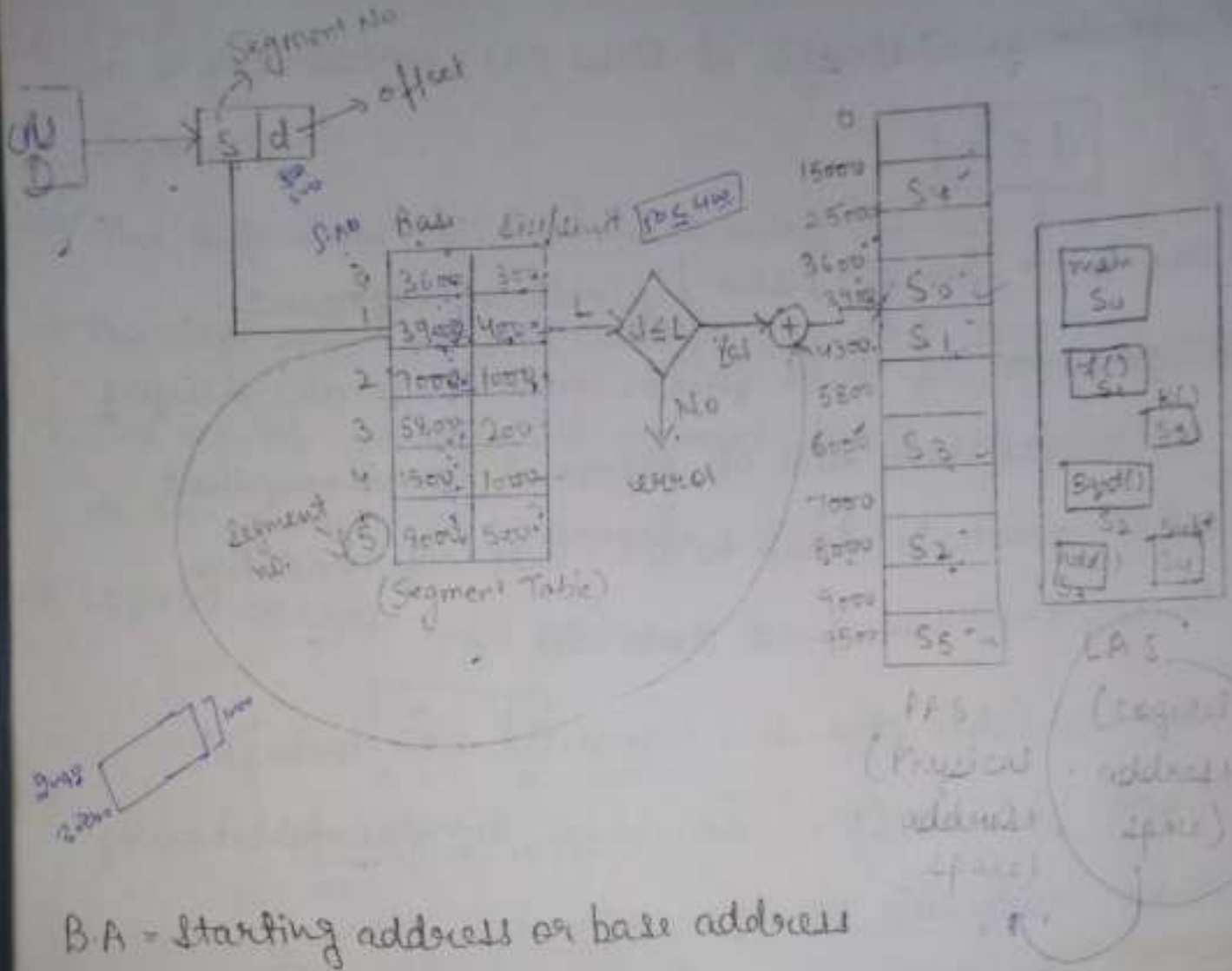
$$\boxed{d \leq L}$$

here L → length / size / limit of segment

if this fails, means we are trying to access data out of segment boundary. that will generate trap to OS. But if condition true, then we find data in PAS.

The variable size segments are brought from LAS to PAS. So, it is similar to behaving like "variable partition scheme". Hence, segmentation still suffer from the external fragmentation.

$$\boxed{s \mid d}$$, here S is particular accessing segment number. through this segment no we find B.A and size of segment.



B.A = Starting address or base address

S = ~~Segment no.~~

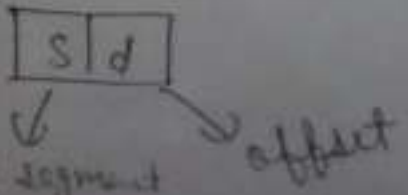
No of bits require to represent segment no.

d = No of bits require to represent segment size or word no of segment or segment offset

size = length or limit of segment

S.No ≈ Segment number.

→ CPU gives the logical address



→ For each segment, there is entry in the segment table. Entry in the table contains two fields – Base address of segment
– Limit of segment.

→ After obtaining the base address and limit in the segment table firstly the segment limit is checked with offset (d) at LA. If $\boxed{d \leq L}$ only then can say that LA is varied otherwise it will not a legal address because if is out of the limit (and we will send trap to o.s)

→ After that if $d \leq L$ true the B.A is added to limit and memory is allocated in PAS.

→ eg. for segmentation No. - 1

Base address = 3900
Limit = 400
4300

(memory in PAS is from 3900 to 4300)

(combination of Paging and segmentation:-

Paging and segmentation can be combined in two ways:-

① Segmented Paging (large segment is divided in pages)

3) Paging on segment table (large segment table is divided into pages)

2) Segmentation Paging :-

→ To avoid, loading large size segment into memory, we apply segment paging. Instead of loading entire segment into memory, we load some of pages of segment.

→ In segmented paging, paging will be application the segments. and the page of segment will be brought into the memory instead of the entire large segment

→ **Example**

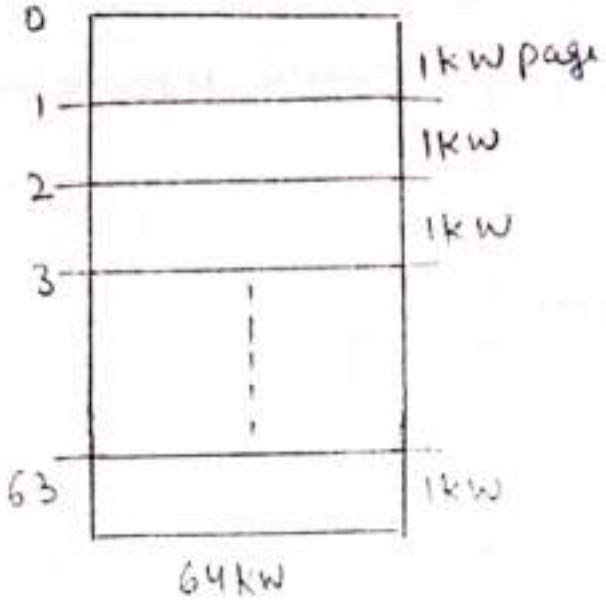
Segment size = 2^{16} = 64 KW
If page size = 1 KW

given | assume

then no. of pages = $\frac{64\ KW}{1\ KW}$ = **64 pages**

LA = 34 bits

given / assume



So, here segment size = 64KW

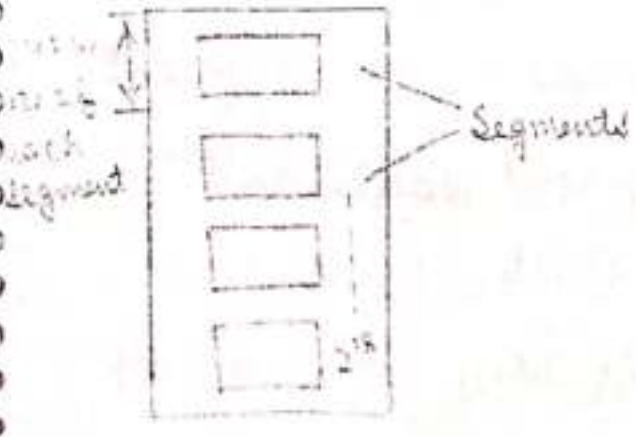
L.A = 34 bits

L.A.S = 2^{34} W

No. of segments = $\frac{2^{34}\text{ W}}{64\text{KW}} = \frac{2^{34}}{2^6 \cdot 2^{10}} = 2^{34-16} = 2^{18}$

64KW: Size of each segment

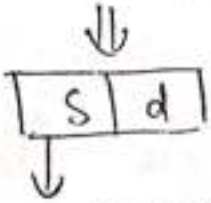
So, LAS is divided into 2^{18} segments and each segment has size = 64KW



In segmentation LAS is divided into segments)

L.A = 34 bits

So, In the L.A using segmentation.

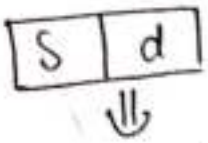


Segment no. or no. of bits required to represent segmentation

S = 18 bits

If total segment = 2^{18}

No. of bits require to represent segmentation = 18



word no. of segment
or offset
or No. of bits require
to represent word no of segment

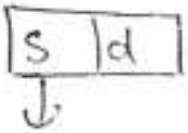
Segment size = 2^{16} W (64 KW)

So, bits require to represent segment word = 16 bits

d = 16 bits

→ For each segment there is a entry in segment table. So, there will be in total 2^{18} entries in segment table. (2^{18} segments)

→ LA will given



Segment No.

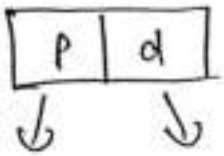
→ Using segment No. base address of the segment can be obtained from segment table and the limit field is used to check if whether

d ≤ L

if true then it is a valid else it is not a valid L.A. and trap is send to O.S.

⇒ If condition gets true. then this 16-bits d i.e offset is again divided into parts.

⇒



page No. — word # inside page or offset

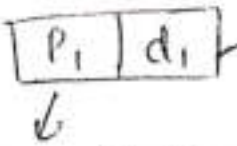
Segment is divided in equal page size. So, now we will go to a particular page and then on the particular word inside that page.

No. of pages inside segment = 64 pages

$$\frac{\text{(size of segment)}}{\text{size of each page}} = \frac{2^{16}}{1\text{KW}} = 2^6 = 64 \text{ pages}$$

So, Bits require to represent each page = 6

i.e.

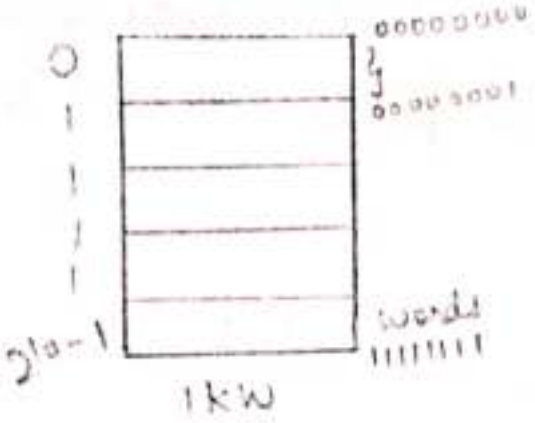


6 bits (bits require to represent page or page no.)

Offset $\xrightarrow{d_1}$ word no. inside page or bits require to represent each word inside page.

So, page structure = 1 KW

(size of each page)

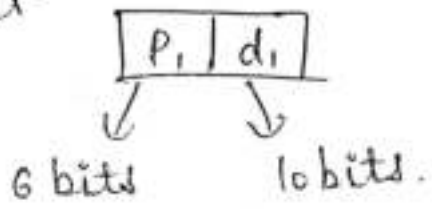


Bits required to represent each word = 10 bits

So,

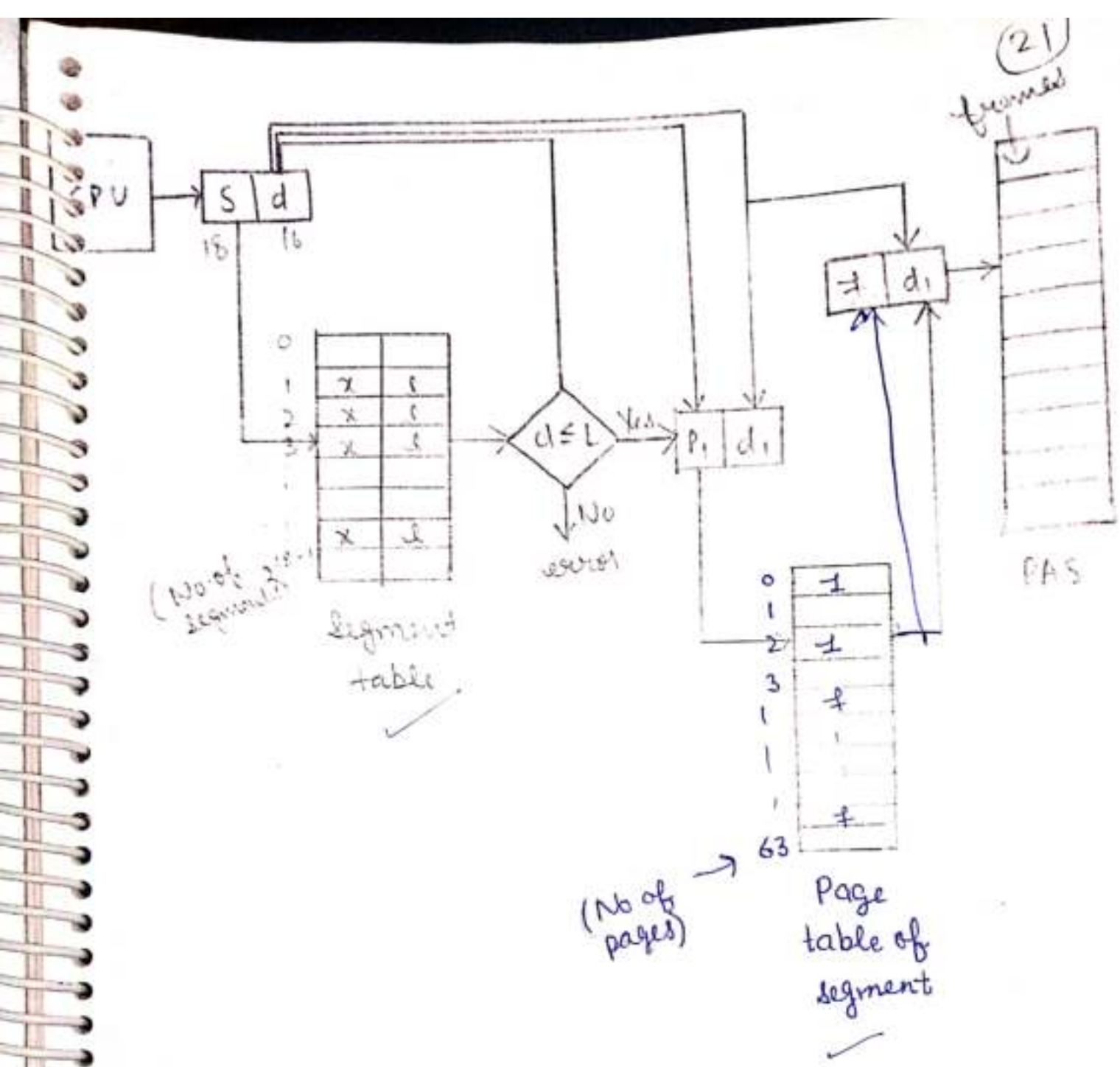
→ After limit is checked with d, i.e. $\boxed{d \leq l}$

we get



→ So, now for each page will have entry in page table and we will have entries in page table = No. of pages.

→ And the page table contains the frame no. and then we can get the physical address PA by using $\boxed{f \mid d}$ in the P.A.S (physical address space.)



UNIT- III

Virtual memory and Paging

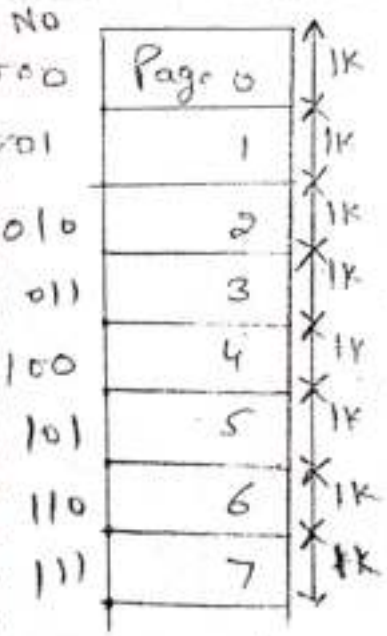
Virtual memory is a concept that the user has illusion that he has large working space at its disposal but it is not so.

In virtual memory, Logical address is divided into fixed size Pages and the Physical memory is divided into frames

Paging is a fixed size memory allocation technique

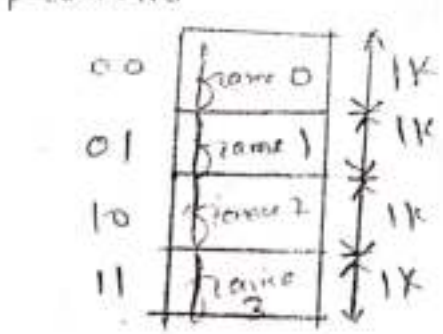
Size of Page = Size of frame

The displacement same.



Virtual memory = 8K

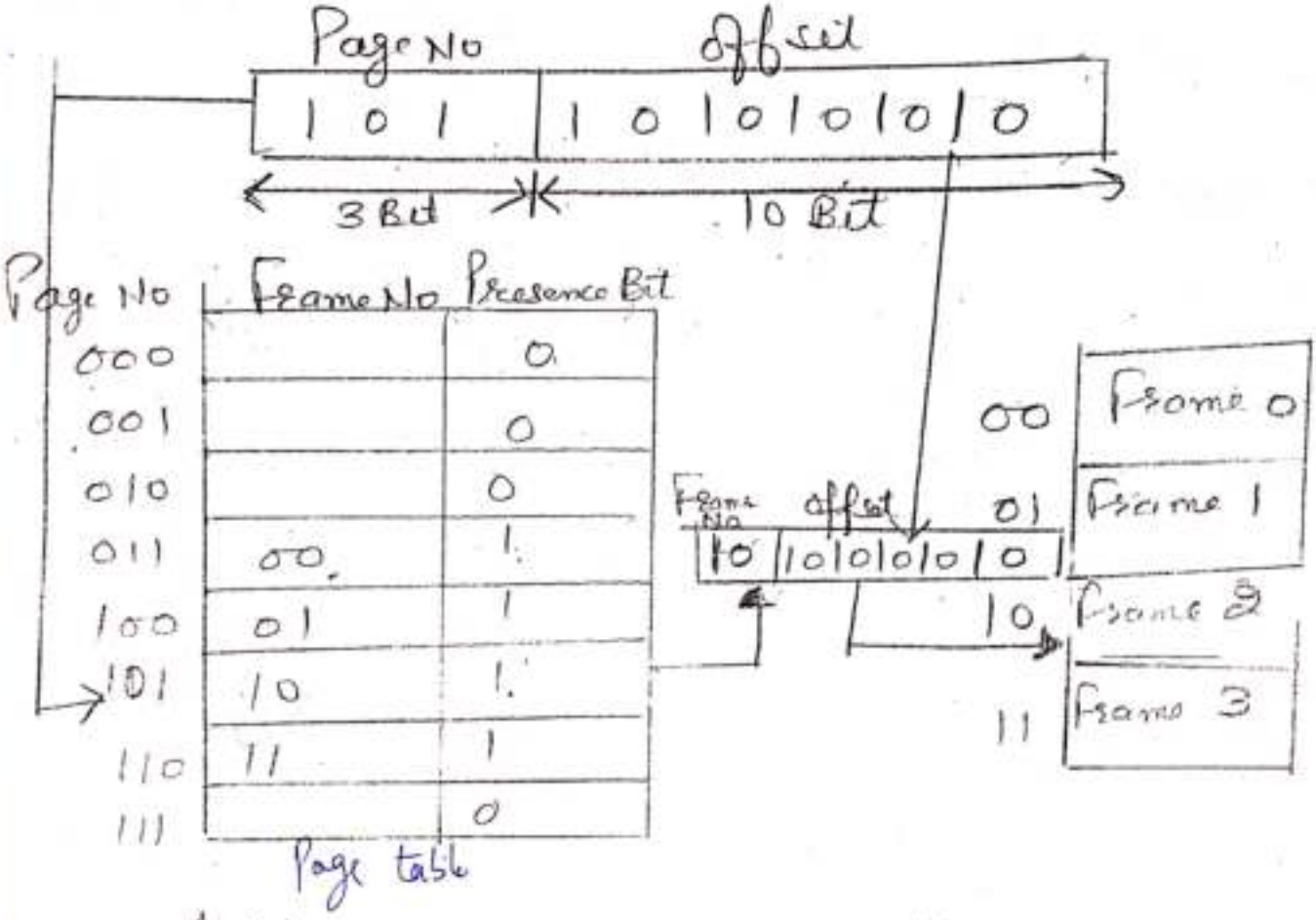
No of bits required = 13



Physical memory = 4K

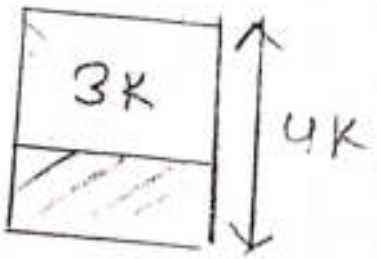
No of bits required = 12

Address mapping → We need to convert the 13 bit logical address into 12 bit physical address.



Address mapping in Virtual memory

Paging suffers from the Internal fragmentation



Eg
A process P_1 is allocated P_1 4K of space & it requires 3K of space then 1K of space will be wasted by process P_1 itself in its address space.

This is the concept of

TLB (Translation Lookaside Buffer)

TLB is a small, fast associative (per) Cache memory.

Page table which was initially stored the memory required a lot of time so Page table was kept in a cache call TLB. If TLB hit occurs then the data will be fetched from memory and if TLB miss occurs then the Page table in the main memory will be accessed & then data will be fetched.

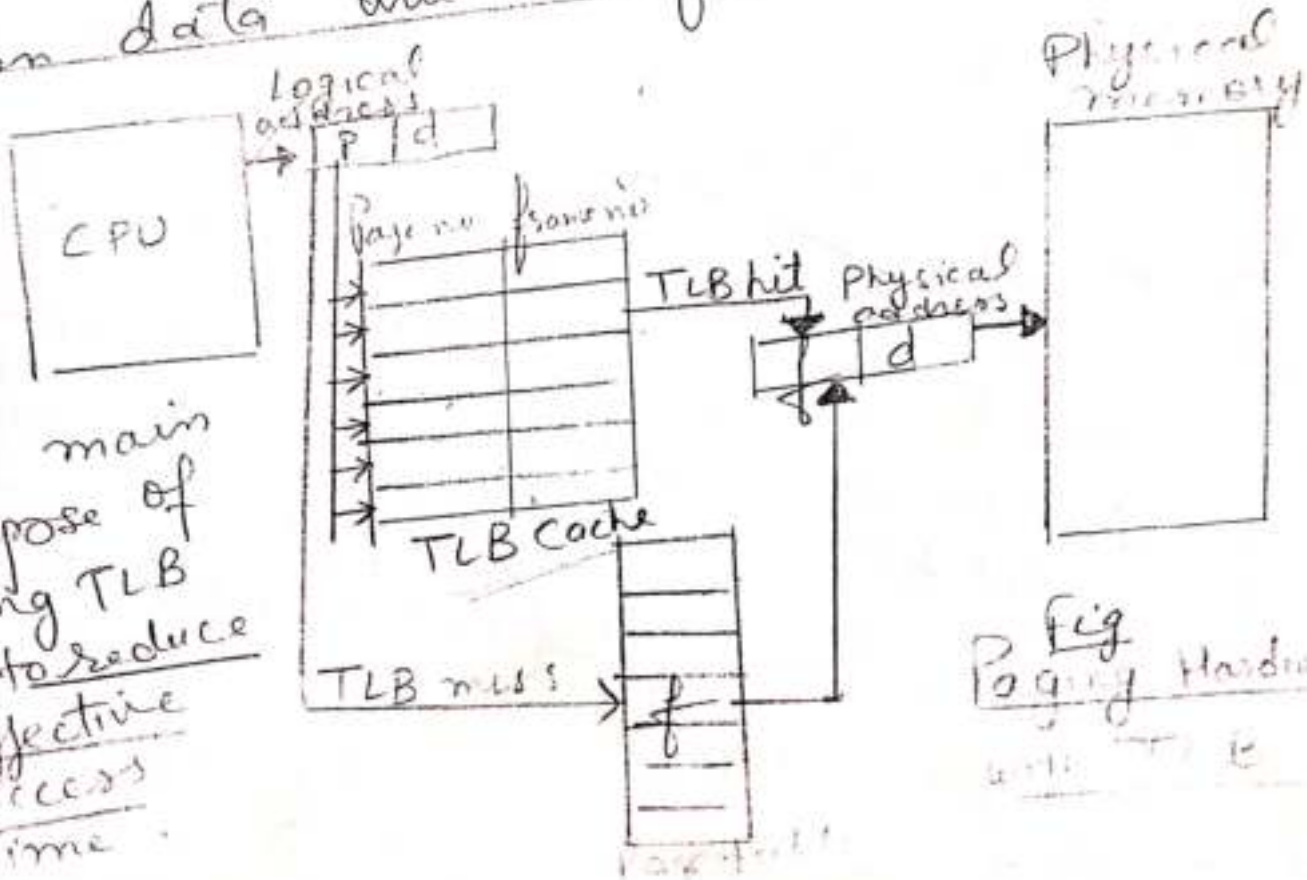


Fig Paging Hardware

The main purpose of using TLB is to reduce effective access time.

Numerical on TLB

Q1 80% hit ratio in TLB. If it takes 20 nanosecond to search the TLB and 100 nanoseconds to search the memory. What is effective memory access time?

Sol^n TLB hit: Data fetch from main memory; TLB miss: Page table in main memory; Data in main memory

$$\Rightarrow \underbrace{\frac{80}{100}(20+100)}_{\text{TLB hit}} + \underbrace{\frac{20}{100}(20+100+100)}_{\text{TLB miss}}$$

$$= \frac{80}{100} \times 120 + \frac{20}{100} \times 220$$

$$= 96 + 44 = 140 \text{ nanoseconds}$$ Ans

Q2 Consider Paging H/w with TLB. Assume page table & Pages are in Physical memory. If it takes 10 ms to search TLB & 80 ms to search memory. If TLB hit is 0.6 what is effective memory access time

Sol^n TLB hit; TLB miss

$$\frac{60}{100}(10+80) + \frac{40}{100}(10+80+80)$$

$$\frac{60}{100} \times 90 + \frac{40}{100} \times 170 = 54 + 68$$

$$= 122 \text{ ms}$$

The Page is not found in the Physical memory then it is Page fault and then have to bring Page from virtual memory and replace it with Physical memory already existing Page.

Page Replacement policies

FIFO (First in first out) Page Replacement

Replace the oldest Page

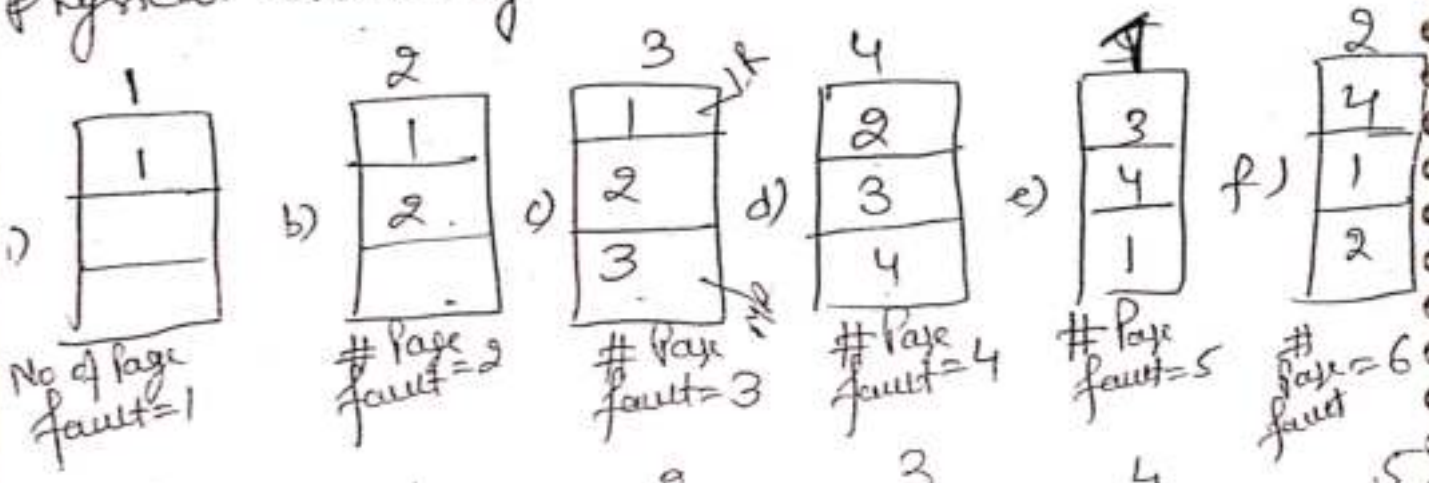
Demand Paging → Initially Physical memory is empty, we bring the Pages in the Physical memory according to the demands the Process. This is called Demand Paging.

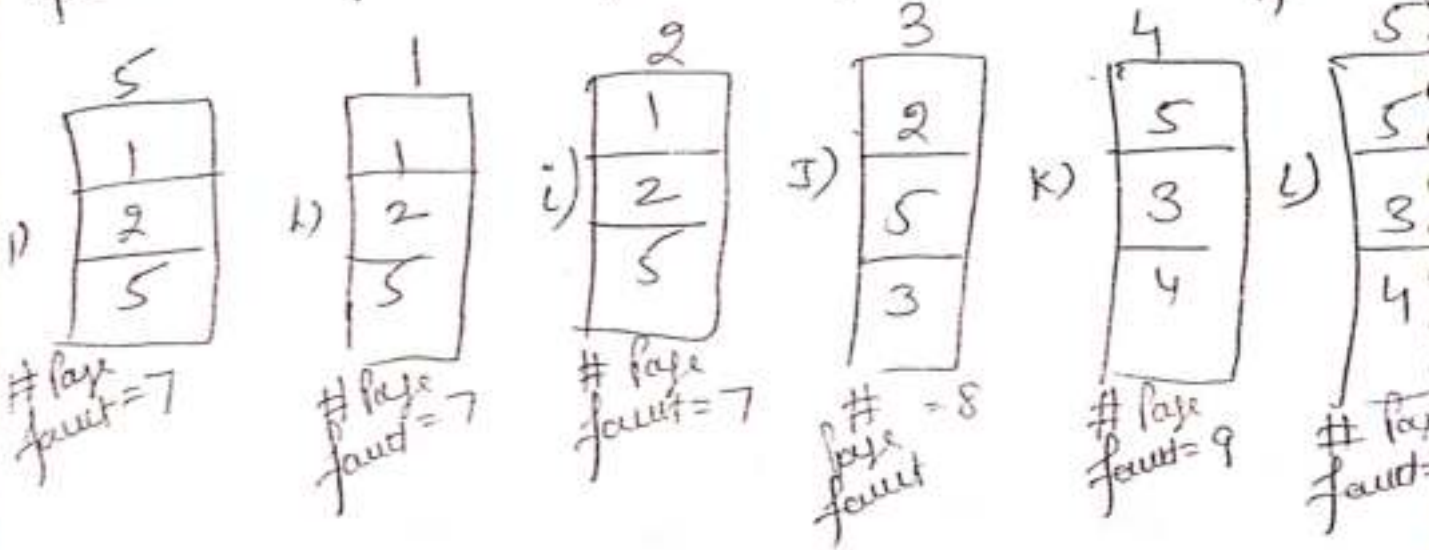
Example of FIFO → Find No of page fault.

for the FIFO page replacement for following demand

→ 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Assume that 3 Page frames are available Physical memory

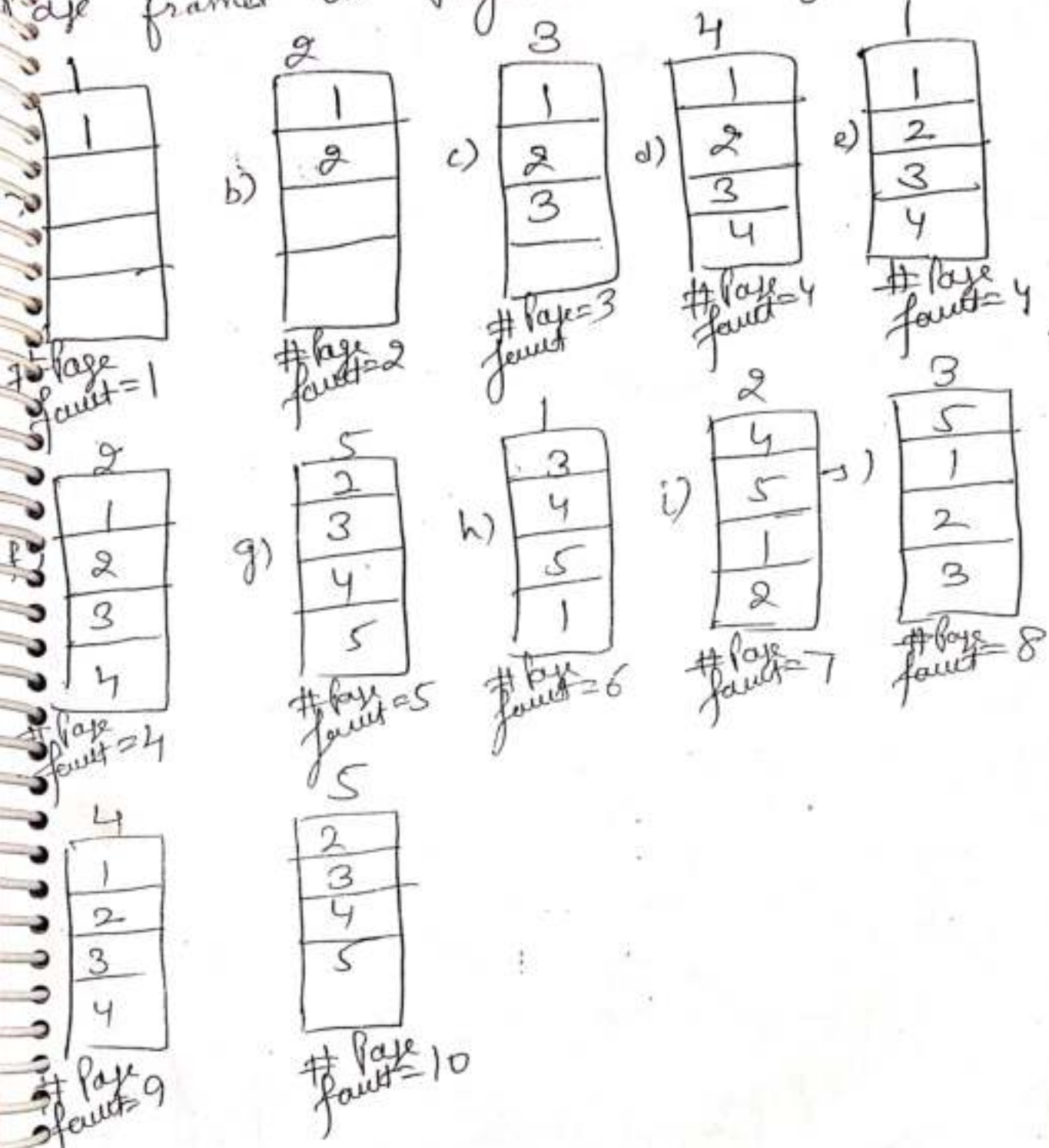




∴ Total No of Page fault for 3 Page frames = 9

1 page already existing no change

Now For same example we take 4 Page frames in Physical memory



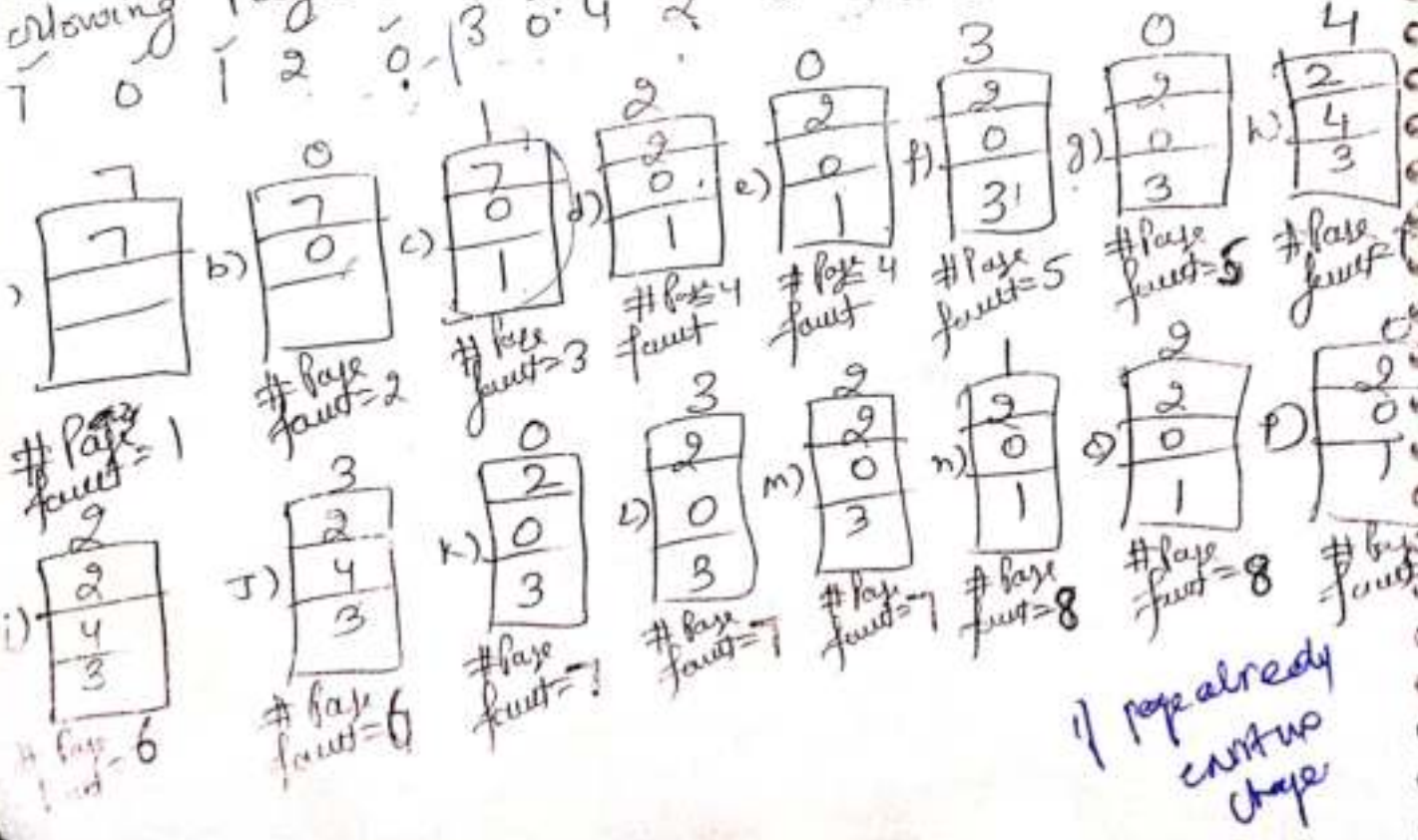
Total No of Page fault for 4 Page frames = 10

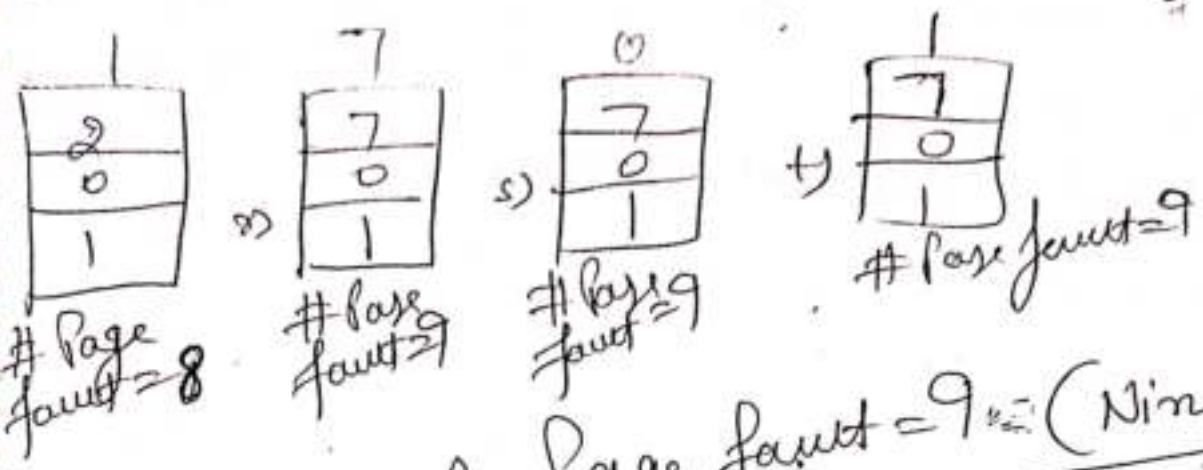
As no of Page frames increase in Physical memory, No of page faults should decrease. But in FIFO the no of page faults are increasing. This Unexpected behavior is called Belady's Anomaly. (Anomaly)

Optimal Page Replacement

Replace the page that will not be used for the Longest period of time in future. It is optimal Page Replacement policy as there will be minimum page faults. Consider the following Page demand & assume 3 page frames

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

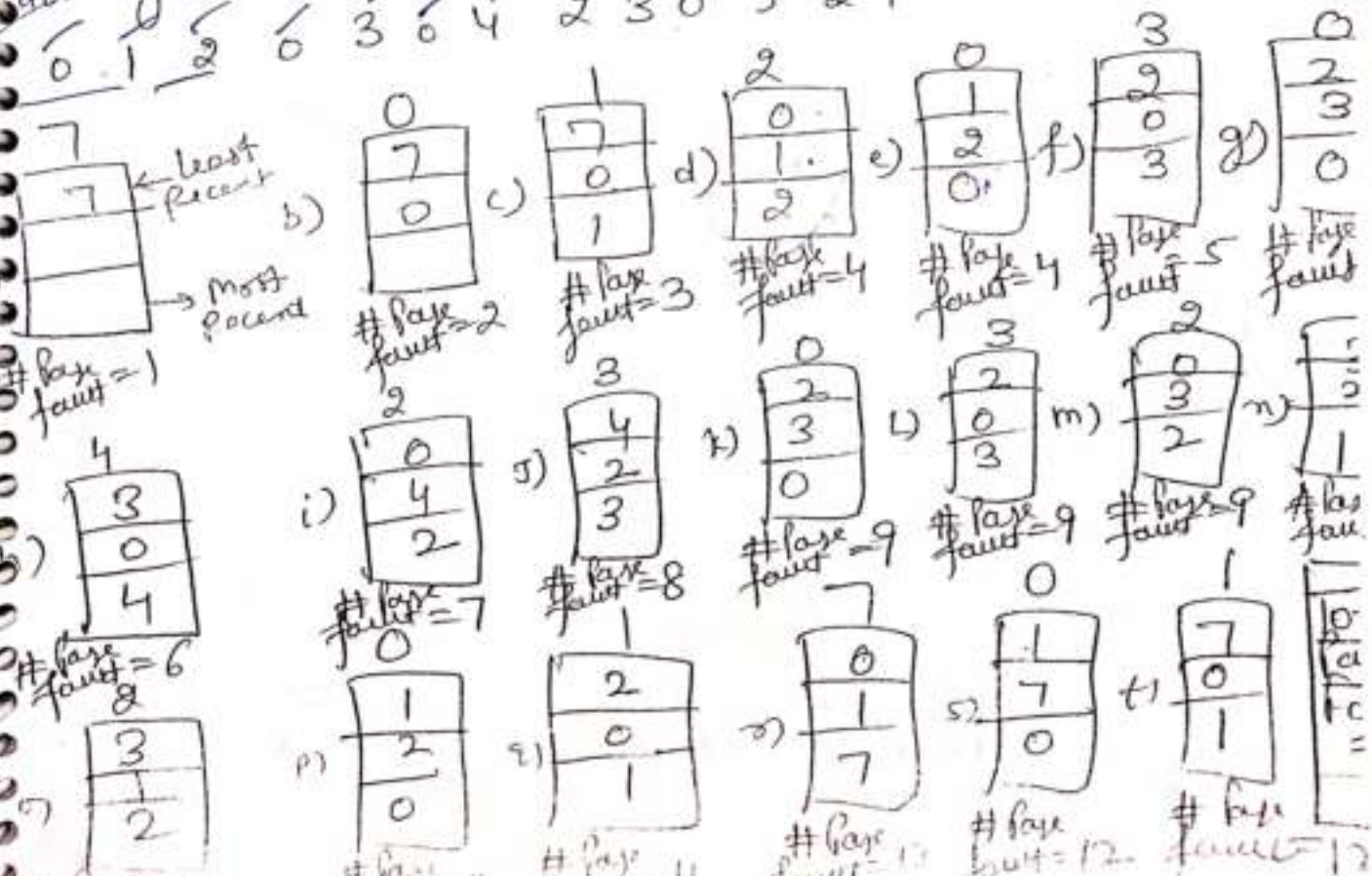




Total No of Page fault = 9 = (Nine)

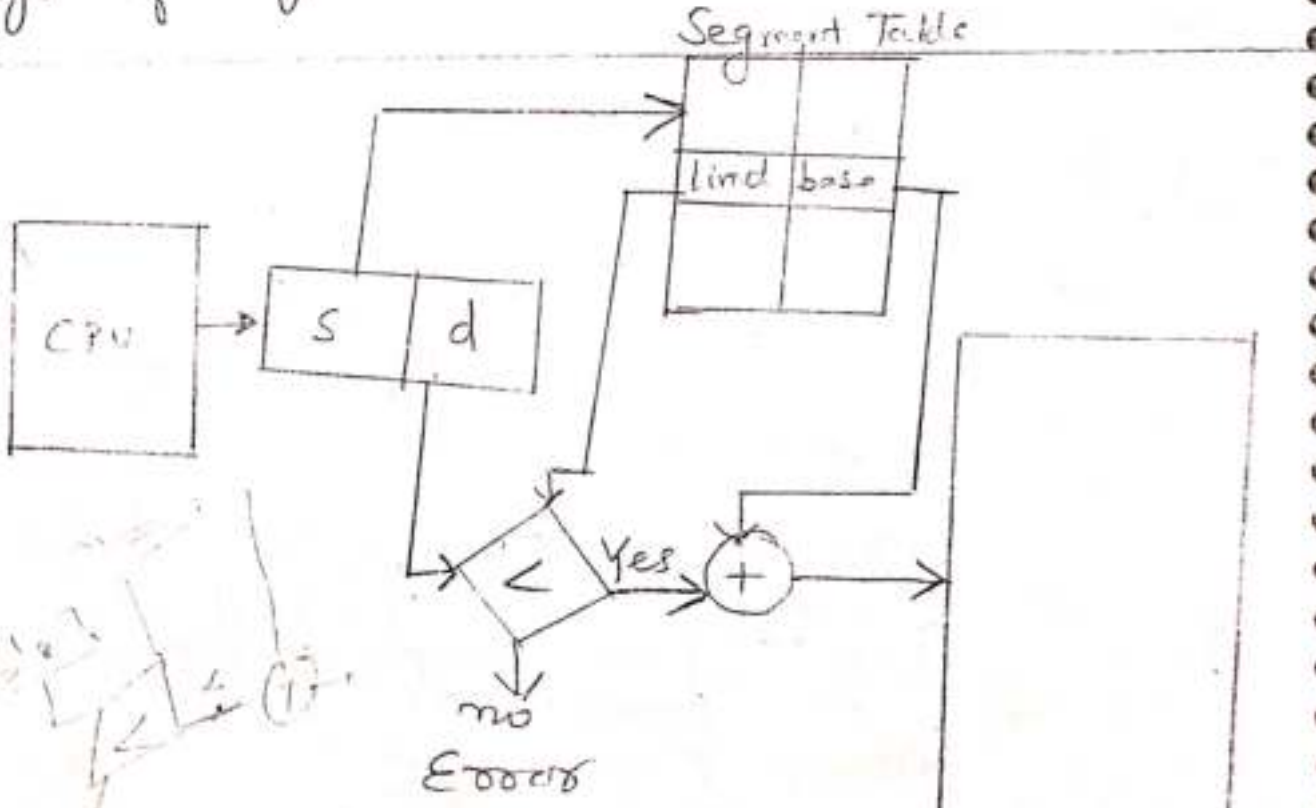
③ LRV (Least Recent use) → Replace the Page that has not been used for the longest Period of time. Consider the following Page demand & assume 3 Page frames

0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1.



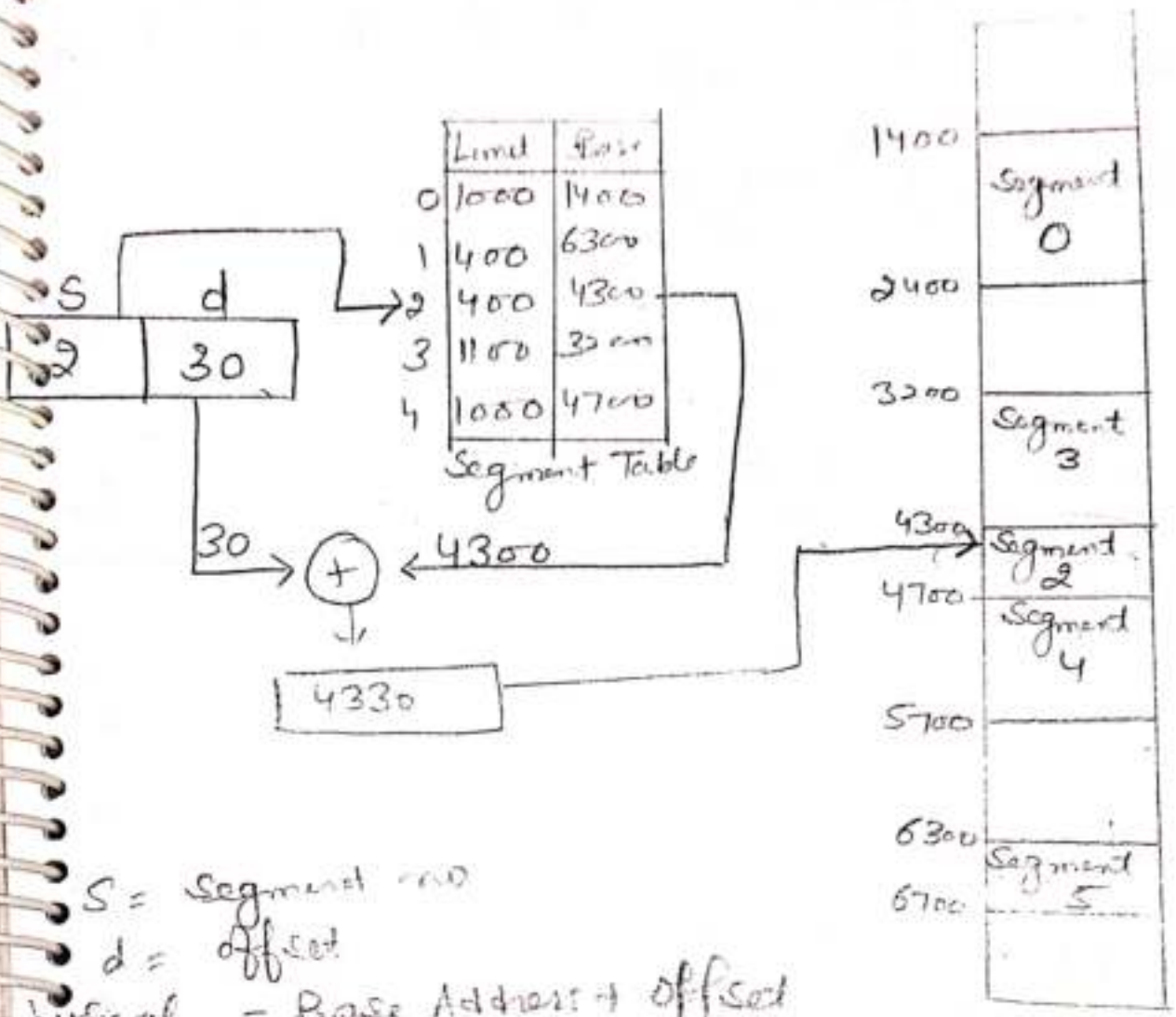
Segmentation - It is a variable size memory allocation scheme. It is a user's point of view in which every Segment is assumed to be of different size.

CPU generates Segno and Offset. Then in Segment table we find corresponding Base address of Segment if offset is less than Length of Segment and then add offset to Base Address to get Physical address. If offset is greater than the length of Segment, then there is error.



Segmentation Hardware

Physical memory



S = Segment no
d = offset
Physical address = Base Address + Offset

Numerical on Segmentation

Consider the following Segment table

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are Physical addresses for following logical Addresses

a) 0, 430

Segment No = 0
Offset = 430
Limit = 600

As offset < limit

∴ Physical address = Base Address + offset = 219 + 430
= 649 Ans

b) 1, 10

Segment No = 1
Offset = 10
Limit = 14

As offset < limit

∴ Physical Address = B.A + offset = 2300 + 10
= 2310 Ans

c) 2, 500

Segment No = 2
Offset = 500
Limit = 100

As offset > limit
So Error

d) 3, 400

Segment No = 3
Offset = 400
Limit = 580 As offset < limit

∴ P.A = B.A + offset = 1327 + 400
= 1727 Ans

(e) 4, 112

Segment No = 4
Offset = 112
Limit = 96
As offset > limit
∴ Error.

Local vs Global Allocation

Global replacement allows a process to select a replacement frame from set of all frames, even if that frame is currently allocated to some other process i.e one process can take frame from another.

Local replacement requires that each process selects from only its own set of allocated frames.

Thrashing →

If any process that doesnot have "enough" frames. If the process doesnot have minimum no of frames it needs to support pages in active use, it will quickly Page fault. At this

Point of time, it must replace some page. However Since all pages are in active use, it must replace a page that will be needed right away. It quickly faults again & again; replacing the pages it must bring back in immediately. High paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing

Causes of Thrashing

Consider the following scenario

The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement is used, it replaces the pages regardless of the processes to which it belong. Now suppose if a process enters a new phase in its execution & needs more frames. It starts faulting and taking frames from other processes. These processes need those pages however, and so they also fault, taking frames from other processes. These faulting processes need paging device to swap in and out the pages. As

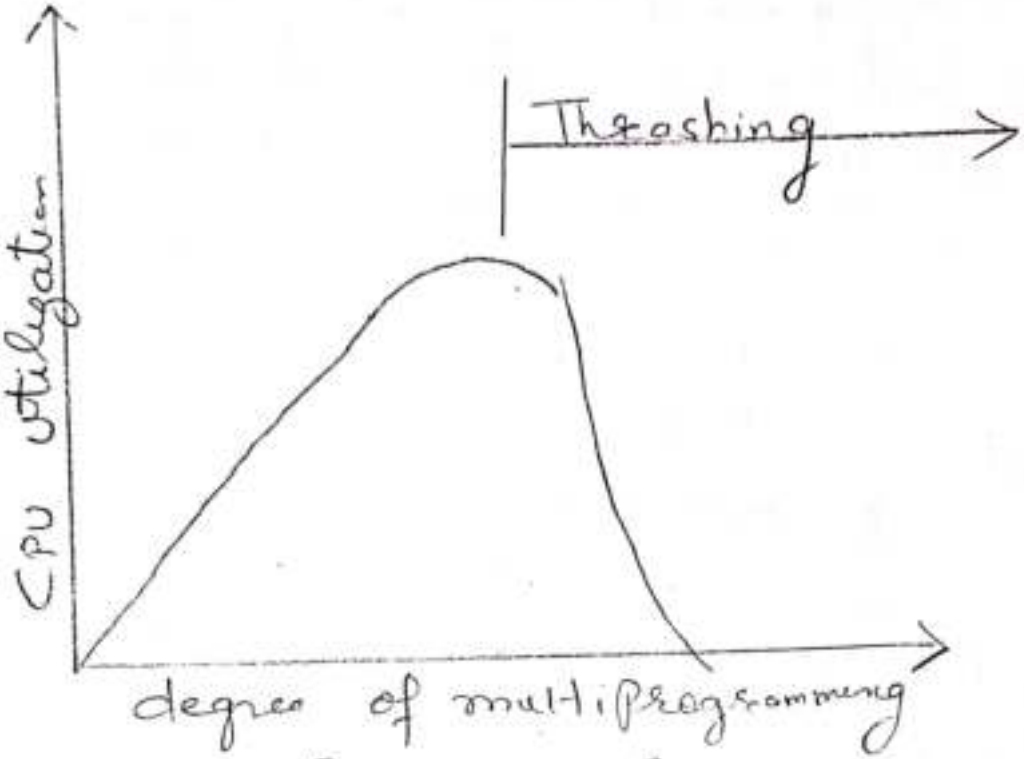


Fig:- Thrashing

they queue up for paging device, the ready que empties. ∴ CPU utilization decreases.
The CPU scheduler sees the CPU utilization to be low and increases degree of multiprogramming the new process tries to get started by taking frames from running processes, causing more page faults. As a result, CPU utilization drops even further & CPU scheduler tries to increase degree multiprogramming ∴ Thrashing has occured & system throughput plunges. No work is getting done, because processes are spending time in paging.

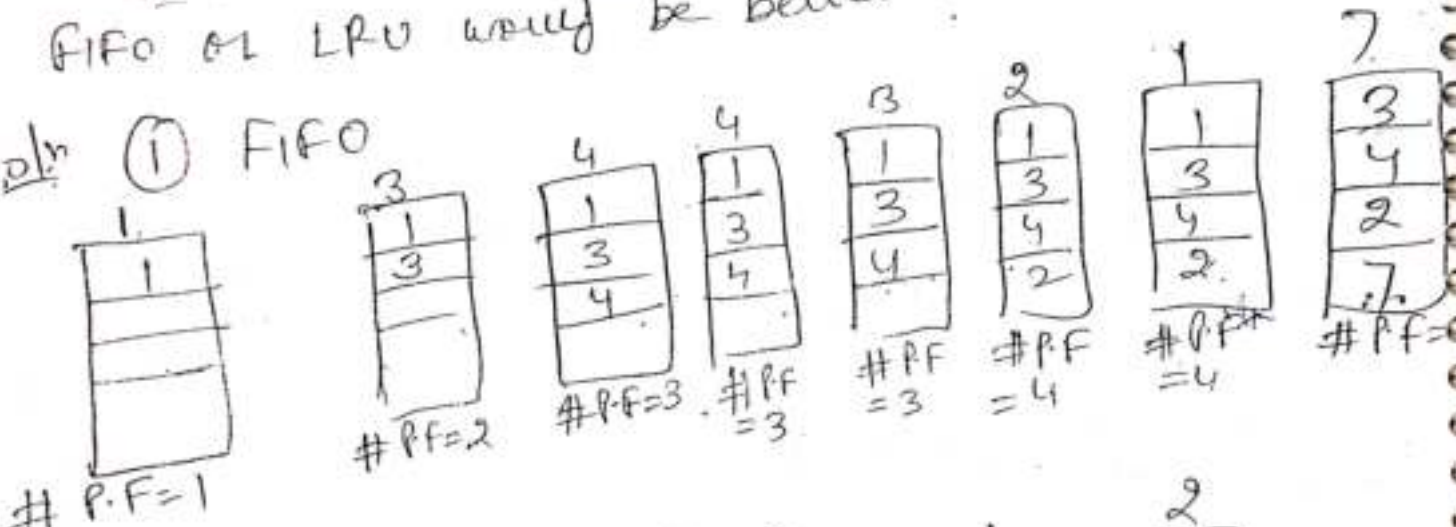
We can limit the effects of thrashing by using local page Replacement policy. With local page Replacement Policy if one process starts thrashing, it can't steal frames from another process and cause the latter to thrash as well.

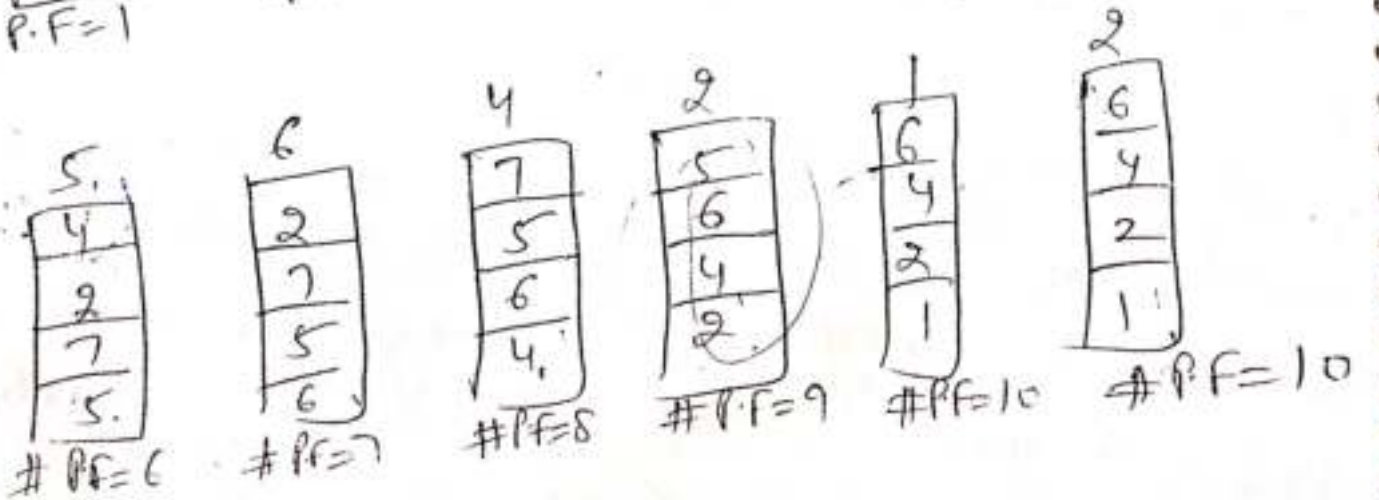
Numerical

2010 Question Paper

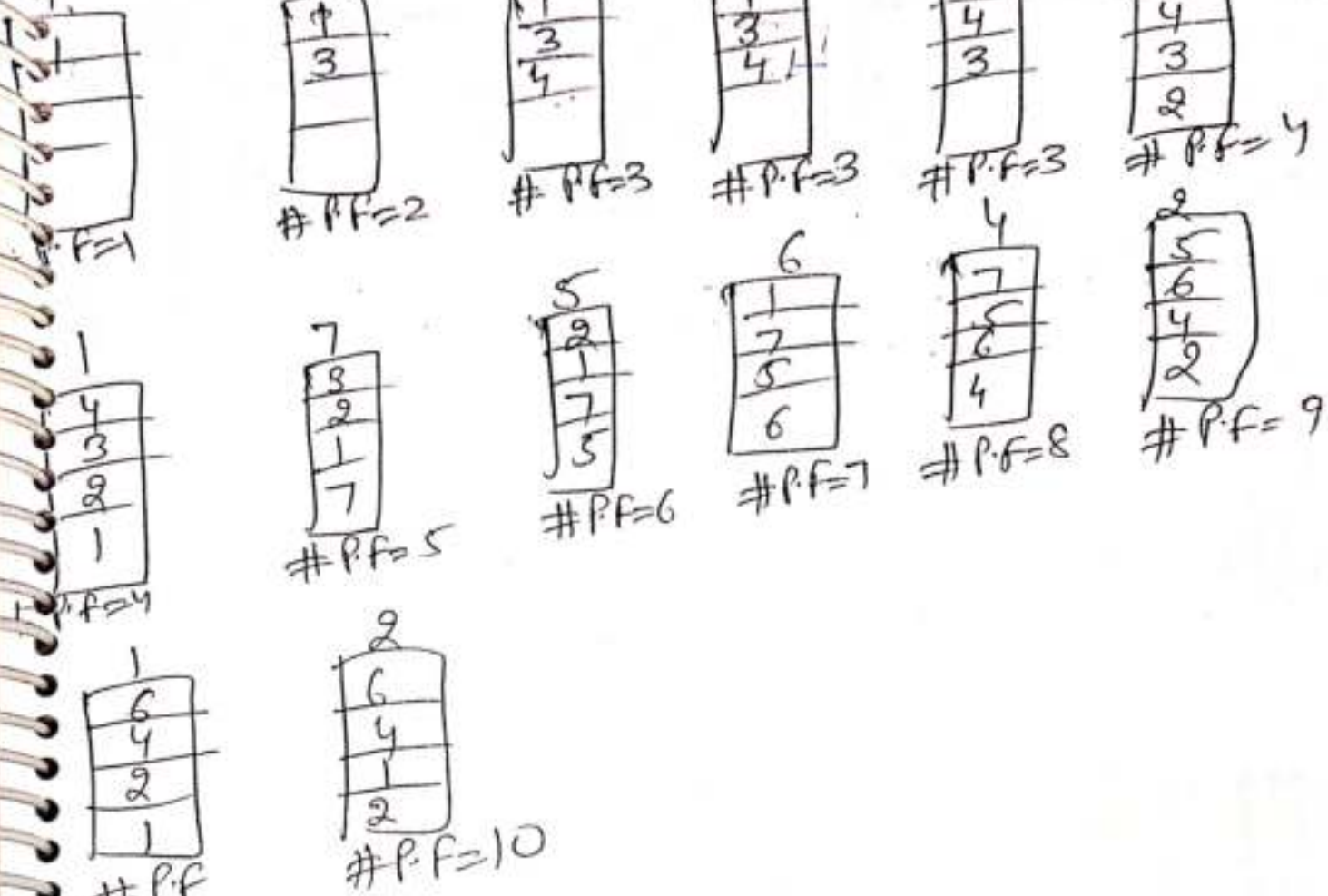
Consider main memory with capacity of 4 Page frames. Reference string is 1, 3, 4, 4, 3, 2, 1, 7, 5, 6, 4, 2, 1, 2 which of FIFO or LRU would be better?

Soln (1) FIFO





No of Page fault = 10



No of Page fault = 10

...ol all memory accesses. Fig. 6.60 illustrates how the TLB supports the 80386DX paging ...hanism.

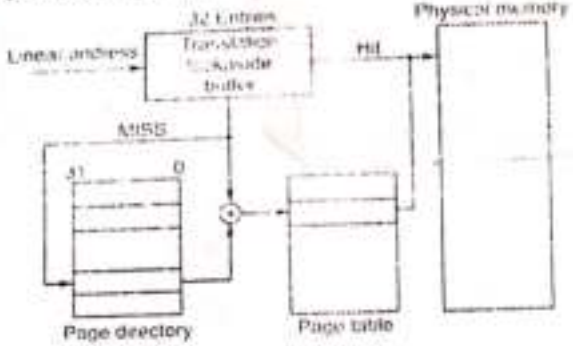


Fig. 6.60. Translation lookaside buffer.

6.8.2.4. Paging Operation

The paging mechanism receives a 32-bit linear address from the segmentation unit. The upper 20-bits of linear address are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 80386DX reads the appropriate Page Directory Entry. If $P = 1$ on the Page Directory Entry indicating that the page table is in memory, then the 80386 reads the appropriate Page Table Entry and set the Access bit. If $P = 1$ on the Page Table Entry indicating that the page is in memory, the updates the Access and dirty bits as needed and fetch the operand. Then 80386DX stores the upper 20-bits of the linear address, read from the page table in the TLB for future accesses. However, if $P = 0$ for either the Page Directory Entry or the Page Table Entry, then the 80386DX generates a page fault, an exception 14.

6.8.3. Demand Paging

Demand paging allows system to create a virtual environment for their programs neither the programmer writing it needs to know how much RAM is really available in the system or where it is located. If a program makes reference which is not in the memory, page fault handler is called. Using this page default handler routine, it then retrieves the data from secondary storage (such as disk) and places it in memory. The previous contents of memory are swapped with data from the disk. In this way, it is possible to create an impression of a system with huge amount of main memory. Its actual size and its location are never known to the program or the programmer, but everything runs as desired.

A demand paging system is similar to a paging system with swapping. A swapper never swaps a page into memory unless that page is needed. Since we are now viewing a program as a sequence of pages, rather than one large contiguous address space, the use of the term swap is technically incorrect. A swapper manipulates entire programs, whereas a pager is concerned with the individual pages of a program. Hence term pager is usually used, rather than swapper, in connections with demand paging.

When a program is to be swapped in, the pager guesses which pages will be used before the program is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used any way, decreasing the swap time and the amount of physical memory needed.

With that scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. Fig. 6.61 shows hardware to implement demand paging.

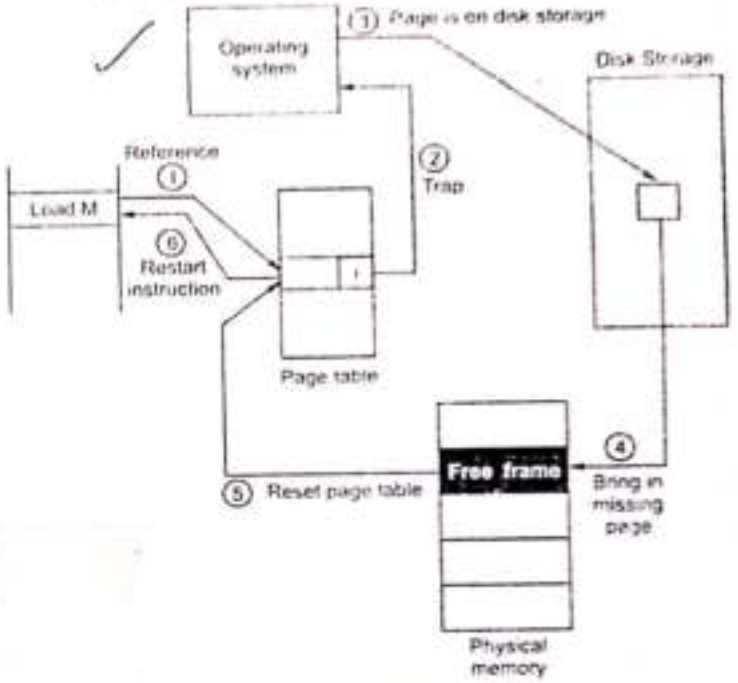


Fig. 6.61. Steps in handling a page fault.

As shown in the Fig. 6.61, the valid-invalid bits are used to distinguish between those pages that are in memory and those pages that are on the disk. When bit is set to "valid", it indicates that the associated page is both legal and in memory. If the bit is set to "invalid", it indicates that the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk. If the process tries to use a page that was not brought into memory, access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory. The procedure for handling this page fault is illustrated in Fig. 6.61. This procedure goes through following steps :

1. It checks an internal table for this program, to determine whether the reference was a valid or invalid memory access.
2. If the reference is invalid, it terminates the process. If it is valid, but the referenced page is not available in the physical memory, trap is activated.
3. It finds a free frame.
4. It schedules a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, it modifies the internal table kept with the program and the page table to indicate that the page is now in memory.
6. It restarts the instruction that was interrupted by the illegal address trap. The program can now access the page as though it had always been in memory.

Use (Sandeep Jain)

It is a collection of related information defined by its creator. File is a sequence of bits, bytes, lines or records whose meaning is defined by its creator.

A file is named and is referred by its name. It has some other properties such as type, time of creation, name of creator, its length & so on.

File can be of many types

→ Text file → It is sequence of characters organized into lines

→ Source file → It is sequence of subroutines and executable statements

→ Object file is sequence of bytes organized into blocks.

→ An executable file is a series of code sections that loader can bring into memory & execute

FILE attributes:

1) Name
2) Identifier
3) Type
4) Location

→ Size
→ Protection
→ Time, date & User Identification.

File operations → File is a abstract data type.

1) Creating a file.
2) writing a file.
3) Reading a file
4) Deleting a file.
5) Truncating a file.
6) Repositioning a file

File Type	Usual Extension	Function
Executable	exe.	Read to run m/c language program
Object	obj, o	Compiled, m/c Lang but not linked
Batch	bat, sh	Commands to Comm Interpreter
archive	zip, avic, tar	related files grouped into one file
multimedia	mpeg,	binary files containing audio a/v Implementation

File Allocation method

An allocation method refers to how disk blocks are allocated for files:

1. Contiguous allocation.
2. Linked allocation.
3. Indexed allocation.

Contiguous Allocation →

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block no) and length (no of blocks) are required

directory

file	Start	Length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Fig Contiguous allocation of disk space

If a file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2, ... b+n-1

] Random access

Problems with contiguous allocation =>

Problem 1

] All these algorithms suffer from External fragmentation

External fragmentation occurs when free space is broken into the chunks and the Largest chunk is insufficient for a request; storage is fragmented into holes and no hole is large enough to store the data.

□ Problem 2

Another problem is determining how much space is needed for a file. How does a creator know the size of file to be created. If we allocate too little space then file can't be (expanded) extended as both sides of file may be in use and too much space allocated will result in wastage of space.

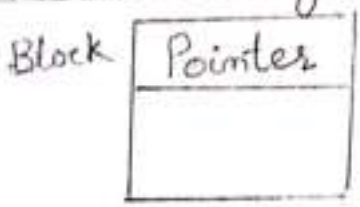
Linked allocation

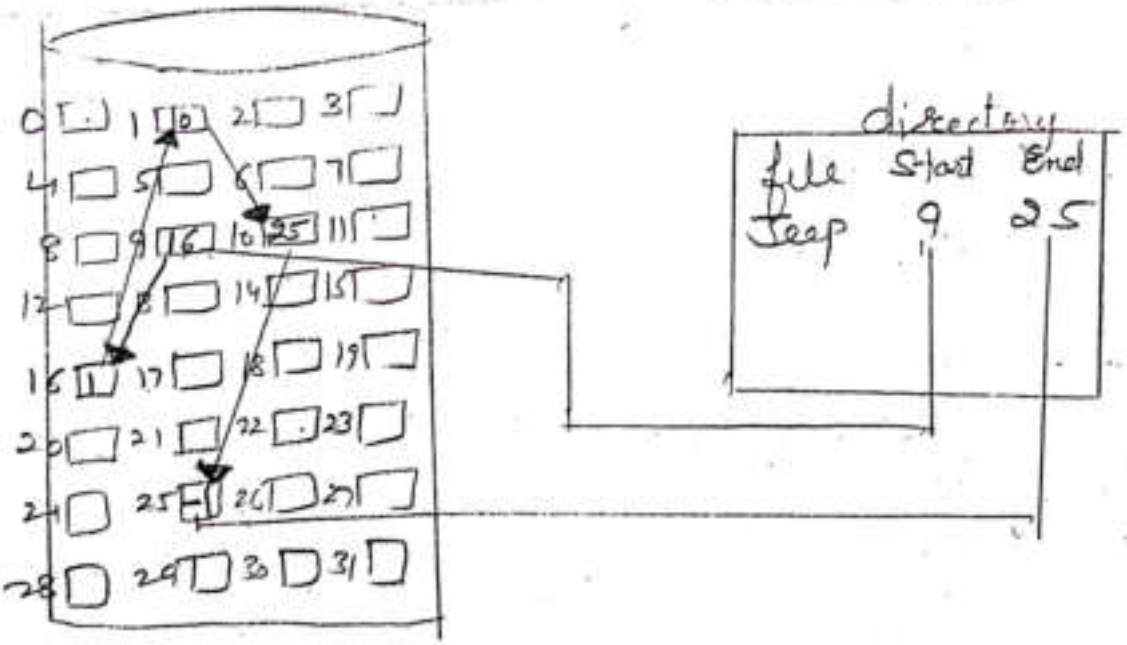
Simple - need only Starting address

Free-space management - no waste of space

No Random access

- Each file is linked list of disk blocks; blocks may be scattered anywhere on the disk.





LINKED ALLOCATION

Directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to next block. These ptrs are not available to the user

9 if each block is 512 bytes & pointer requires 1 bytes, then user sees blocks of 508 bytes

Limitations ⟶

1). It can be used efficiently for sequential access files. To read ith block, we must start at beginning of file & follow the pointers until we get ith block.

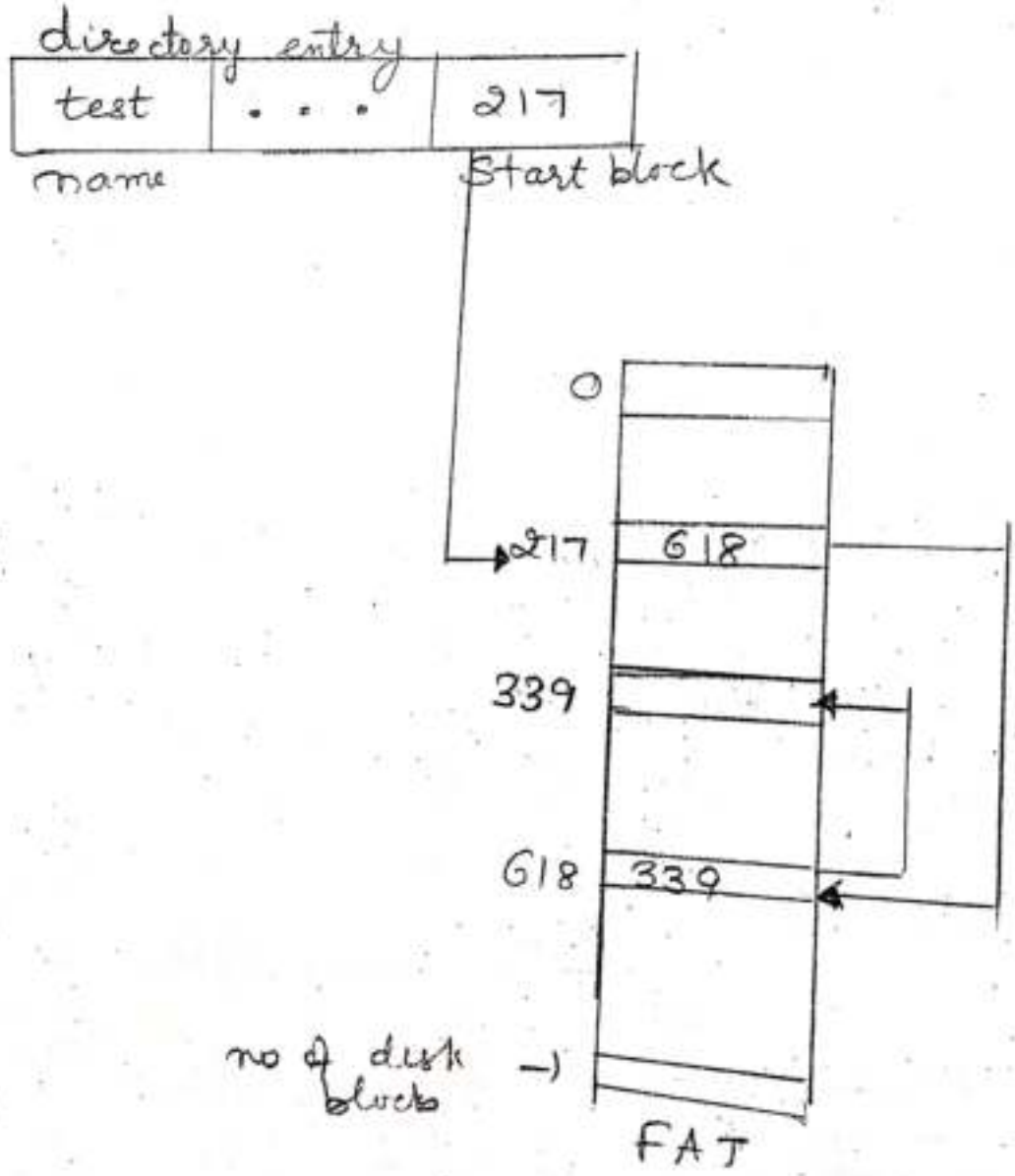


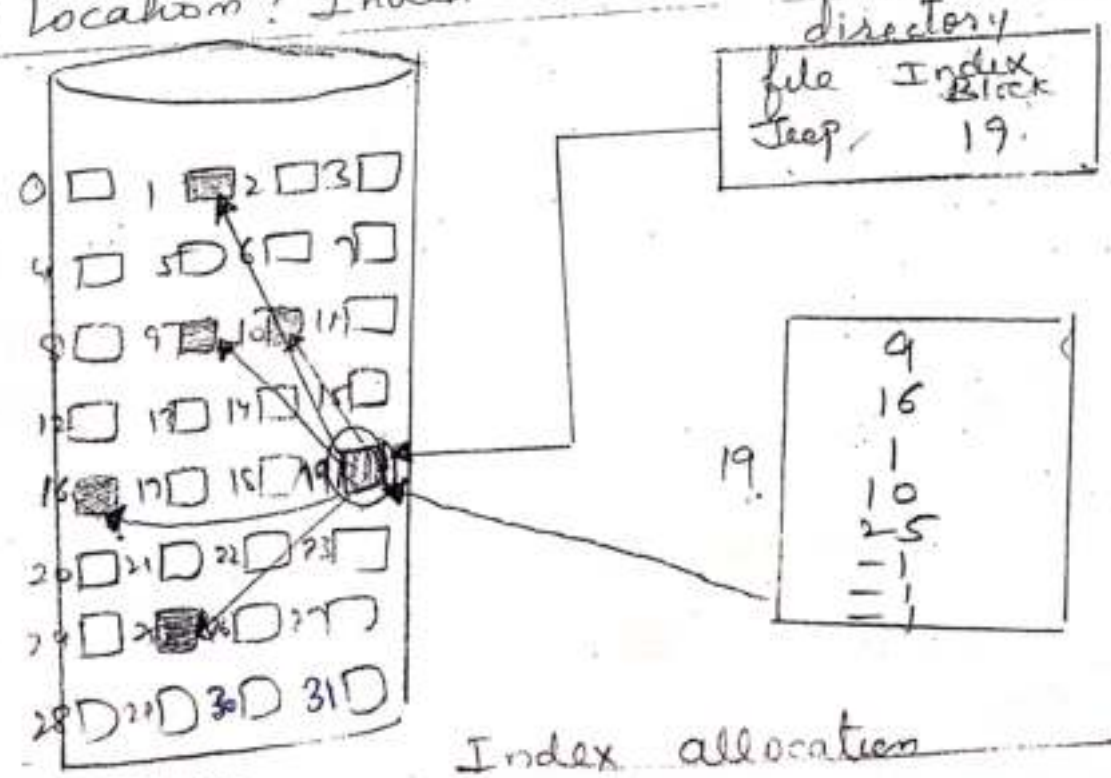
FIG: FILE ALLOCATION TABLE

Another disadvantage is space requirement of pointers. If a pointer requires 4 bytes out of 512 byte block then 0.78 % of disk is being used for pointers.

Indexed Allocation

Linked allocation doesn't support direct access, to solve problem Index allocation is used.

In Index allocation solves this problem by bringing all the pointers together into one location: Index Block.



Index allocation

Index allocation supports direct access without suffering from external fragmentation

Pointer overhead of the index block is greater than the pointer overhead of linked block.

Access Methods

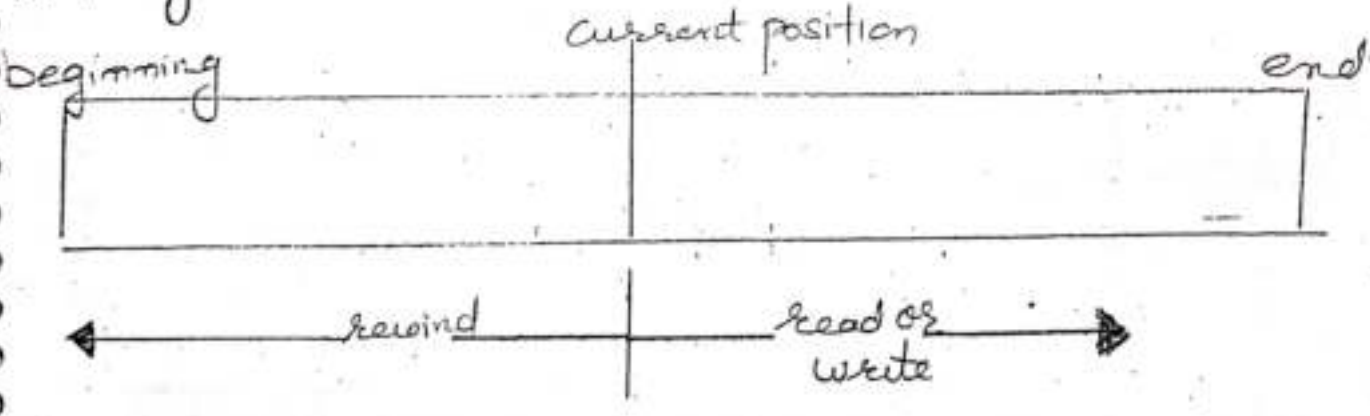
- Access Methods determine the way in which files are accessed and read into memory

Sequential Access

- Commonly used by editors and compilers
- Information is processed in order

read next
write next
reset
no read after last write

- Based on Tape model of file.
- May be able to skip forward n records



Direct access

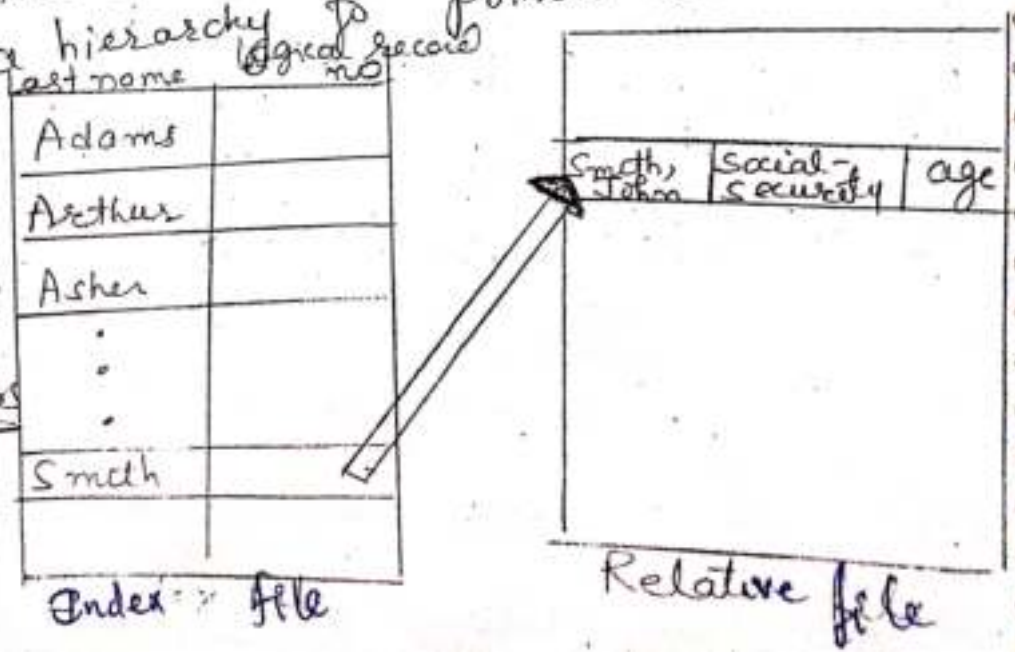
- File is viewed as numbered sequence of blocks or records
- Useful in databases
- Random access file organization provides accessing the records directly
- Each record has its own address on file with by the help of which it can be directly accessed for reading or writing

→ Airline reservation system.
→ Customes account processing in a bank.

3) Index & Relative files

Index Sequential Access method (ISAM) – uses indexes in a hierarchy to point to records in a file.

Index is created for each of which contains pointers various blocks



Directory

A directory contains a no of entries one per file. Each entry contains filename, file attributes, disk addresses where data are stored.

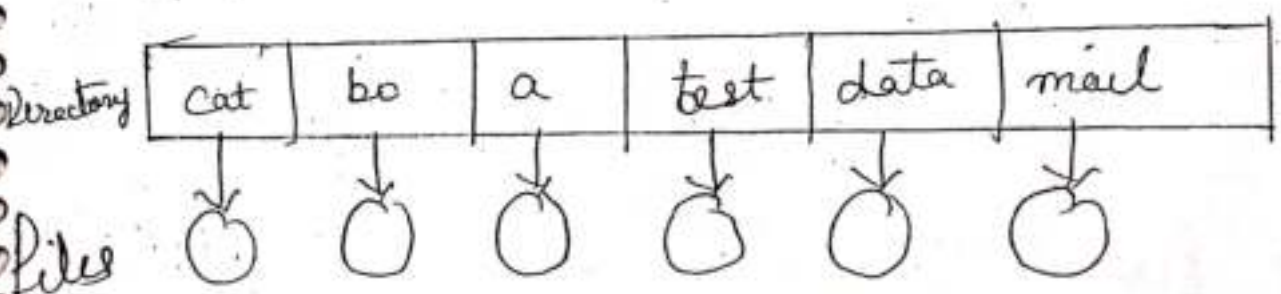
operations performed on directory

1. Searching for a file.
2. Creating a file.
3. Deleting a file.
4. List a Directory.
5. Rename a file.
6. Traverse the file system.

Single Level Directory

All files are contained in the same directory.

advantage → easy to support & understand



Limitations

1) When system has more than one users, Since (than) all files are in the same directory they must have unique file names. But it mayn't be possible for 2 users to give unique filename.

2) And if single user on single-Level directory may feel it difficult to remember the name of all files as no of files increase

---

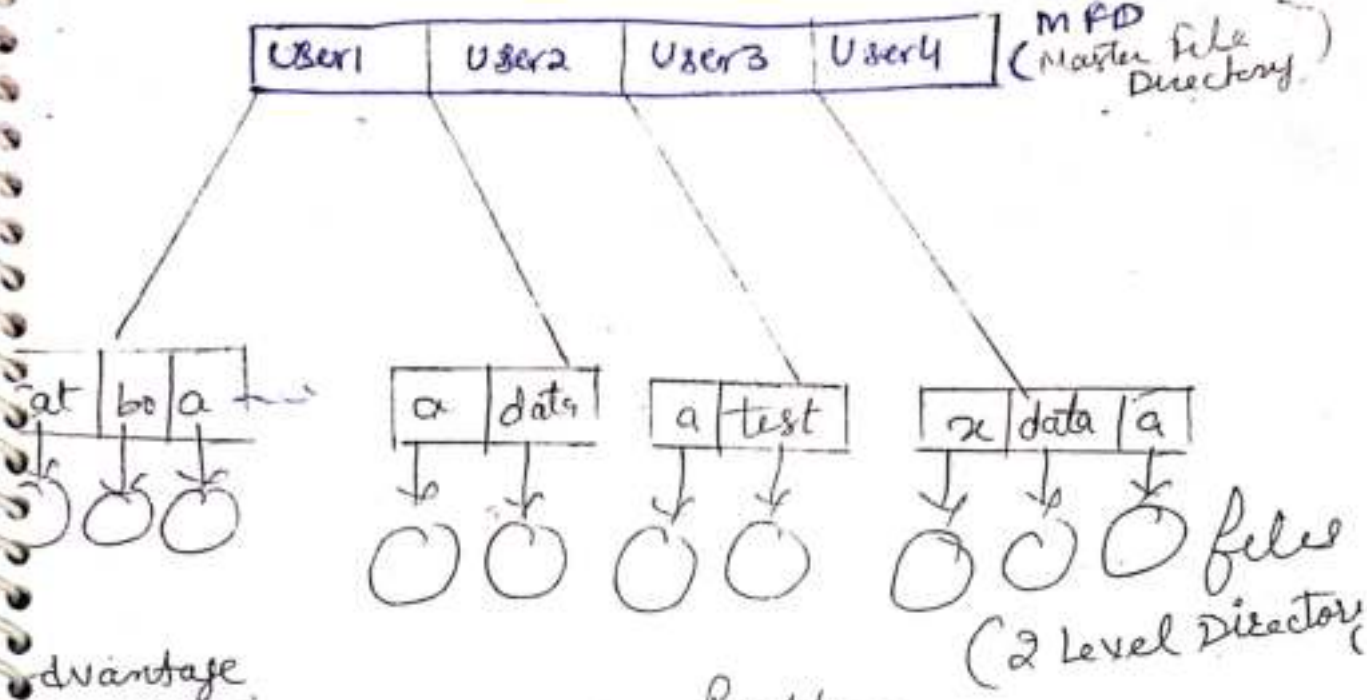
2 Level Directory

→ Separate Directory is created for each user. Called User File Directory (UFD)

→ When a user job starts or user log in the system it searches the master file Directory (MFD)

→ When a user refers to a file only his own UFD is searched.

→ All file names within same UFD must be unique



Advantage

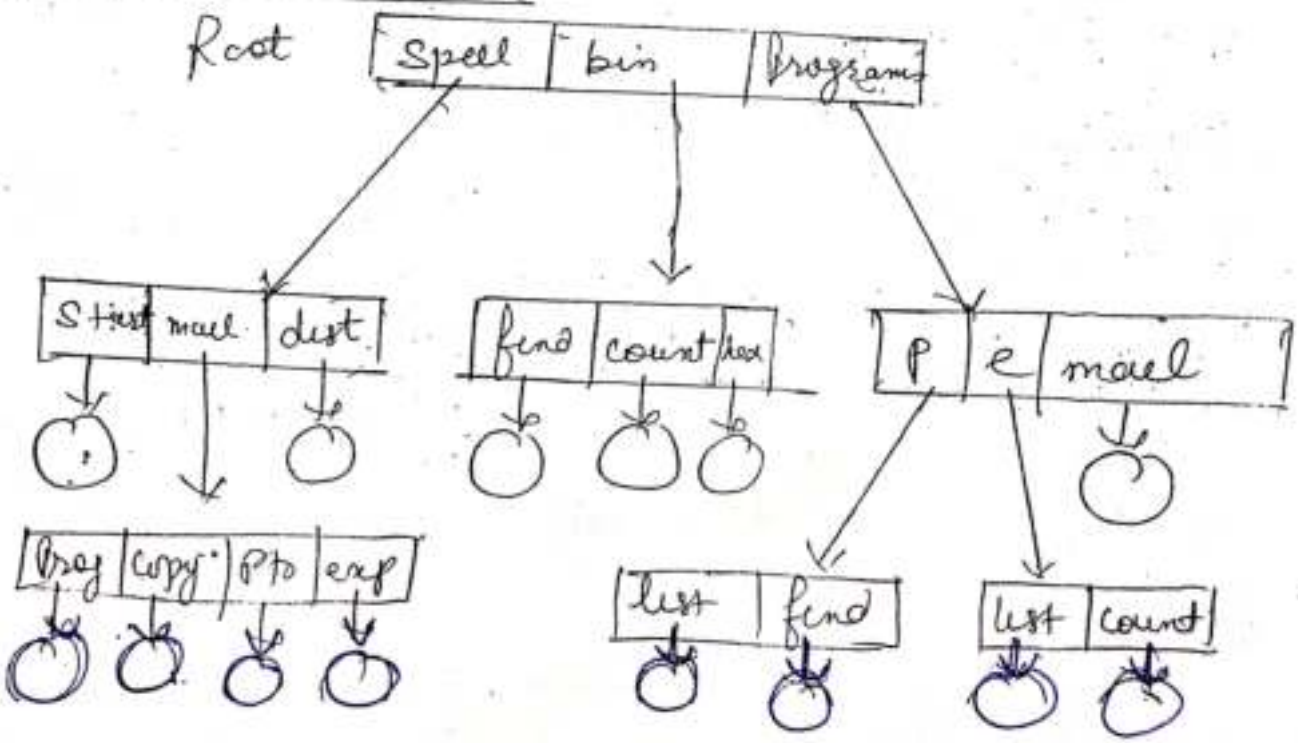
(1) Solves naming Problem

Disadvantage

(1) It isolates one user from another, so there is no sharing of data.

---

(3) Tree structured →



It allows users to create their own subdirectories & to organize their files in them.

→ every file in the system has unique path.

→ Directory contains set of files or subdirectories

Path names can be of 2 types -:

1 Absolute path → Begins at root and follows a path up to specified file giving directories names on path

eg root/spell/mail/ptr

2 Relative Path → Path from current directory

Disadvantage → Sharing of files & sub directories can't be done

Acyclic Graph directory

A shared directory or file will exist in the system in 2 (or more) places at once.

It is graph with no cycles. It allows the directories to share sub-directories with shared file, only one copy of actual file exists, so any changes made by one user are immediately visible to others.

→ When people are working as a team all the files they want to share can be put in one directory.

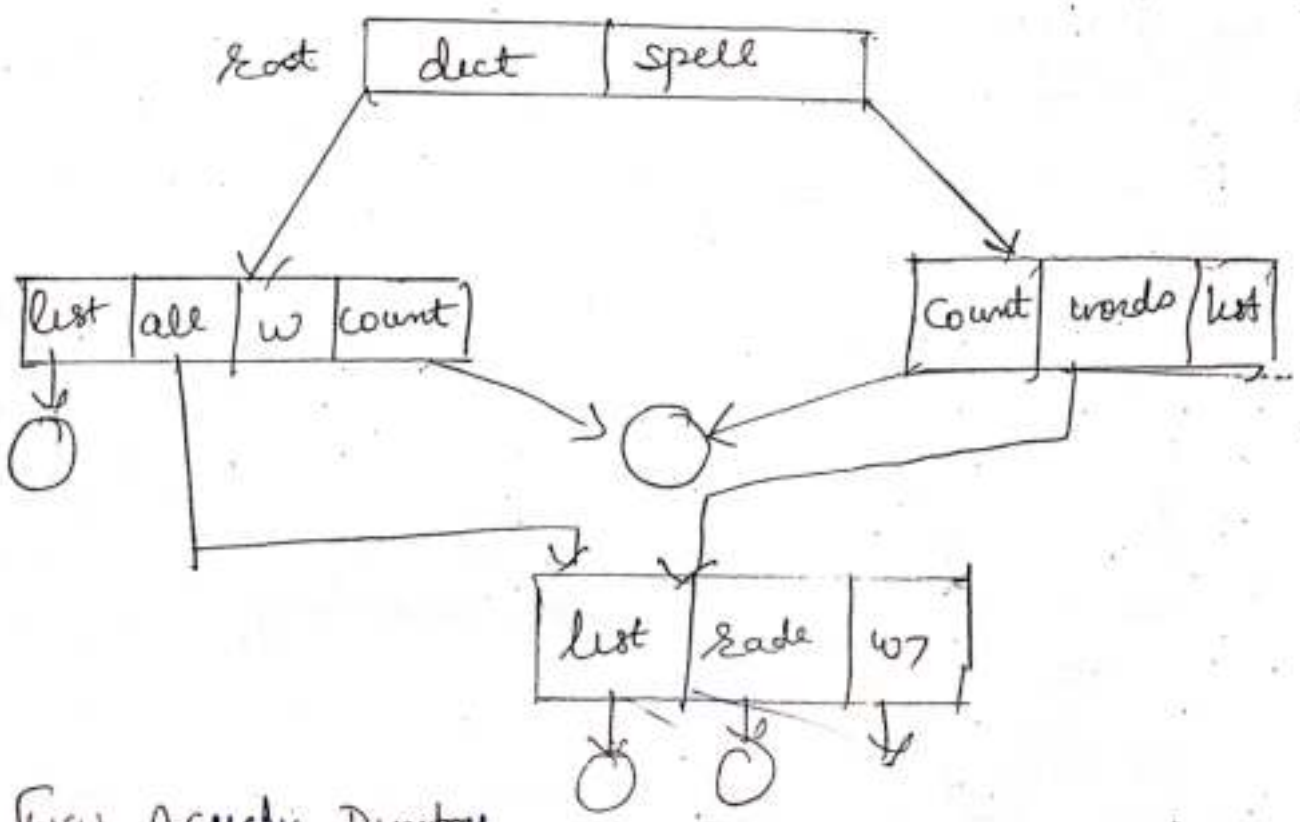


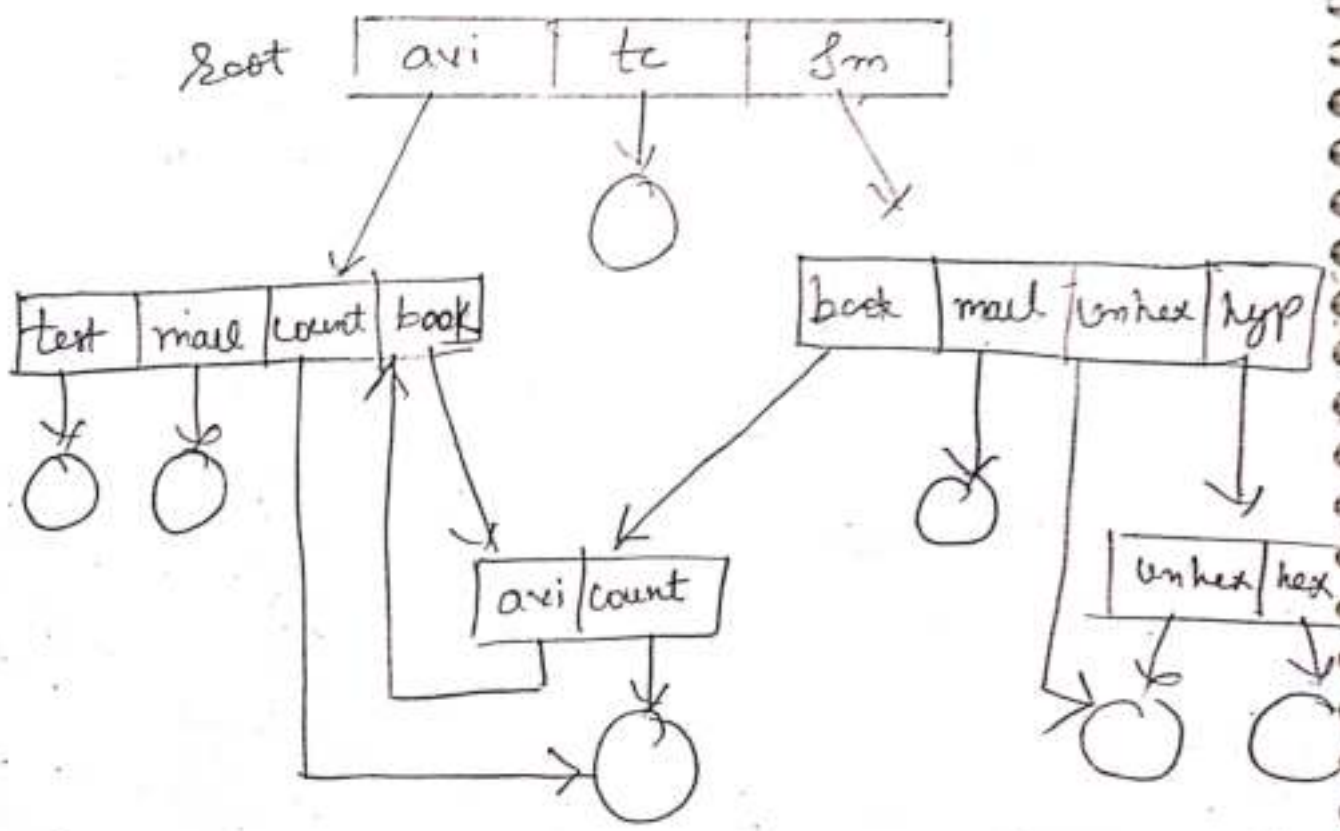
Fig: Acyclic Directory

Advantages

Sharing of files & directories

General Graph directory

When we add links to an existing tree structured directory, tree is destroyed result in simple graph.



General Graph Directory

⇒ Free - Space Management :

- File system maintains free - space list to track available blocks / clusters
- Limited amount of disk space.
- Necessary to reuse space from deleted files.

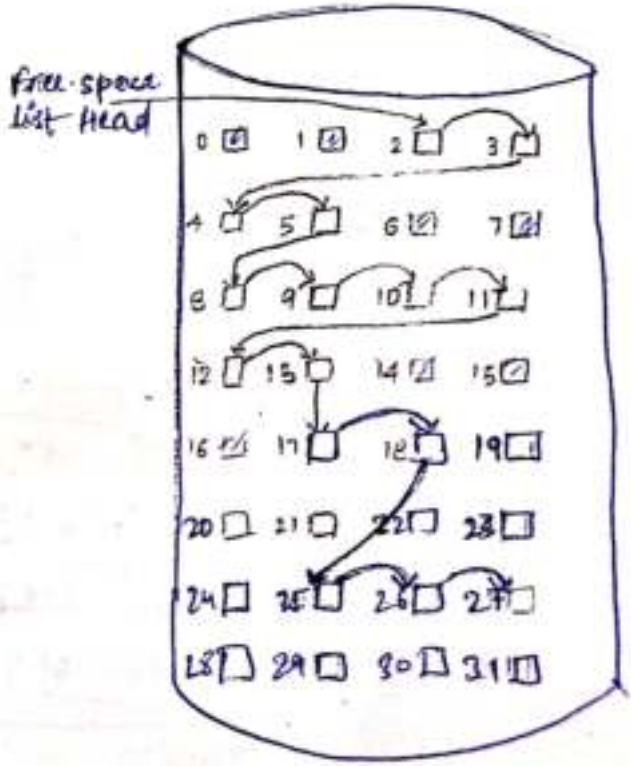
⇒ Bit vector or bit map (n blocks) :

- If block is free the bit is set to 1.
- If block is allocated the bit is set to 0.
- Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, are free and rest are allocated.
- Bit Map = 00111100111110000000 - - - - - - - - -

⇒ Linked Free Space List on Disk :

⇒ Linked list (free list) :

- Cannot get contiguous space easily.
- No waste of space.
- No need to traverse the entire list (if # free blocks recorded).



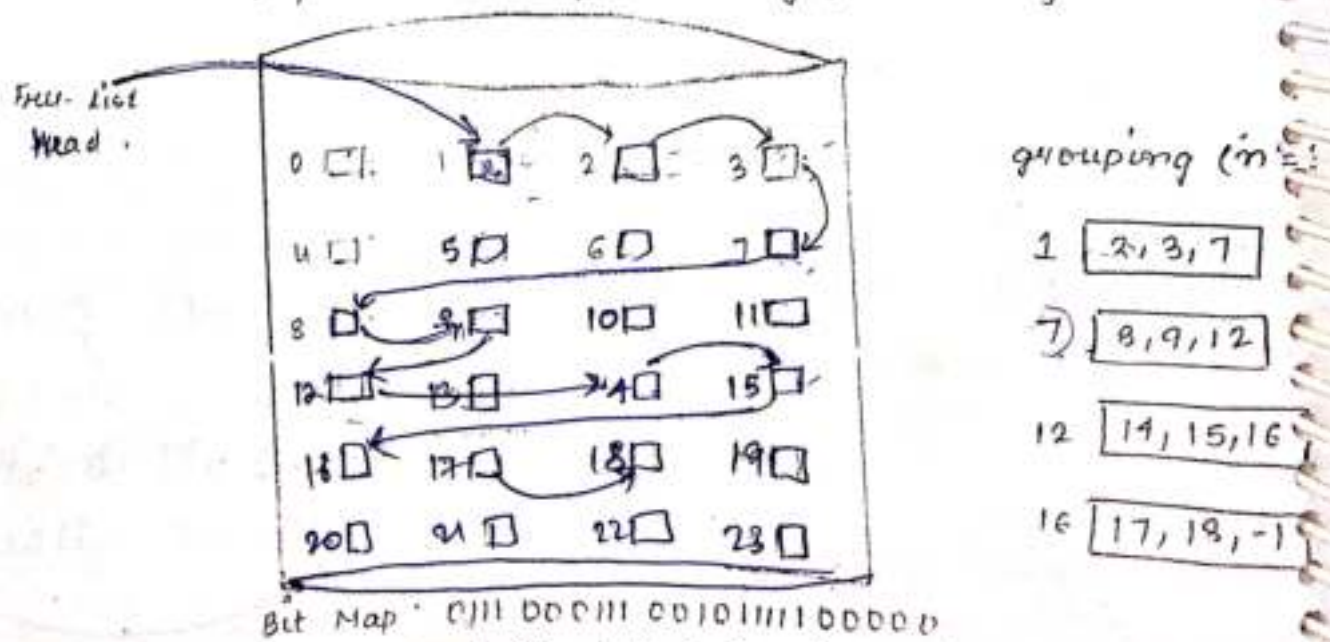
Free Space Management (cont.)

- Grouping :
  - Modify linked list to store address of next n-1 free blocks in first free block, plust a pointer to next block that contains free-block-pointers (like this one).

- Counting :
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents or clustering.
    - Keep address of first free block and count of following free blocks.
    - Free space list then has entries containing addresses and counts.

Example

Bit Map / linked list / Grouping / counting



v. imp. Pg 404.

MODES OF DATA TRANSFER

I/O - MANAGMEN

1) Programmed I/O (Polling)

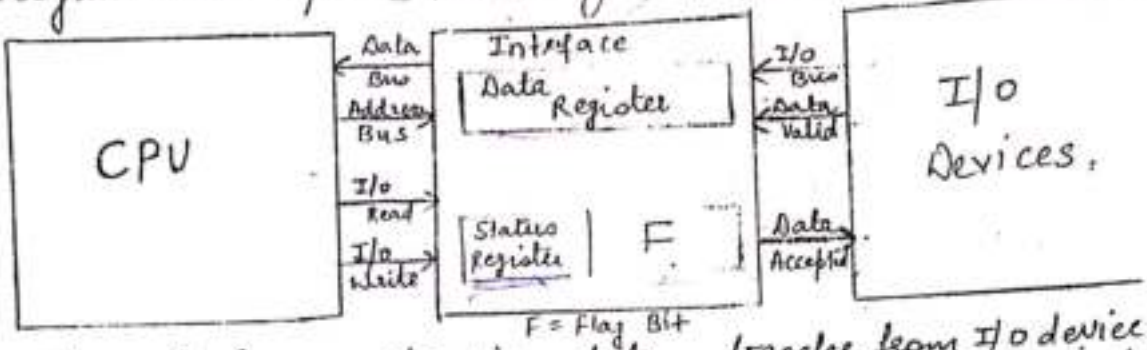
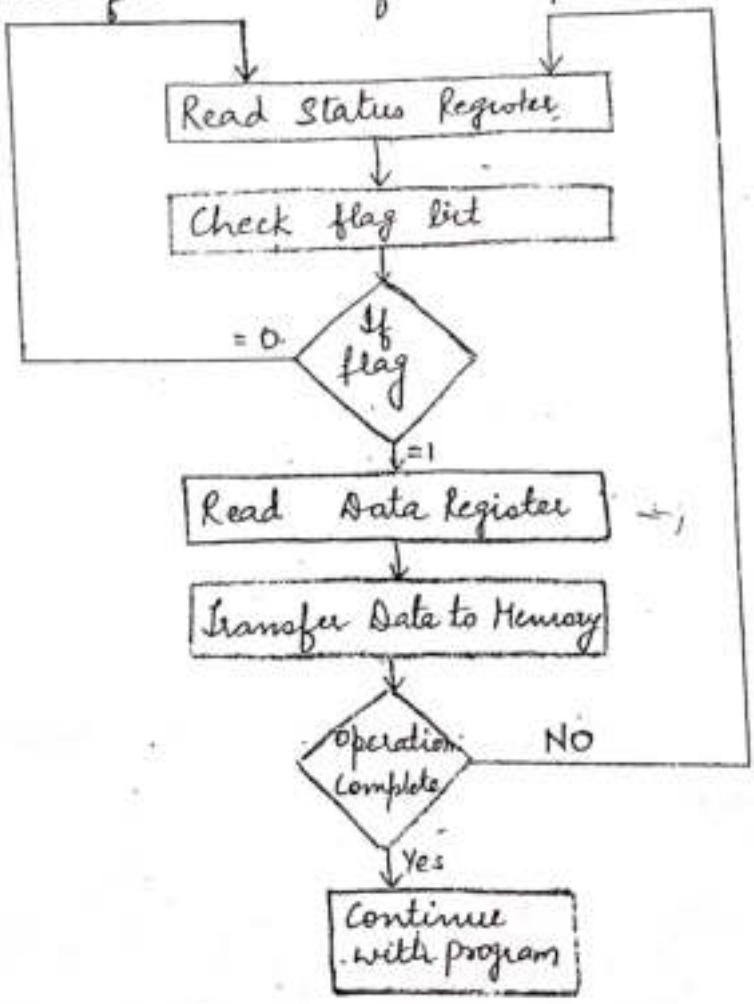


Fig: Diagram showing data transfer from I/O device

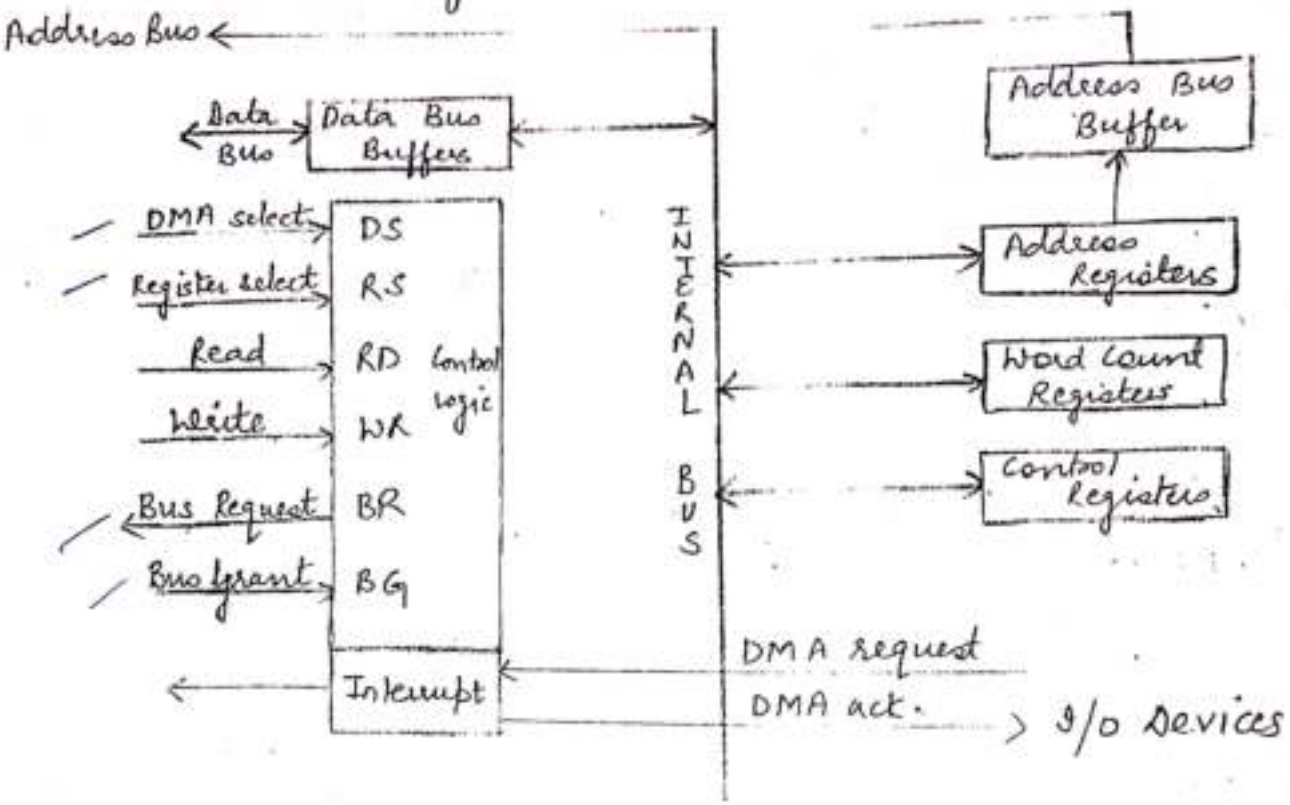
→ Programmed I/O operations are the result of I/O instr written in the computer program.

→ Each data transfer is initiated by the instruct and the flow of data is from I/O device to CPU.

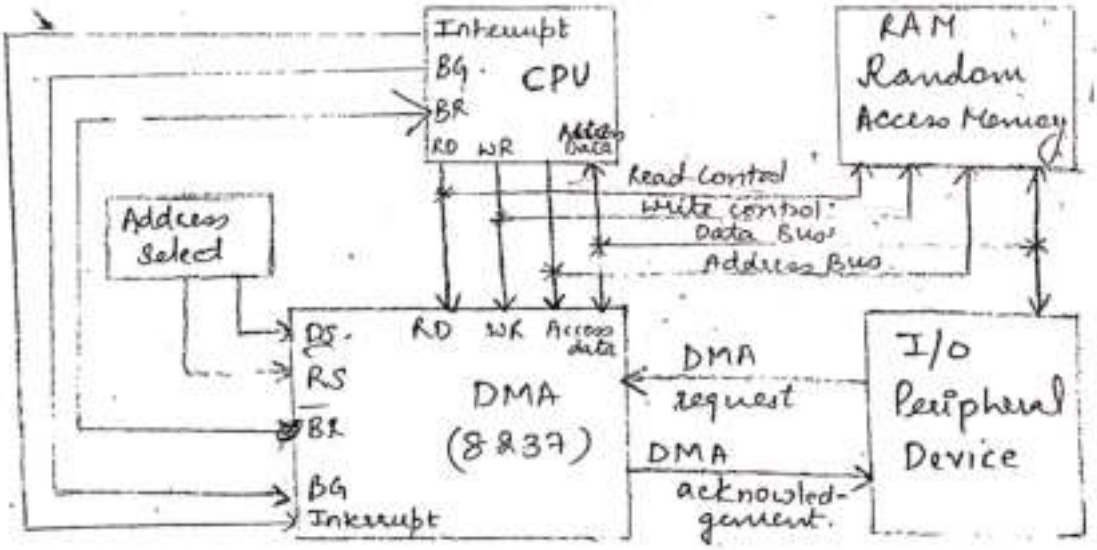


Flowchart for CPU Program to input dat

* Structure of DMA Controller (8237)



* The Process of DMA



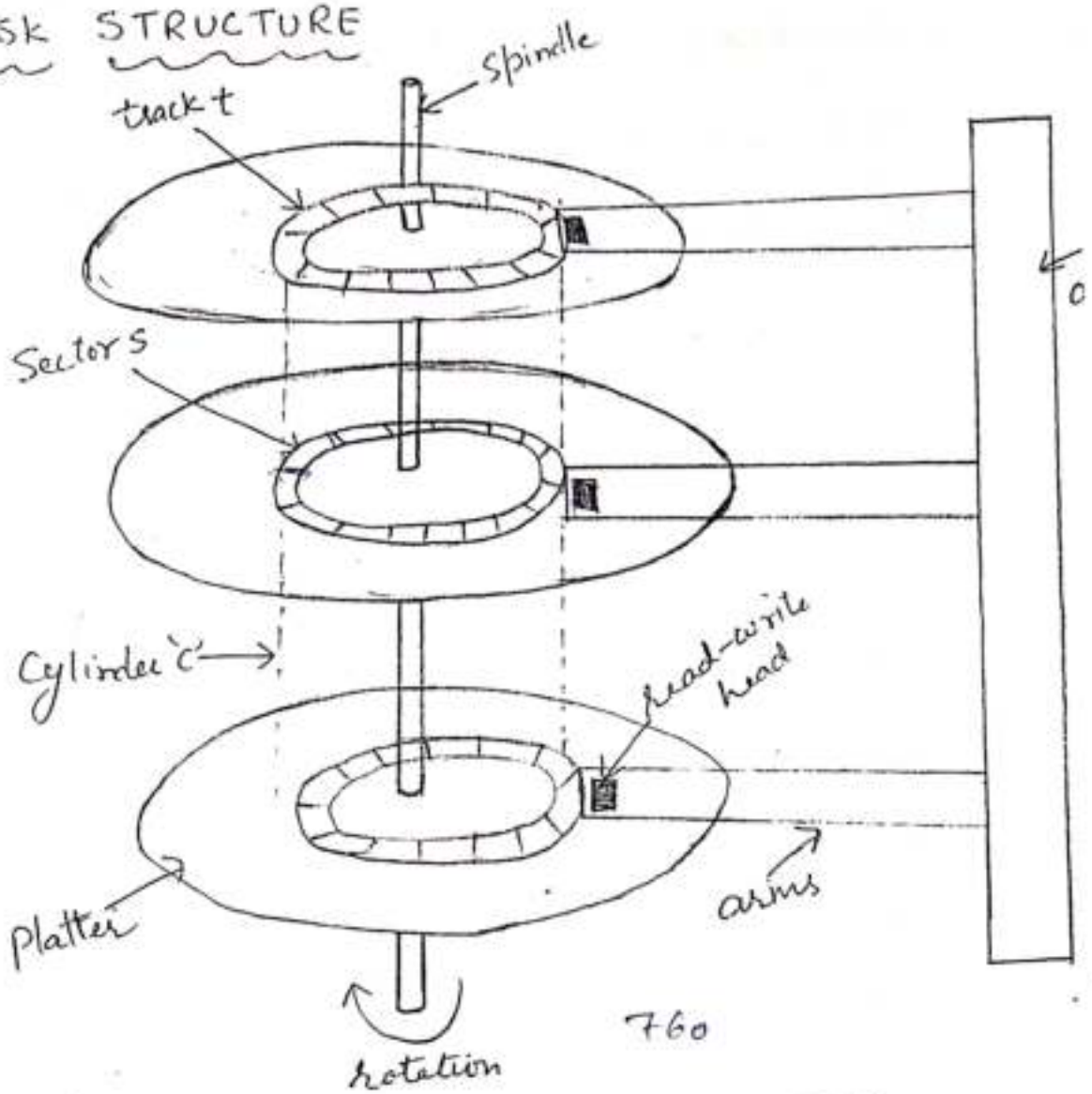
* Performance Matrices →

EXECUTION TIME → Time from submission to completion for a particular task.

$$\text{Performance} \propto \frac{1}{\text{Execution Time}}$$

UNIT-4 DISK MANAGMENT

(1) DISK STRUCTURE



(MOVING HEAD DISK STRUCTURE)

→ A hard disk Consist of a stack of disks called platters mounted on a Common spi

→ Each Surface of the platter is coated with magnetic material.

→ In top and bottom platters only the inner Si are used.

→ for each surface of platters, a separate read write head is used.

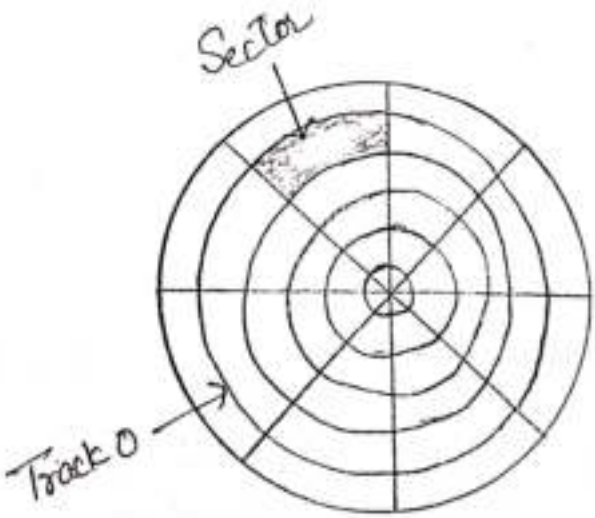
→ The set of read-write head is joined with a common arm

→ The disk pack is rotated at a high speed as 7600 rpm

→ When a constant speed is reached then all the arms move together on the required track on all surface.

→ only one head is selected at a time access the data from its corresponding

→



→ The above fig. shows the disk contain different tracks.

→ The tracks are divided into Sectors.

→ The Sector is the smallest addressable unit a the disk.

→ In Context of disk geometry, the following t are important.

1) Seek Time :- It is a time taken to move a R/W head on to a particula track is called a Seek time

2 Rotational Latency :- It is the additional time for the disk to rotate the desired Sector to the disk head.

3 Data transfer time :- It is the time to for the transfer of data.

Disk Access Time = Seek time + Rotational Latency + Data Transfer time.

→ Some time seek is a slow operation because it involves physical movement of the read/write arm from the current track to the target track.

→ So the O.S avoids high seek time by storing data in tracks that are below or above each other, present in different platters.

→ The group of same numbered tracks different surface/platters is called cylinder.

→ The advantage of cylinder is that info from a cylinder can be accessed without physically moving the read-write arm

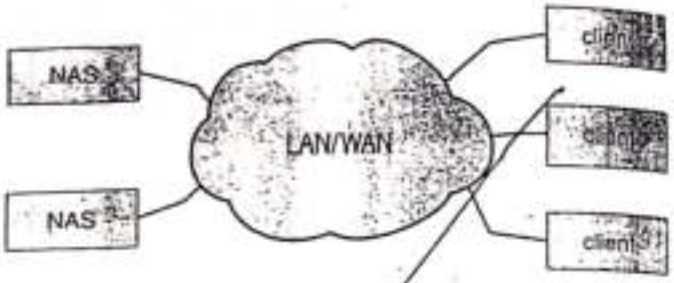


Figure 12.2 Network-attached storage.

Network-attached storage provides a convenient way for all the comput on a LAN to share a pool of storage with the same ease of naming and acc enjoyed with local host-attached storage. However, it tends to be less efficie and have lower performance than some direct-attached storage options.

iSCSI is the latest network-attached storage protocol. In essence, it uses th IP network protocol to carry the SCSI protocol. Thus, networks—rather th SCSI cables—can be used as the interconnects between hosts and their storag As a result, hosts can treat their storage as if it were directly attached, even the storage is distant from the host.

12.3.3 Storage-Area Network

One drawback of network-attached storage systems is that the storage i operations consume bandwidth on the data network, thereby increasing t latency of network communication. This problem can be particularly a in large client–server installations—the communication between servers an clients competes for bandwidth with the communication among servers a storage devices.

A storage-area network (SAN) is a private network (using storage protoc rather than networking protocols) connecting servers and storage units, shown in Figure 12.3. The power of a SAN lies in its flexibility. Multiple ho and multiple storage arrays can attach to the same SAN, and storage c be dynamically allocated to hosts. A SAN switch allows or prohibits acc between the hosts and the storage. As one example, if a host is running h on disk space, the SAN can be configured to allocate more storage to that ho SANs make it possible for clusters of servers to share the same storage and f storage arrays to include multiple direct host connections. SANs typically ha more ports, and less expensive ports, than storage arrays.

FC is the most common SAN interconnect, although the simplicity of iSCSI increasing its use. An emerging alternative is a special-purpose bus architect named InfiniBand, which provides hardware and software support for hig speed interconnection networks for servers and storage units.

12.4 Disk Scheduling

Disk scheduling →

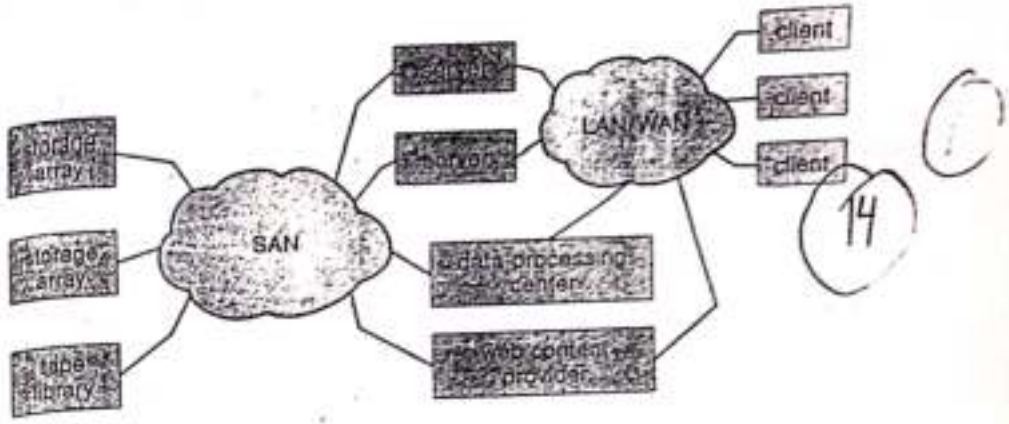


Figure 12.3 Storage-area network.

fast access time and large disk bandwidth. The access time has two major components (also see Section 12.1.1). The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The rotational latency is the additional time for the disk to rotate the desired sector to the disk head. The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:

- Whether this operation is input or output
- What the disk address for the transfer is
- What the memory address for the transfer is
- What the number of sectors to be transferred is

If the desired disk drive and controller are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive. For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the operating system chooses which pending request to service next. How does the operating system make this choice? Any one of several disk-scheduling algorithms can be used, and we discuss them next.

Seek time

12.4.1 FCFS Scheduling

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with requests

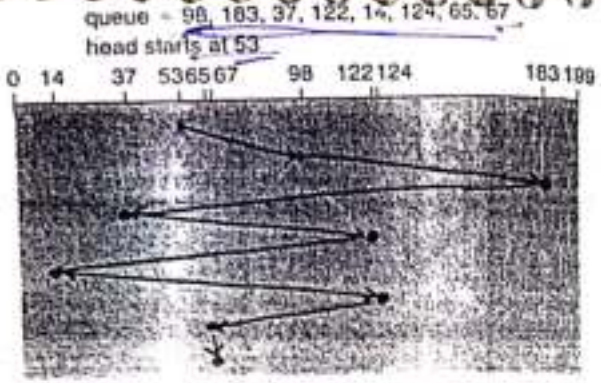


Figure 12.4 FCFS disk scheduling.

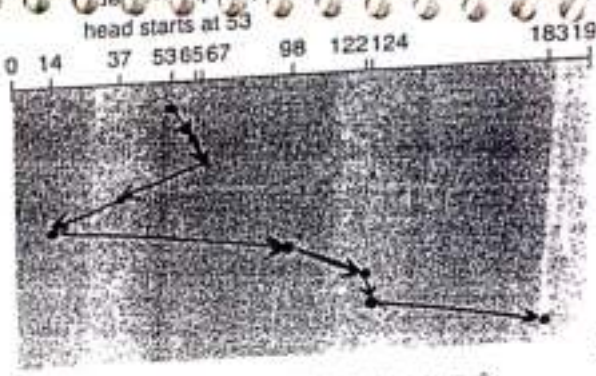


Figure 12.5 SSTF disk scheduling.

in that order. If the disk head is initially at cylinder 53, it will first move to 65 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders. This schedule is diagrammed in Figure 12.4.

The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

12.4.2 SSTF Scheduling

It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the shortest-seek-time-first (SSTF) algorithm. The SSTF algorithm selects the request with the least seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

For our example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183 (Figure 12.5). This scheduling method results in a total head movement of only 236 cylinders—little more than one-third of the distance needed for FCFS scheduling of this request queue. Clearly, this algorithm gives a substantial improvement in performance.

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling, and like SJF scheduling, it may cause starvation of some requests. Remember that requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while the request from 14 is being serviced, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could cause the request for cylinder 186 to wait indefinitely.

This scenario becomes increasingly likely as the pending-request queue grows longer.

Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal. In the example, we can do better by moving the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122, 124, and 183. This strategy reduces the total head movement to 208 cylinders.

12.4.3 SCAN Scheduling

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the *elevator* algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

Let's return to our example to illustrate. Before applying SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement in addition to the head's current position. Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183 (Figure 12.6). If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

Assuming a uniform distribution of requests for cylinders, consider the density of requests when the head reaches one end and reverses direction. At this point, relatively few requests are immediately in front of the head, since these cylinders have recently been serviced. The heaviest density of requests

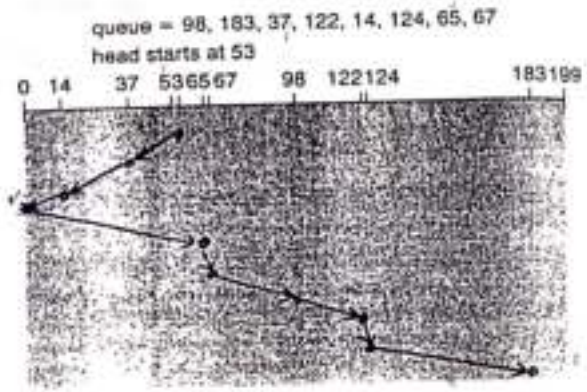
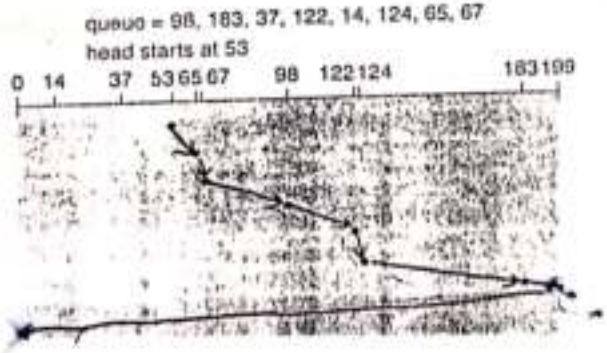


Figure 12.6 SCAN disk scheduling.

is at the other end of the disk. These requests have also waited the longest, why not go there first? That is the idea of the next algorithm.

12.4.4 C-SCAN Scheduling

C-SCAN scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip (Figure 12.7). The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.



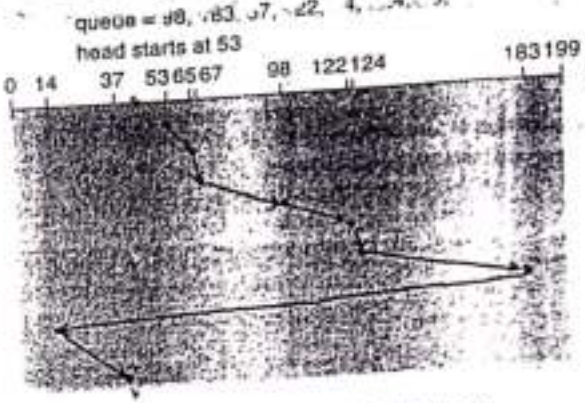


Figure 12.8 C-LOOK disk scheduling.

12.4.5 LOOK Scheduling

As we described them, both SCAN and C-SCAN move the disk arm across the full width of the disk. In practice, neither algorithm is often implemented this way. More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. Versions of SCAN and C-SCAN that follow this pattern are called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in a given direction (Figure 12.8).

12.4.6 Selection of a Disk-Scheduling Algorithm

Given so many disk-scheduling algorithms, how do we choose the best one? SSTF is common and has a natural appeal because it increases performance over FCFS. SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem. For any particular list of requests, we can define an optimal order of retrieval, but the computation needed to find an optimal schedule may not justify the savings over SSTF or SCAN. With any scheduling algorithm, however, performance depends heavily on the number and types of requests. For instance, suppose that the queue usually has just one outstanding request. Then, all scheduling algorithms behave the same, because they have only one choice of where to move the disk head: they all behave like FCFS scheduling.

Requests for disk service can be greatly influenced by the file-allocation method. A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.

The location of directories and index blocks is also important. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently. Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. In

UNIT-4

DISK PERFORMANCE PARAMETERS

→ The three important aspect of storage perf are: ① Speed ② Reliability ③ cost

① Speed :- The speed of tertiary storage has aspects

① Bandwidth ② latency

→ we measure the bandwidth in bytes per sec

→ The bandwidth are of two types

① Sustained Bandwidth
② effective Bandwidth.

① Sustained Bandwidth :- It is no. of bytes d by the transfer time.

② Effective Bandwidth :- It calculates the avera over the entire I/O times, inclu the time for Seek() or locate().

→ The bandwidth of a drive is generally the sustained bandwidth.

→ For removable disks, the bandwidth rang from a few megabyte per second for the

slowest to over 40MB per second for the fastest.

→ The second aspect is the access latency.

→ By this performance measure, disks are much faster than tapes.

→ If a jukebox is involved, the access lat can be significantly higher.

→ For tape, the robotic-arm time is als the same as for disk.

→ So, in generalize, we say that random ac in a disk jukebox has a latency of te of seconds, whereas random access in a jukebox has a latency of hundred of seco

→ The bandwidth to storage capacity ratio robotic library is much less favorable that of fixed disk.

→ The low cost of tertiary storage res from having many cheap & few ex drives.

→ The removable storage is best storage of used data.

2)

DISK RELIABILITY

→ Removable magnetic disk are somewhat less re than are fixed hard disks because the cartri is more likely to be exposed to harmful condi such as dust, large changes in temperature an humidity and mechanical forces such as shock bending.

→ Optical disks are very reliable because the lay that stores the bits is protected by a trans plastic or glass layer.

→ The reliability of magnetic tape varies widel depending on the kind of drive.

→ By comparison with a magnetic-disk head, head in a magnetic-tape drive is a weak.

→ In conclusion, the fixed disk drive is like to be more reliable than a removable disk tape drive and an optical disk is likely to more reliable than a magnetic disk or tape

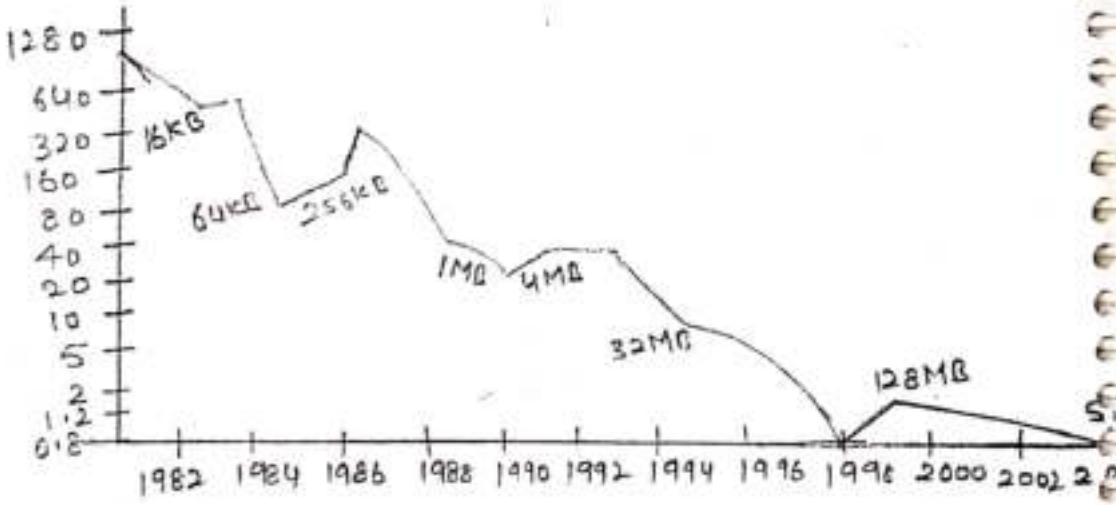
→ A fixed magnetic disk has one weakness

→ A head crash in a hard disk generall destroy the data.

3) Cost :-

→ The removable media way lower the overa
Storage cost.

→ The cost per gigabyte of removable storage
may well be lower than the cost per gigaby
of a hard disk, because the expense of o
drive is averaged with the low price of m
removable cartridges..



(Price Per megabyte of DRAM, from 1981 to 20

→ From the above graph we see that the co
of storage has fallen dramatically over the pa
twenty years or so.

→ By comparing the graphs, we can also se
that the price of disk storage has plummeted
relative to the price of DRAM and Tape.

UNIT-1

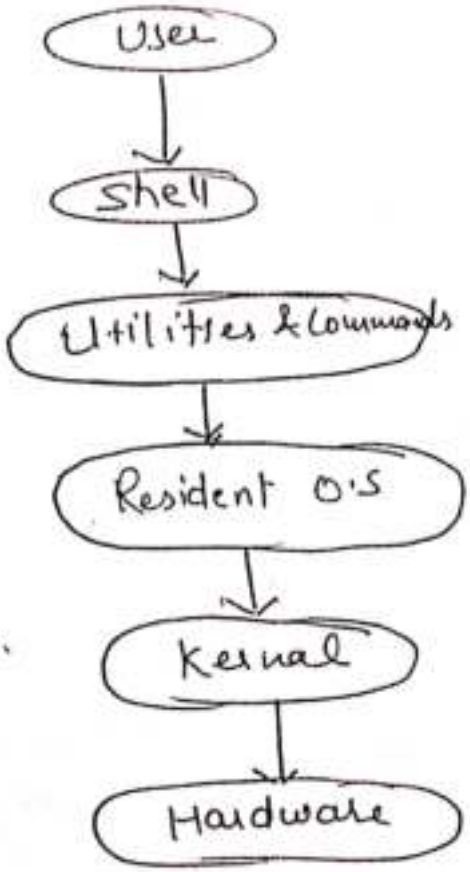
(8)

UNIX

→ UNIX is a time sharing system, whose pro
Segments swapped in & out of the memory o
required.

→ UNIX user communicates with the operating
System through shell.

UNIX STRUCTURE



(Simple layer structure of UNIX)

UNIX FILE SYSTEM

→ If the UNIX kernal is the heart of the operating system, then the file system is its soul.

→ The file system usually refers to the entity which contains all the data for the workst or server: server data, system data, user da & all binary executable programs runnable on machine.

Virtual FILE SYSTEM OR UNIX DIRECTORY TREE

→ The hierarchical representation of files & directories of files with which a user or administer may perform various functions.

Eg. editing, viewing, executing etc.

→ This is the UNIX interface to the physical system.

→ Many UNIX resources are aviable via the virtue file system.

PHYSICAL FILE SYSTEM ON DISK

→ The physical file system may be arranged on or more actual storage devices typically hard d

Disk partitions & the disk label :-

(4)

Disk label contains two other necessary pieces of information for the OS to boot

(i) Disk geometry
(ii) Code to find the UNIX kernel on the disk & load it. [Boot Loader]

The kernel-loading code is generally referred to as the boot loader, but has different names under different flavors of UNIX (eg. lilo for Linux and sash for IRIX.)

In the UNIX file System, six Types of files are distinguished

1. Regular or ordinary
2. Directory
3. Special
4. Named pipes
5. Links
6. symbolic Links

Regular, or ordinary :- Contains arbitrary data in zero, or more data blocks.
Regular files contain information entered in them by a user, an application program, or a system utility program.

Directory :- Contains a list of filenames plus pointers to associated inodes (index nodes).
* Directories are hierarchically organized.
* Directory files are actually ordinary files with special write protection privileges so that only the file system can write into them.

Special :- Contains no data, but provides a mechanism to map physical devices to file names.

Named pipes :

* Pipe is an interprocess communication facility.
* A pipe file buffers data received in its input so that a process that reads from the pipe's o/p receives the data on a FIFO basis.

Links :- Link is an alternative file name for an existing file.

Symbolic links :- This is a data file that contains the name of the file it is linked to.

Volume Structure

UNIX filesystem resides on a single logical disk or disk partition & is laid out with the following elements :-

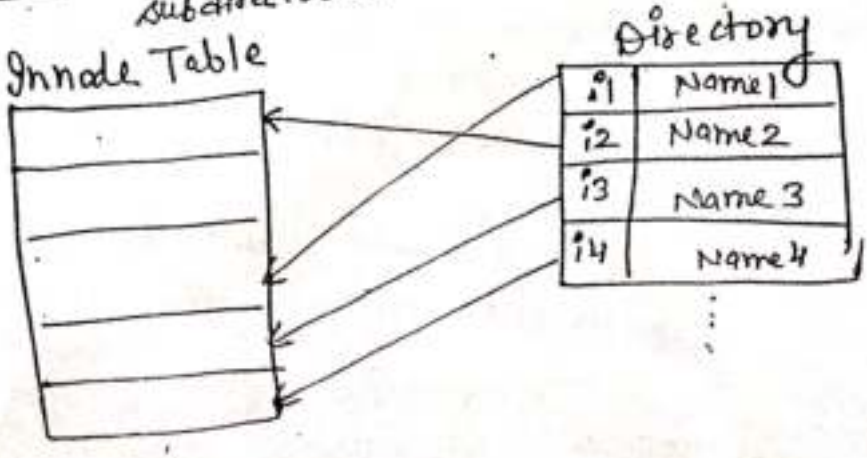
1. Boot block
2. super block
3. Inode Table
4. Data blocks

Boot Block :- Contains code required to boot the OS.

Superblock :- contains attributes & information about the filesystem such as size of partition, inode table size

Inode Table :- The collection of inodes for each file.

Data blocks :- Storage space available for data files & subdirectories.



Inodes :-

* All types of UNIX files are administered by the OS by means of inodes.
* An inode (index node) is a control structure that contains the key information needed by the OS for a particular file.

=) Several filenames may be associated with a single inode, but an active inode is associated with exactly one file, & each file is controlled by exactly one inode.

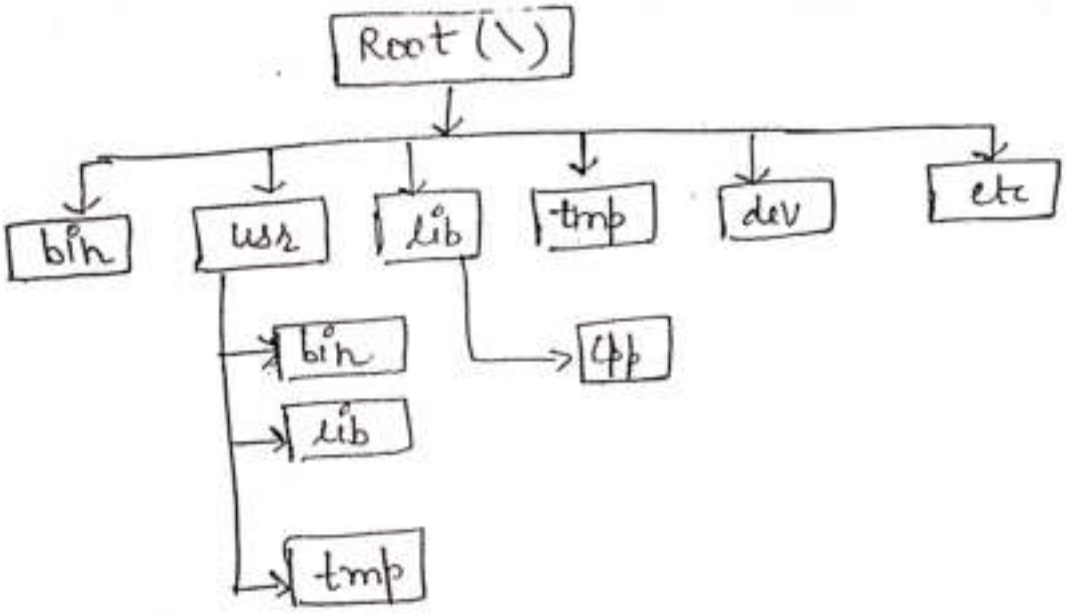
Superblocks :-

* Superblock provides the mapping to the root directory, & from there one can find any other file on the filesystem.
* superblock is a data structure which includes information about the
  (i) type of filesystem (NFS, ext2fs etc)
  (ii) size
  (iii) modification time of filesystem
  (iv) list of free & allocated blocks & first inode which points to root directory.
* superblock is always replicated to provide tolerance against disk failure in the first superblock.

* Unix system creates a directory, two subdirectories "." and ".." are created automatically.

"." → refer to the current working directory

".." → refer to the parent directory



/dev → contains specific files for peripheral devices

/tmp → maintains all temporary files created while executing different utilities such as editors.

/usr → important directory, to which user belong. Contains 3 major subdirectories – lib, bin, tmp

/usr/bin → contain many executable programs & utilities

/usr/lib → contains libraries for programs & languages such as C

/usr/tmp → stores those files which are used for short duration i.e files created during the execution of a program.

/etc → contains system administration utilities & machine specific administrative configuration files.

It is to be noted that 'ext2fs' enhances the efficiency of I/O operations by writing the blocks of a file in close vicinity of each other. For small files it places the blocks into block groups, and for large files the blocks are allocated within a cylinder group with a view to reduce seek time.

11.9 WINDOWS FILE SYSTEM

Windows mainly support three types of file systems—FAT (File allocation table), NTFS (New technology file system), and HPFS (high performance file system). The FAT was developed by Microsoft for MS-DOS in 1976, and is also supported by OS/2. It evolved through many stages such as FAT 12, FAT 16, and FAT 32. This file system supports a maximum of 64 K units of allocation, indicating that for a disk having a partition size 64 MB, the FAT will have an allocation unit of size 1 KB (64 MB/64 K). Similarly, a 32 MB partition will have an allocation unit of size 512 bytes and a disk partition of 512 MB will have the size of allocation unit as 4 KB. Thus, as the size of the disk partition grows the size of the allocation unit also grows. For smaller files, the larger allocation units lead to a significant wastage of memory because the last unit of the file remains unfilled. Because of this shortcoming, Windows developed a new file system called NTFS (New Technology File System), which extends the capabilities of both FAT and HPFS.

Nevertheless, the FAT file system is extensively used for interchange between digital devices such as cameras, scanners, printers, flash memory, among others.

The NTFS is supported by Windows NT, 2000, 2003, and XP. In this file system, the file manager of Windows maintains a B-tree based index for files. The similar files created by an application are linked and tracked with the help of a link-tracking mechanism. This mechanism quickly tracks the required file and opens it at a very fast rate. The files are so created that their units are allocated in close proximity, spanned over contiguous clusters. When the clusters become fragmented by insertions and deletions performed on the file, the user can defragment the file by using the defragmentation API provided by NTFS. The important features of NTFS are:

1. **POSIX compliance:** NTFS uses file names as per the POSIX convention.
2. **Recoverability:** It provides the ability to recover from disk failures and system crashes, by reconstructing the disk volumes. In fact it keeps track of the changes made to files and records them as transactions. Each transaction is considered an atomic action. As and when a failure occurs, the transactions are used to roll back the file system to an immediate previous state. NTFS not only stores the critical file data on the disk but also duplicates it on a separate designated area. In the event of a sector becoming bad, it uses the stored critical data to reconstruct the lost data.
3. **Security:** A file is created as an object and NTFS assigns security attributes to the object. The Windows security system consults the object's security descriptor before allowing a subject to access the file object.
4. **Multiple data streams:** NTFS allows multiple data streams of data for a file. For instance, the data and the meta-data of a file can be treated as two data streams.

Figure 11.33: Format of a disk volume

6. **Compression:** NTFS uses Lempel-Ziv compression to compress file data. This algorithm is a lossless compression method for compressing data. A sparse file is an excellent candidate for compression because it contains excessive numbers of '0' characters.

The NTFS divides the disk into logical partitions consisting of clusters. A logical partition is also called a **volume** (see Fig. 11.33) A cluster comprises contiguous sectors, and the number of sectors per cluster is a power of 2 i.e., 1, 2, 4, 8, 16, and so on. NTFS can support a maximum of 2^{32} clusters. The file system is independent of the size of the sector and therefore can support any standard or nonstandard disk with any sector size.

In the above figure,

Partition boot sector: This is the first sector or sectors that contain the bootstrap code and other information about the volume layout.

Master file table (MFT): It contains information about the directories and files present on the volume. Each entry in the table pertains to information of a file or a directory such as access parameters, file name, owners id and access rights.

System files: After the MFT, 1 MB of area is reserved for the following system files.

1. MFT2: This is a duplicate copy of the MFT. In the event of an MFT failure, this copy of the MFT is used to recover the data stored in the volume.
2. Log File: This file contains the trail of transactions made in the system. NTFS uses this log file to recover from system crashes.
3. **Cluster bit map:** It keeps track of the clusters being used in the volume.
4. **ADT (attribute definition table):** This table contains the list of attributes supported on the volume.

The input /output operations are performed on virtual files using asynchronous I/O. A virtual file is a file object. It is opened for operations by the following steps:

1. The user process makes the following I/O request on a file (say sudent.dat)

```
fptr = fopen("c:\student.dat", r);
```

2. It is converted into a system call (by the compiler) such as:

```
createFile ("c:\student.dat.....);
```

3. Windows converts the system call into its internal system call by using Win32 DLL. It is handed over to the system services.

```
NtCereateFile ("c:\student.dat.....);
```

4. On receiving this system call, the I/O manager requests the object manager to create the file object.
5. The object manager responds by providing a file handle of the file object to the I/O manager.
6. The I/O manager returns the file handle to the user process.

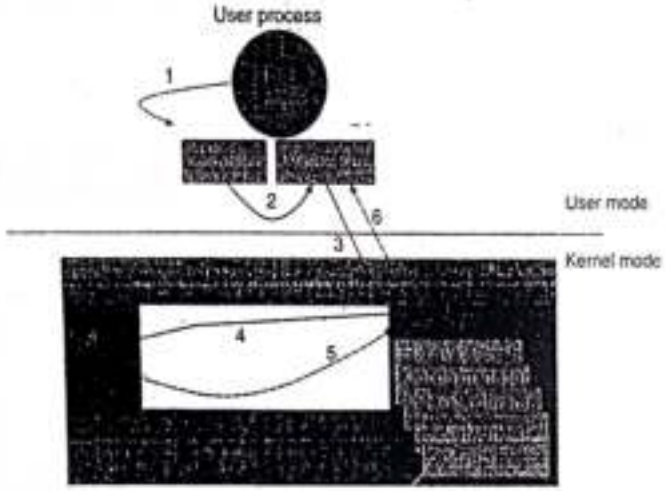


Figure 11.34: Opening a file in Windows

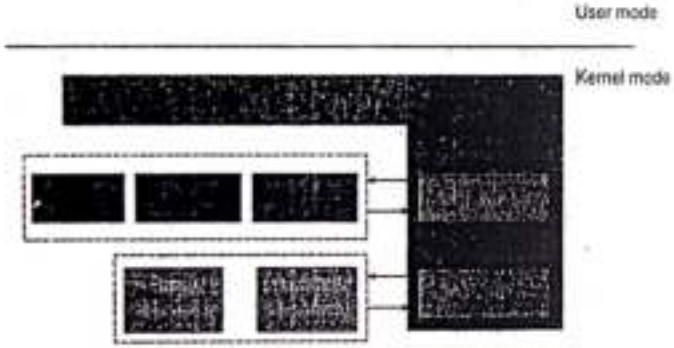


Figure 11.35: Two-layered driver system

It may be noted from Fig. 11.35 that Windows uses a multi-layered model of device drivers (file drivers and device drivers). The request for read/write operations on the file object is directed by the I/O manager to the file system. The file system calls the file derivers to perform the action on a virtual file. The virtual file is taken by the I/O manager and the associated physical device located, on which the I/O is desired. Having identified the device, the I/O manager calls its device driver to complete the I/O operation. As and when the I/O is complete, the I/O manager informs the user process about it.

FILL IN THE BLANKS

1. Human beings have a basic tendency to record _____.
2. Most of the ancient Indian scriptures were written on paper-like sheets of the bark of a _____ tree.
3. _____ is a named collection of data of a given length, stored on a media.
4. The information contained in a file can be _____ among the peer group.
5. An individual item within a record is called a _____.
6. The field that uniquely defines a record of a file is called a _____ _____ _____.
7. The arrangement of records in a file according to date and time is called _____ order.
8. On a magnetic tape, the gap between two blocks of data is called the _____ _____ gap.
9. The_____ sequential file is similar to a telephone directory.
10. In an index sequential file, extra storage space is required to store the _____ of the file.
11. In _____ file organisation, for every record, an address is generated by a _____ function.
12. In the Windows operating system the hard disk is divided into _____ drives.
13. A file in a hierarchical directory structure is identified by its name qualified by its _____ from the root of the directory.
14. A file name is generally composed of two parts—a base name and an _____.
15. The last sector on a track is kept reserved as an _____ area.
16. If an index sequential file has a blocking factor equal to 1 then the file is called a _____ file.
17. A file descriptor contains _____ data about a file.
18. A read count is used to track the number of _____ which are currently _____ the file.
19. In Linux, every file system registers itself with _____ _____ _____ during the booting of the OS.
20. In NTFS, similar files created by an application are tracked by a _____ _____ mechanism.
21. Windows uses multi _____ model of device drivers.
22. A file name in Unix can have a maximum of _____ characters.
23. In Unix, an ordinary file can be of type: _____ or _____.
24. A device in Unix is represented by a _____ file.
25. A pipe is also treated as a _____ file.
26. Unix assigns an 'inode' to every _____.
27. A link to a file from a directory can be of two types— ____ and ____ link.
28. A file system in Unix can be attached by a _____ command.

MULTIPLE CHOICE QUESTIONS

1. The computer is the source of
   a. Generation of information
   b. Processing of information
   c. Storage of information
   d. All of the above
2. Information stored on permanent media is called
   a. Data
   b. Information
   c. File
   d. None of the above
3. A file is a named repository of
   a. Volatile data
   b. Persistent data
   c. Both of the above
   d. None of the above

Security and Protection in OS

Introduction

- Overview of Security and Protection
- Security Attacks
- Formal Aspects of Security
- Encryption
- Authentication and Password Security
- Protection Structures
- Protection Domain
- Capabilities
- Classifications of Computer Security
- Case Studies in Security and Protection

Overview of Security and Protection

- A *threat* is a possible form of interference
  - *Security:* threats to resources from *nonusers*
  - *Protection:* threats from users

Table 15.1 Terminology Used in Security and Protection of Information

Term	Explanation
Authentication	Authentication is verification of a user's identity. Operating systems most often perform authentication *by knowledge*. That is, a person claiming to be some user X is called upon to exhibit some knowledge shared only between the OS and user X, such as a password.
Authorization	Authorization has two aspects: 1. Granting a set of access privileges to a user; for example, some users may be granted read and write privileges for a file, while others are granted read-only privileges. 2. Verifying a user's right to access a resource in a specific manner.

Overview of Security and Protection (continued)

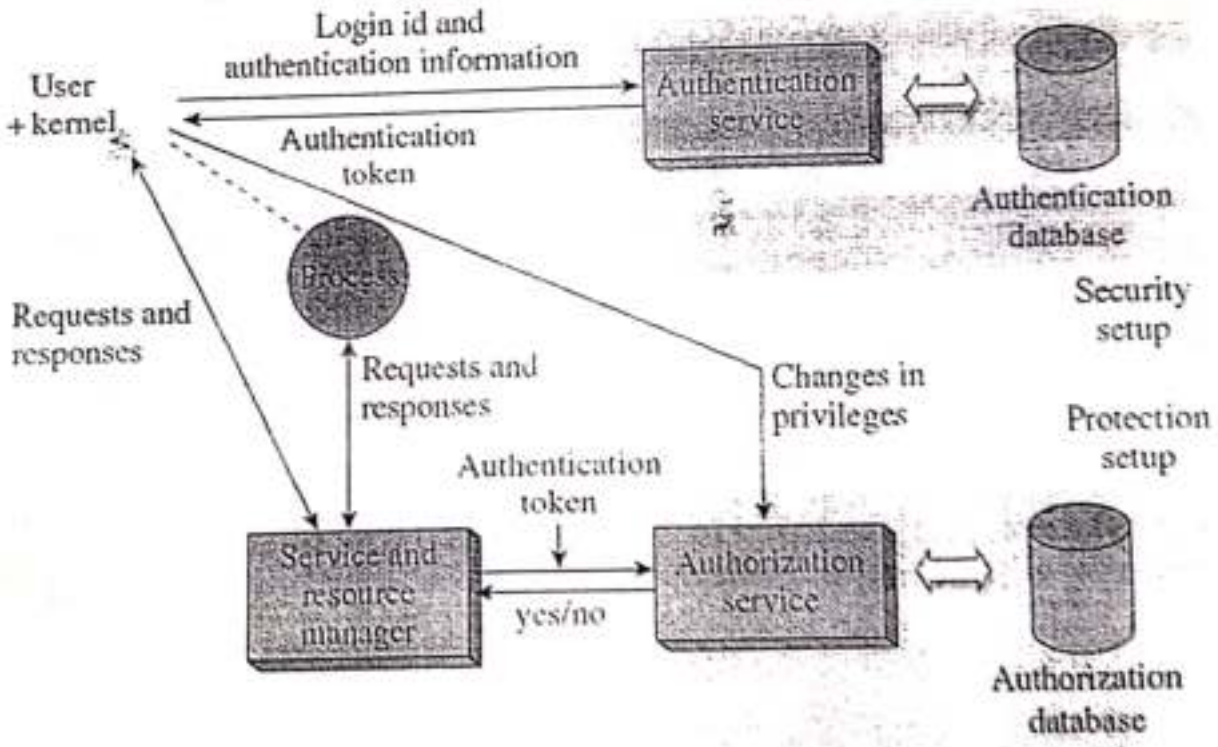


Figure 15.1 Generic security and protection setups in an operating system.

Overview of Security and Protection (continued)

Table 15.2 Policies and Mechanisms in Security and Protection

Security	• *Policy:* Whether a person can become a user of the system. The system administrator employs the policy while registering new users. • *Mechanisms:* Add or delete users, verify whether a person is a registered user (i.e., perform authentication), perform encryption to ensure confidentiality of passwords.
Protection	• *Policy:* The file owner specifies the authorization policy for a file. It decides which user can access a file and in what manner. • *Mechanisms:* Set or change authorization information for a file. Check whether a file processing request conforms to the user's privileges.

Goals of Security and Protection

Table 15.3 Goals of Computer Security and Protection

Goal	Description
Secrecy	Only authorized users should be able to access information. This goal is also called *confidentiality*.
Privacy	Information should be used only for the purposes for which it was intended and shared.
Authenticity	It should be possible to verify the source or sender of information, and also verify that the information has been preserved in the form in which it was created or sent.
Integrity	It should not be possible to destroy or corrupt information, for example, by erasing a disk.

- Only *privacy* is exclusively a protection concern
  - *Controlled sharing* based on *need-to-know* principle

Security and Protection Threats

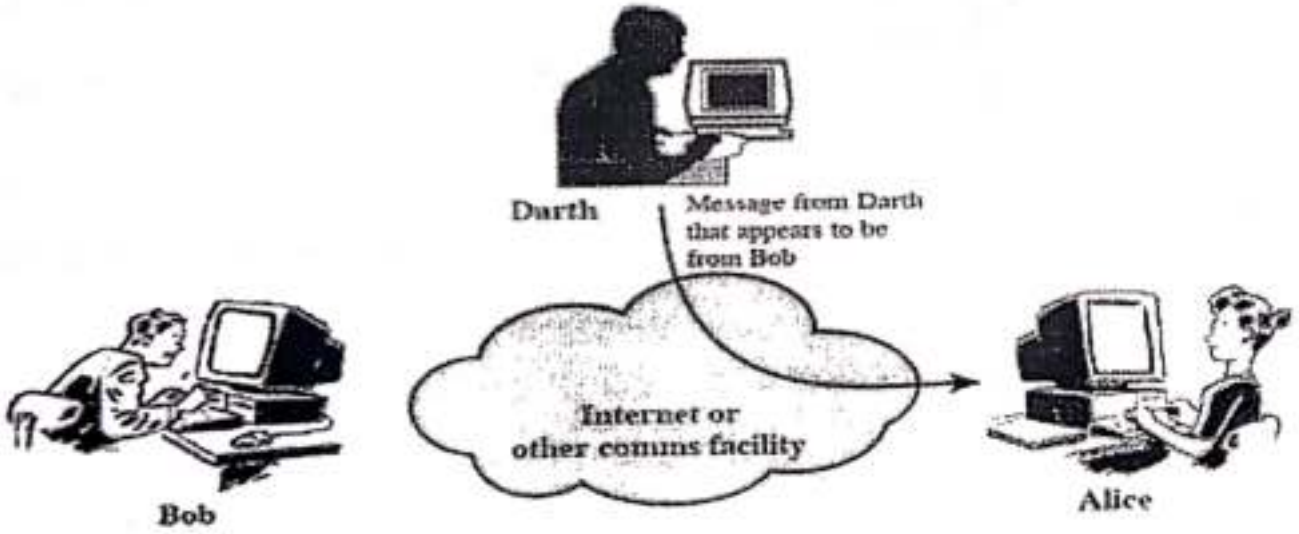
- Examples of security threats:
  - Threats raised by data and programs downloaded from the Internet
- Examples of protection threats:
  - Illegal access to a resource or a service by a process
  - An attempt to tamper with messages
- Security threats can arise more easily in a distributed OS

Security Attacks

- *Security attack:* attempt to breach security of a system
- Terminology: *security attacks, adversary, intruder*
- Two common forms of security attacks are:
  - *Masquerading:* assume identity of a registered user through illegitimate means
  - *Denial of service* (DoS)
    - Prevent users from accessing resources for which they possess access privileges
      - *Network DoS attack, distributed DoS attack*
- Other types of attacks:
  - Message *eavesdropping*
  - Tampering with messages

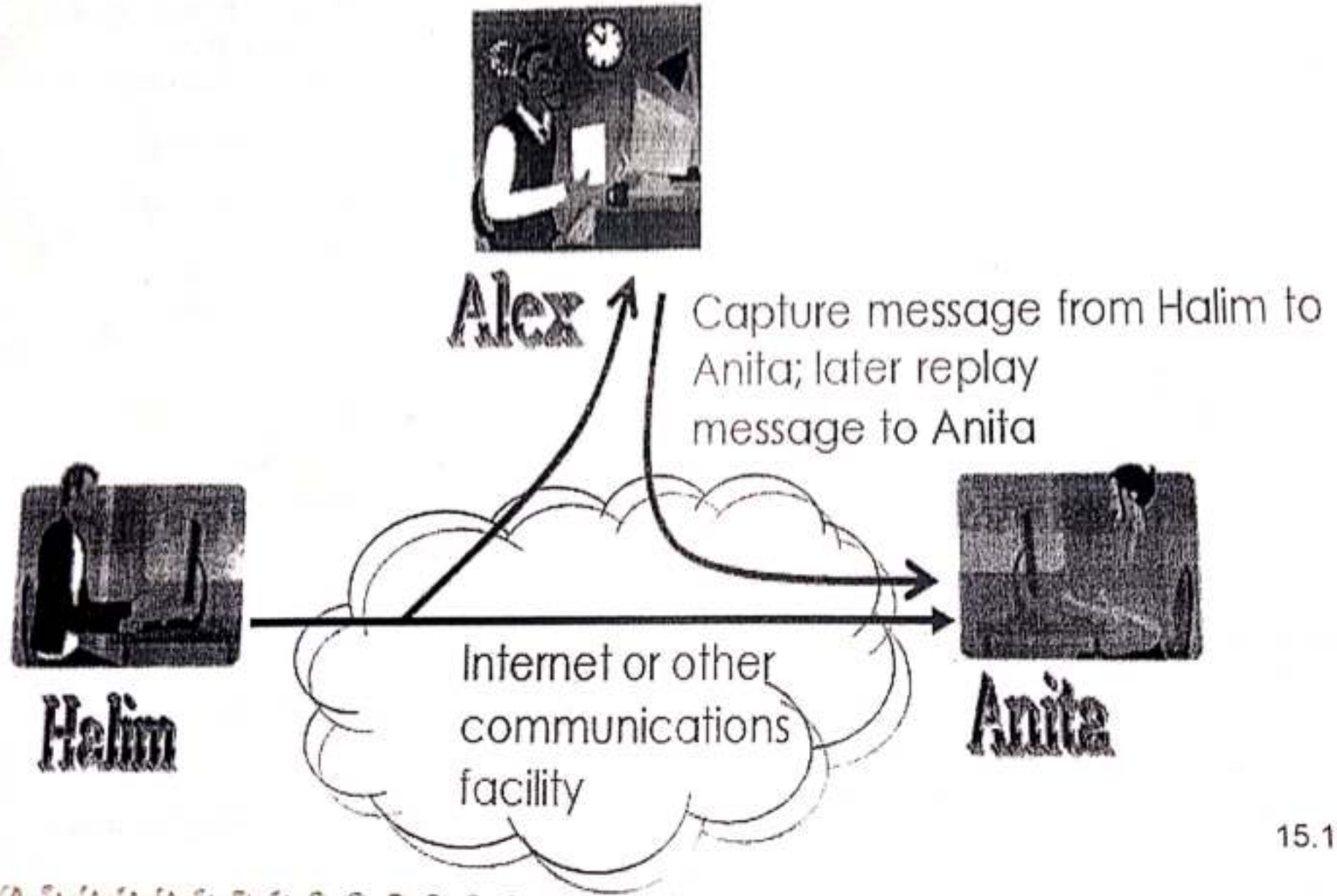
Masquerade

A masquerade is a type of attack where the attacker act as an authorized user system in order to gain access to it or to gain greater privileges than they are authorized for.



(a) Masquerade

Replay attack



Trojan Horses, Viruses, and Worms

- *Trojan horses*, *viruses*, and *worms* contain code that can launch a security attack when activated

Table 15.4 Security Threats through Trojan Horses, Viruses, and Worms

Threat	Description
Trojan horse	A program that performs a legitimate function that is known to an OS or its users, and also has a hidden component that can be used later for nefarious purposes like attacks on message security or masquerading.
Virus	A piece of code that can attach itself to other programs in the computer system and spread to other computer systems when programs are copied or transferred.
Worm	A program that spreads to other computer systems by exploiting security holes in an OS like weaknesses in facilities for creation of remote processes.

Trojan Horses, Viruses, and Worms (continued)

- A virus typically sets up a *back door* that can be exploited for a destructive purpose at a later date
  - E.g., executable virus, boot-sector virus, e-mail virus
- *Worms* may spread using *buffer overflow* technique
- Measures to foil security attacks:
  - Using caution while loading new programs into a computer
  - Using antivirus programs
  - Plugging security holes

Encryption

- Encryption: application of an algorithmic transformation to data
  - *Cryptography* deals with encryption techniques
  - *Plaintext* is transformed to *encrypted/ciphertext* form
  - Confidentiality provided through encryption also helps to verify integrity of data
  - Two types: *symmetric* and *asymmetric*

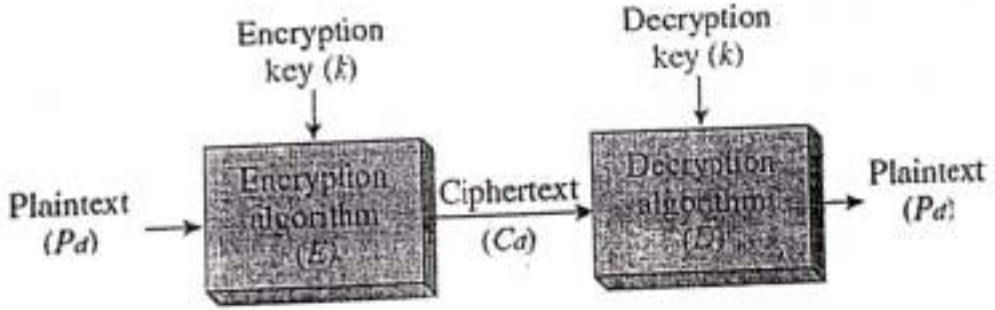
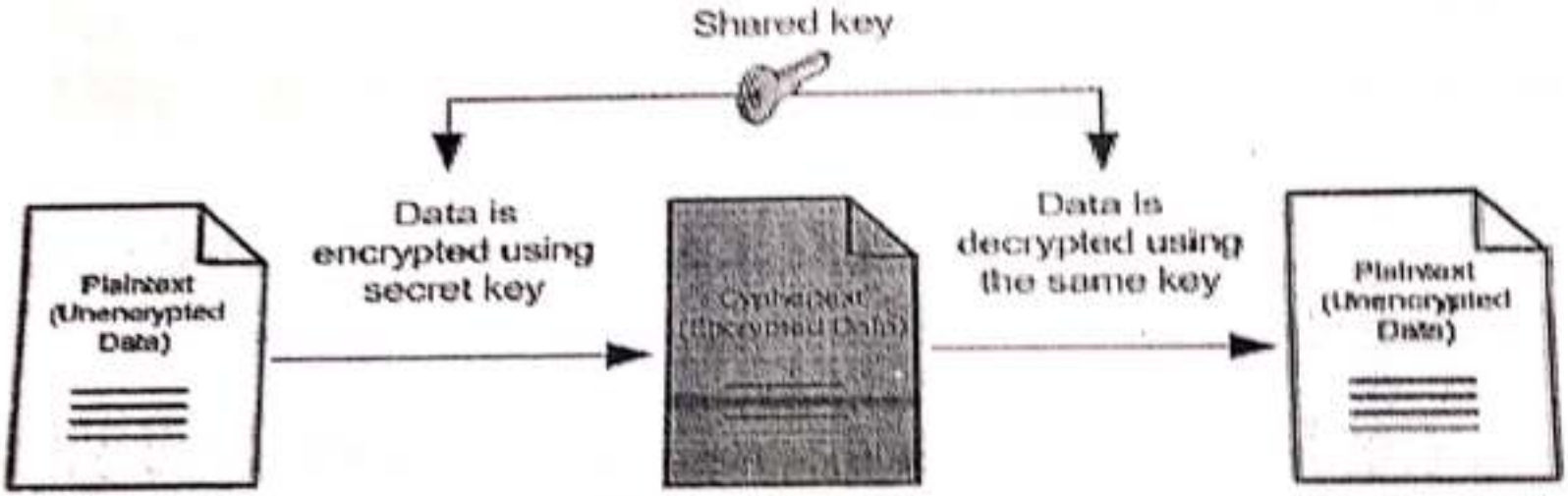


Figure 15.3 Symmetric encryption of data *d*.

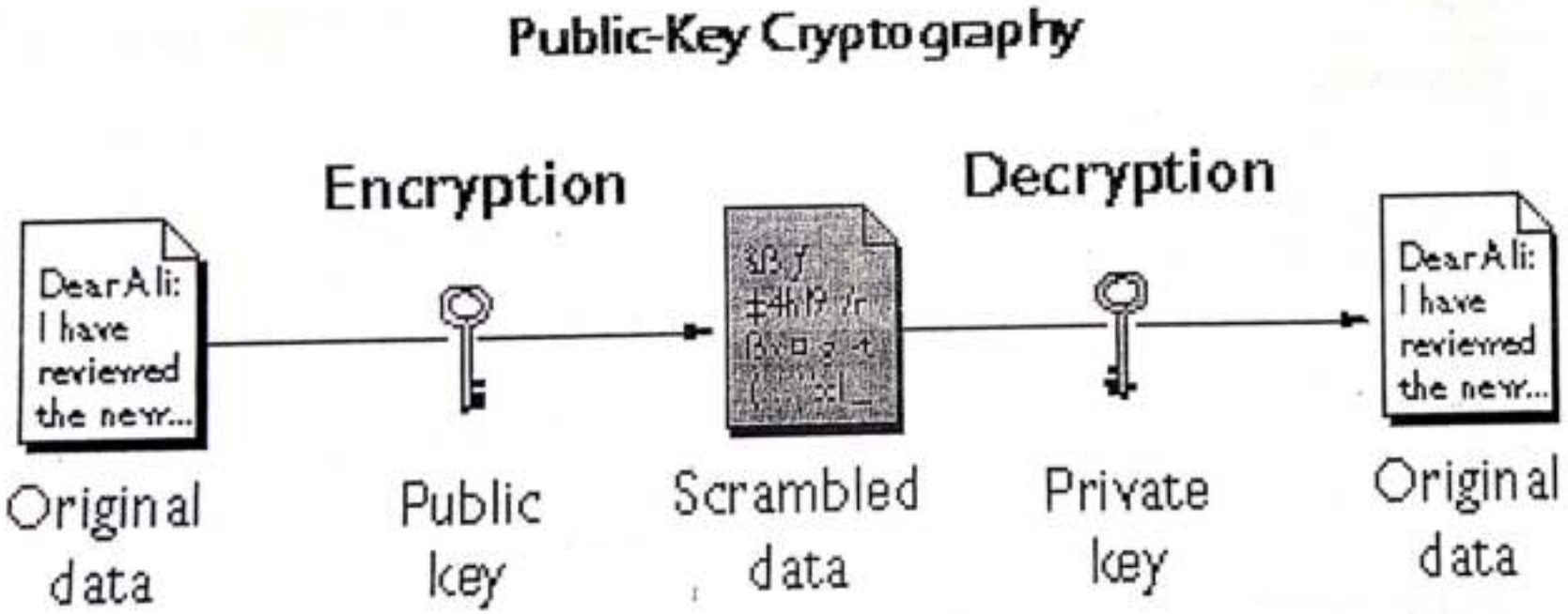
Symmetric key encryption

- It is also called secret key encryption in which sender and receiver share a common secret key for encrypting and decrypting the message



Asymmetric key encryption

- In asymmetric key encryption message is encoded usin senders public key and decoded using senders private key which is shared with the recipient of the message.



Authentication and Password Security

- Authentication typically performed using passwords

Table 15.6 OS Techniques for Defeating Attacks on Passwords

Technique	Description
Password aging	Encourage or force users to change their passwords frequently, at least once every 6 months. It limits the exposure of a password to intruder attacks.
System-chosen passwords	A system administrator uses a methodology to generate and assign *strong* passwords to users. Users are not allowed to change these passwords. An intruder would have to use an exhaustive attack to break such passwords.
Encryption of passwords	The encrypted form of passwords is stored in a system file; however, the ciphertext form of passwords is visible to all users in the system. An intruder can use one of the attacks described in Section 15.4.1 to find E_k, or launch an exhaustive attack to crack an individual user's password.
Encrypt and hide password information	The encrypted form of passwords is not visible to any person within or outside the system. Hence an intruder cannot use any of the attacks described in Section 15.4.1.

Protection Structures

- *Protection structure:* classical name for the authorization database
  - Access *privilege* (for a file): right to make a specific form of access to the file
  - *Access descriptor:* representation of a collection of access privileges for a file
    - *Access control information* (for a file): collection of access descriptors

Access Control Matrix

- An access control matrix is a protection structure that provides efficient access to:
  - Access privileges of users to various files
  - Access control information for files

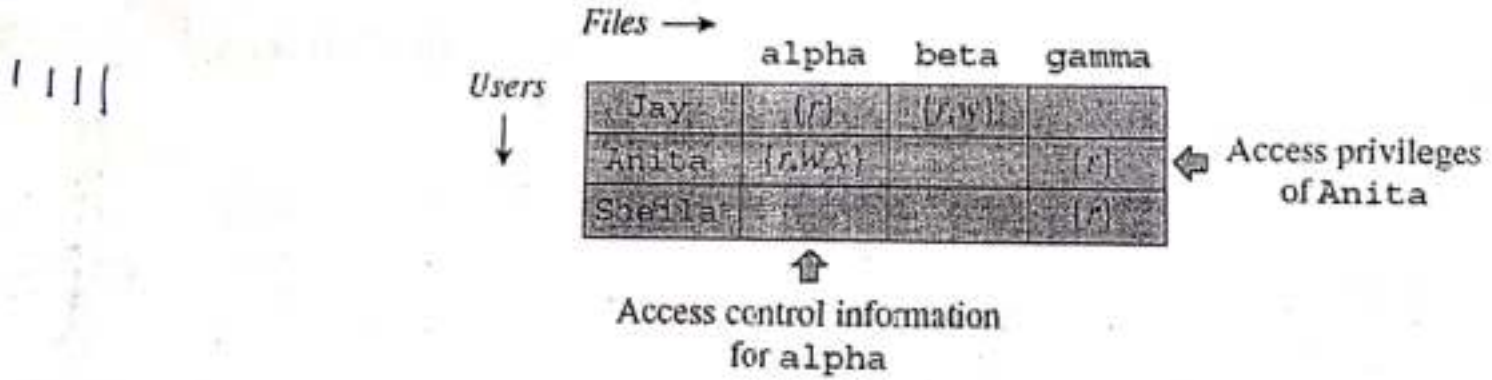


Figure 15.7 Access control matrix (ACM).

Access Control Lists (*ACLs*)

- ACL of a file is a representation of its access control information
  - Contains the non-null entries that the file's column would have contained in the ACM

File name	*Access control list (ACL)*
alpha	{(Jay, {r}), (Anita, {r, w, x})}
beta	{(Jay, {r, x})}
gamma	{(Anita, {r}), (Sneila, {r})}

Figure 15.8 Access control lists (ACLs).

Protection Domain

- Use of access control matrix, ACL, or C-list used to confer access privileges on users achieves secrecy
- *Privacy* goal requires that information should be used only for intended purposes
  - Access privileges granted to a *protection domain*
  - A process operates "within" a protection domain
  - It can switch domains during operation

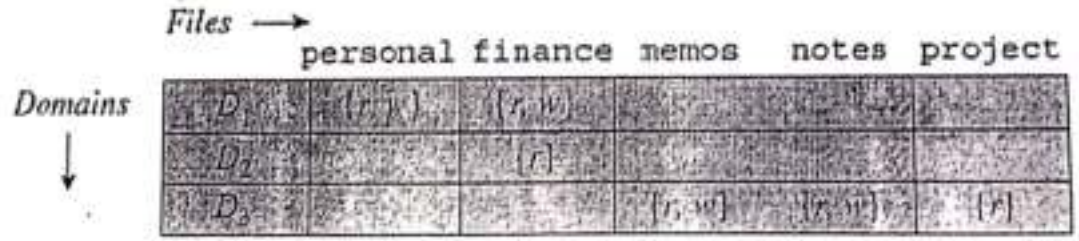


Figure 15.10 Protection domains.