S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Syllabus of Operating system		
		1. Basic Introduction - types, process, diagram, System call		
FIFO				
SJF		2. Process scheduling		
PR		3. Process synchronization - semaphore.		
Round Robin		4. Deadlock & threads - Banker's Algo		
Virtual Mem		5. Memory Management - Paging, Segmentation, Fragmentation		
		6. Disk scheduling		
		7. UNIX commands		
		8. File Mgmt & Security		

Scanned by TapScanner

An Operating System performs all the basic tasks like managing files, processes and memory. The manager of all the resources i.e resource manager

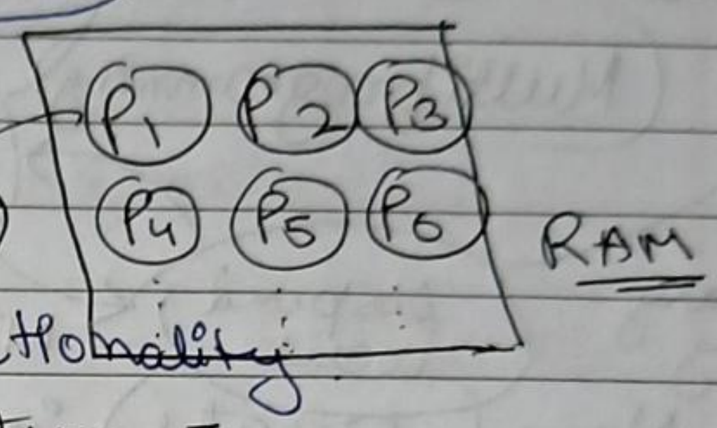
27/05

Types of Operating system → interface b/w user & machine

1) Batch → (IDLENESS)
2) Multiprogrammed. – Non Preemptive
3) Multitasking – Preemptive (Faster Response)
4) Real time OS → (RESPOSIVENESS)
5) Distributed.
6) Clustered (one network)
7) Embedded.

Hard
Soft

↳ works only on fixed functionality



Batch Operating System –
There is an operator which takes similar jobs having the same requirement and group them into batches.

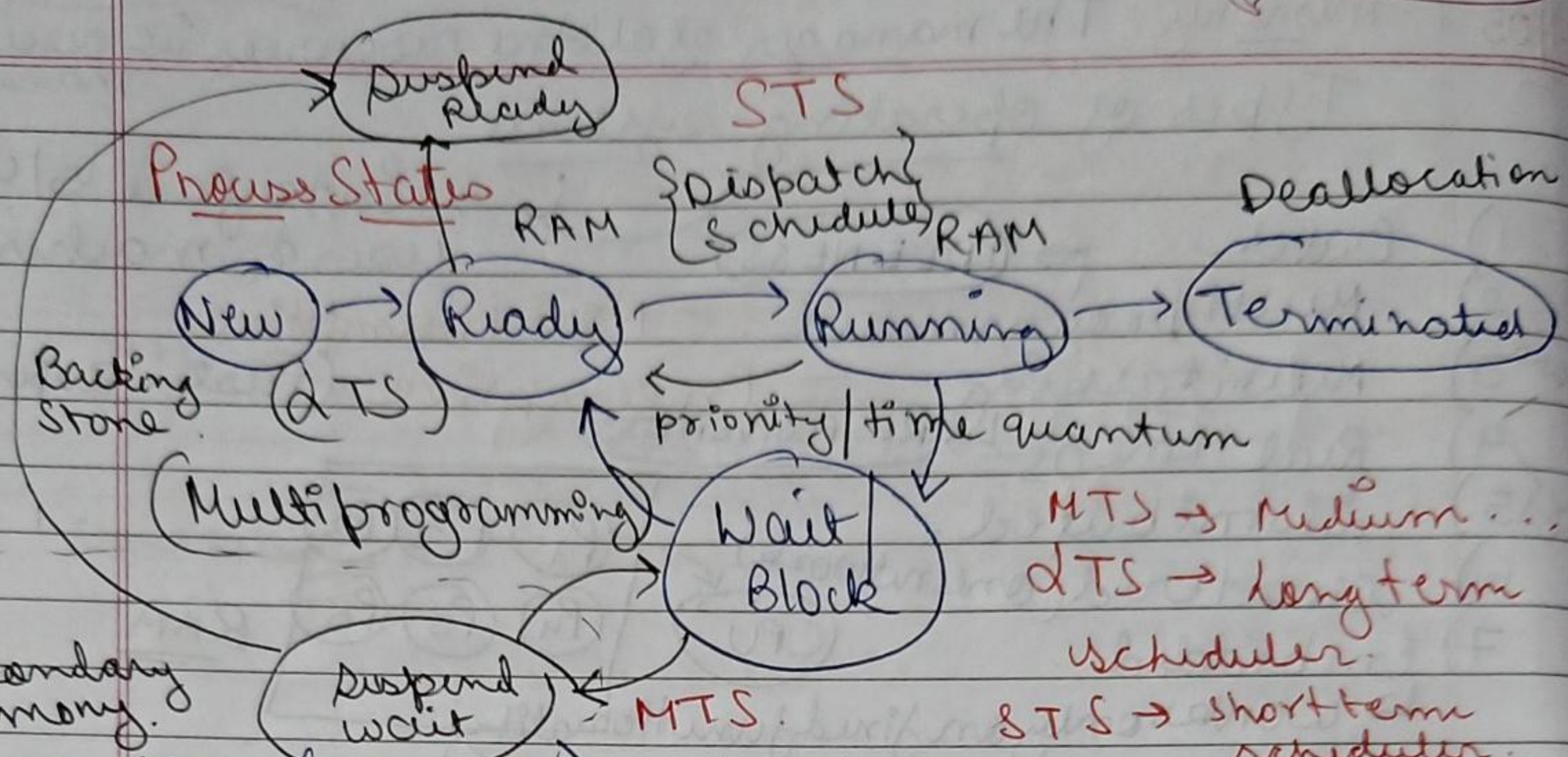
Time Sharing Operating System (Multitasking)
Each task is given some time to execute so that all the tasks work smoothly.

Distributed Operating System
Various autonomous interconnected computers communicate with each other using a shared communication network.
Processing Environment.

Network Operating System
These systems run on a server and provide the capability to manage data, users, groups, security, applications and other networking functions.

Real time OS:
The time interval required to process and respond to inputs is very small.
Hard – strict – no delays.
Soft – not critical.

Scanned by TapScanner



- New (Create) :- the process is about to be created but not yet created.
- Ready :- The process here is ready to run and is waiting to get the CPU time for its execution.
- Run :- The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- Blocked / wait :- Whenever the process requests access to I/O or needs input from the user or needs access to a critical region it enters the blocked or wait state.
- Suspend wait / suspend blocked.
- Suspend ready.
- Terminated or completed :- process is killed as well as PCB is deleted.

Scanned by TapScanner

Which command is used to assign only Read permission to all three categories of file 'note'

A) Chmod a-rw.
B) Chmod go+r note
C) Chmod ugo=r note
D) Chmod u+r, g+r, o-r note.

'Chmod ugo+rw note' command can be represented in Octal notation as

A) Chmod 555 note
B) Chmod 666 note
C) Chmod 333 note
D) Chmod 444 note

r w x r w - r - x
u g o

u r 4
g w 2
o x 1

Suppose you have a file 'f' whose contents are:-
1234567890abcdefghij
here 'lseek' is used two times sequentially
lseek (n, 10, SEEK_CUR);
lseek (n, 5, SEEK_SET); n is file descriptor
After applying lseek two times, what will be current position of R/W head?

→ 5 (index starts from 0)

Scanned by TapScanner

27/05 ↱ interface between a process and an operating system.

System Call

- A system call is a request from computer software to an operating system's Kernel.
- The application program interface (API) connects the operating system's functions to user program.

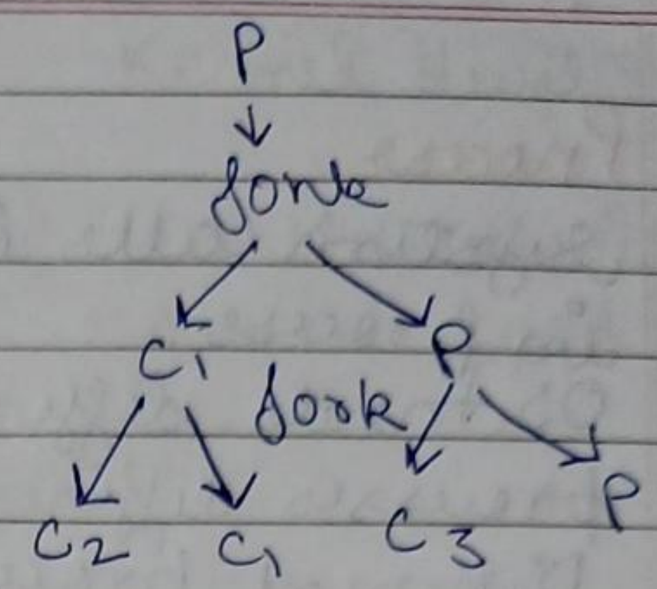
- File Related – file manipulation
  Open(), Read(), Write(), Close(), Createfile etc.
- Device Related – responsible for device manipulation
  Read, Write, Reposition, ioctl, fcntl
- Information – handle info & it transfer b/w the OS
  getPid, attributes, getsystem time and data
- Process control – deal with processes
  Load, Execute, abort, fork, wait, signal, allocate etc.
- communication – interprocess communication
  Pipe(), create/delete connections, shmget()

Fork()
- 0 child
- +1 +ve → parent
- −1 child ✗]

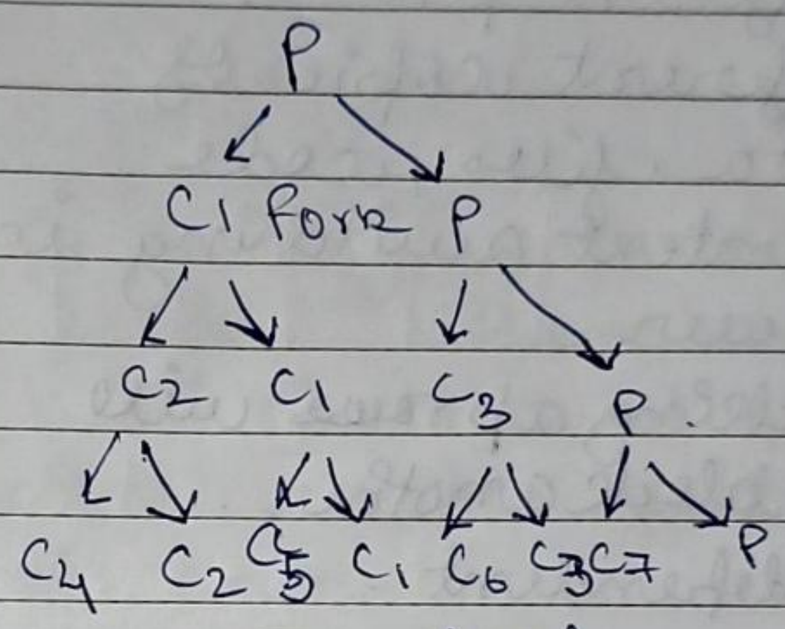
Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process)
It takes no parameters and returns an integer value. Below are different values returned by fork().
Negative value: creation of child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value:- Returned to parent or caller. The value contains process ID of newly created child process.

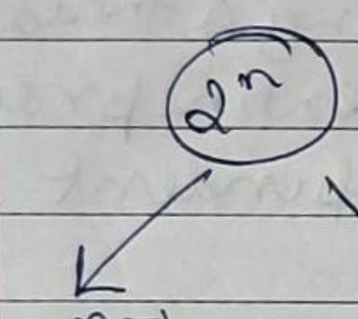
Scanned by TapScanner

```
main() {
 fork();
 fork();
 printf ("hello");
}
```




```
fork
fork
fork
pf ("hello")
```

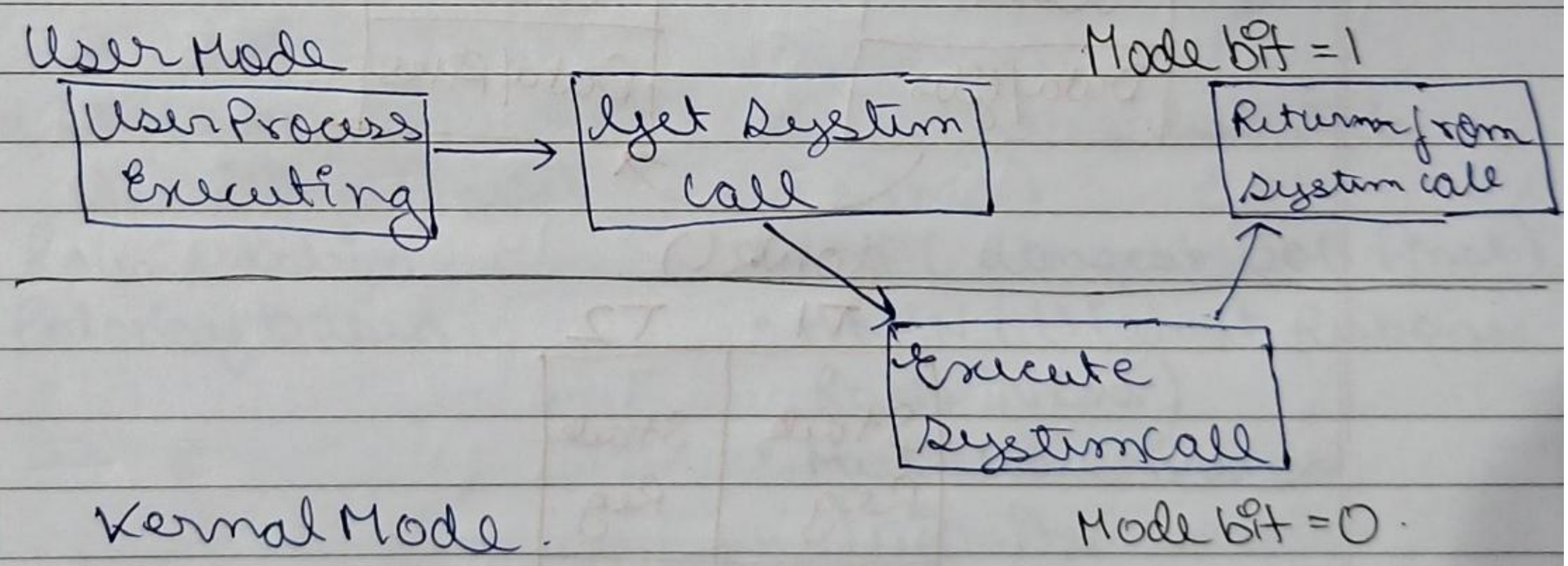




7 - child

1 - parent.

User Mode VS Kernal Mode



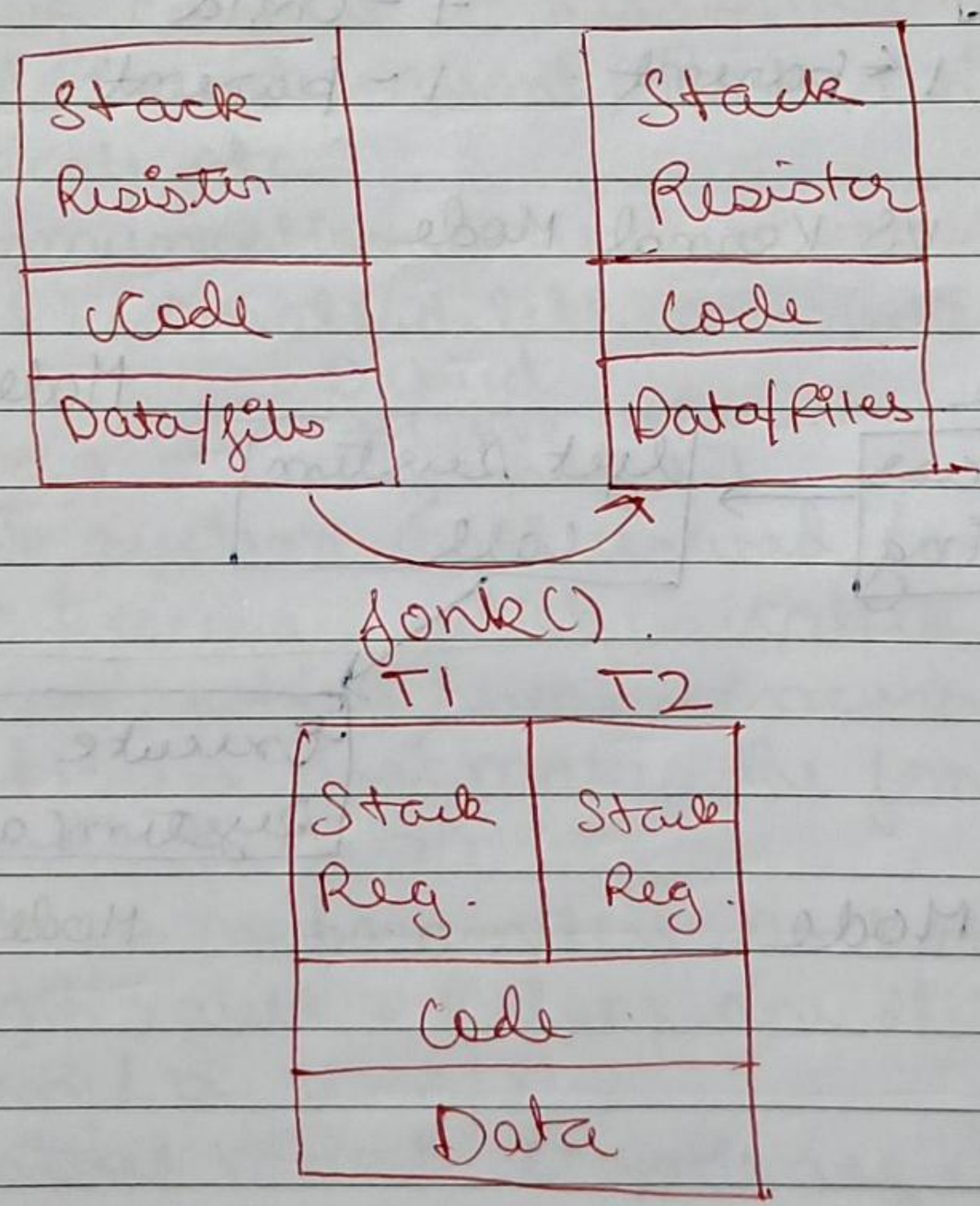
Scanned by TapScanner

classmate

Date
Page

29/5/2022.

Process	Threads
1. System calls involved in process.	1. There is no system call involved.
2. OS treats different processes differently.	2. All user level threads treated as single task for OS.
3. Different process have different copies of data, files, code.	3. Threads share same copy of code & data
4. Context switching is slower.	4. Context switching is faster.
5. Blocking a process will not block another.	5. Blocking a thread will block entire process
6. Independent.	6. Independent.



Scanned by TapScanner

Process { CPU bound / I/O bound.

User level Thread	Kernel level Thread.
1) User level Threads are managed by user level library.	1) Kernel level threads are managed by OS.
2) User level threads are typically fast.	2) Kernel level threads are slower than user level.
3) context switching is faster.	3) context switching is slower.
4) If one user level threads perform blocking operation then entire process get blocked.	4) If one kernel level thread blocked, no affect on others.

Scheduling Algorithms.

Pre-Emptive.	Non-Pre-Emptive.
• SRTF (Shortest remaining time first)	• FCFS (First come first serve).
• LRTF (Longest Remaining Time first)	• SJF (Shortest Job first)
• Round Robin.	• LJF (Longest Job first)
• Priority based.	• HRRN (Highest Response Ratio Next)
	• Multilevel Queue.
	• Priority.

CPU Scheduling

- Arrival time: The time at which process enter the Ready Queue or State.
- Burst time: Time required by a process to get execute on CPU.
- Completion time: The time at which process complete its execution.
- Turn Around time: Completion time - Arrival Time.
- Waiting time: Turn Around Time - Burst Time.
- Response time: {The time at which a process get CPU first time} - {Arrival Time}.

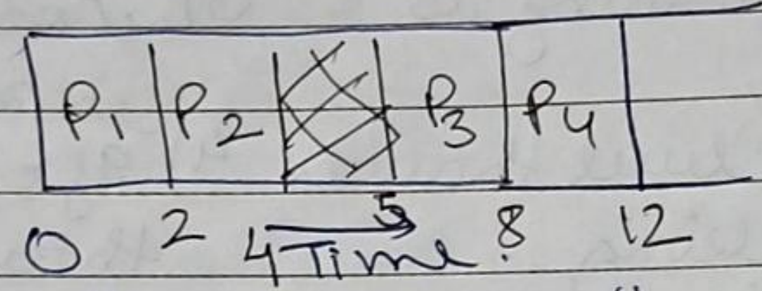
Scanned by TapScanner

First come First Serve

FCFS

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4.	6	4	12	6	2	2

Gantt Chart



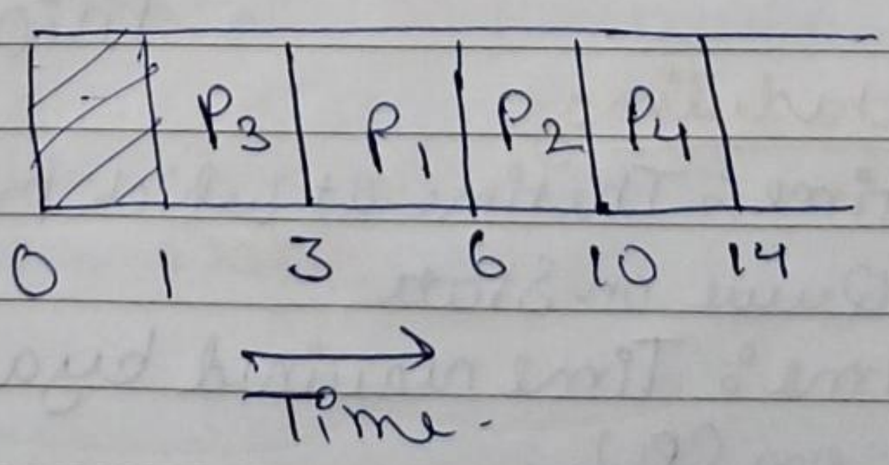
Criteria : "Arrival Time"
Mode : "Non-Preemptive"

Shortest Job First (SJF).

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4.	4.	4.	14.	10	6	6.

Criteria : "Burst Time"
Mode : "Non-preemptive"

Gantt Chart



Scanned by TapScanner

Shortest Remaining Time First (SRTF).

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT.
P1	0	~~5 4 3 2~~ 1 0	9	9	4	0
P2	1	~~3~~ ~~2~~ 1 0	4	3	0	0
P3	2	~~4~~ 0	13	11	7	7
P4	4.	1 0	5	1	0	0

Criteria : "Burst Time"
Mode : "Preemptive"
TAT = CT - AT.
WT = TAT - BT.
RT = {CPU first time - AT}

Gantt Chart.

P1	P2	P2	P2	P4	P1	P1	P1	P1	P3

0 1. 2 3 4 5. 6 7 8 9

Time.

29/5

Process	Arrival Time	Burst time.
P1	0	7
P2	1	4
P3.	2	8.

The Gantt Chart for Preemptive SJF Scheduling algorithm is

→

P1	P2	P1	P3

0 1 5 11 19.

Scanned by TapScanner

Round Robin (RR) CPU scheduling Algorithm.

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P1	0	5	12	12	7	0
P2	1	4	11	10	6	1
P3	2	2	6	4	2	2
P4	4	1	9	5	4	4

Given
TQ = 2.

Ready Queue: ~~P1~~ | ~~P2~~ | ~~P3~~ | ~~P1~~ | P4 | P2 | P1

Running Queue: P1 | P2 | P3 | P1 | P4 | P2 | P1

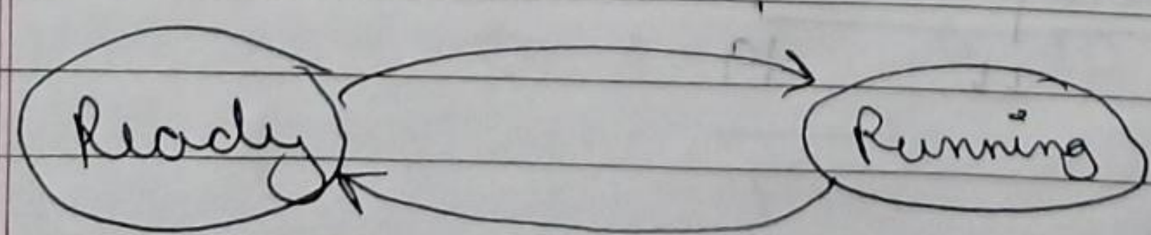
0 2 4 6 8 9 11 12

(Gantt Chart) → time

6 → Context switching.

Criteria : "Time Quantum"
Mode : "Preemptive"

TAT = CT − AT.
WT = TAT − BT.
RT = {CPU first time − AT.}



Sequence of processes in R/Q.

Scanned by TapScanner

CPU idleness |
CPU usage.

- Pre-emptive Priority scheduling Algorithm.

Priority	Process No.	Arrival Time	Burst Time.	Completion Time.	TAT	WT.
10	P_1	0	5	12	12	7
20	P2	1	4	8	7	3
30	P3	2	2	4	2	0
40	P4	4.	1	5.	1	0.

★ higher the no.
higher the priority.

P_1	P_2	P_3	P_3	P_4	P_2	P_1

0 1 2 3 4 5 8 12.

Criteria : "Priority"
Mode : "Preemptive"

- Example of Min Burst Time (CPU & I/O both) in CPU scheduling.

Process	AT	Priority	CPU	I/O	CPU.
P1	0	2	1	5	3.
P2	2	3	3	3	1
P3	3	1 H.	2	3	1
P4.	3.	4 2	2.	4	1

4/18 ~CPU IDLENESS.

Mode : Pre emptive.
Criteria : Priority based.
Find CT of P_1, P_2, P_3, P_4

P_3 I/O P4 I/O

P_1		P_2	P_3	P_3	P_2	P_1	P_1	P_3	P_1	P_2	P_4	P_4		P_2		P_4

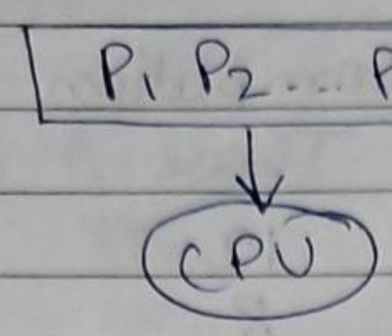
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18

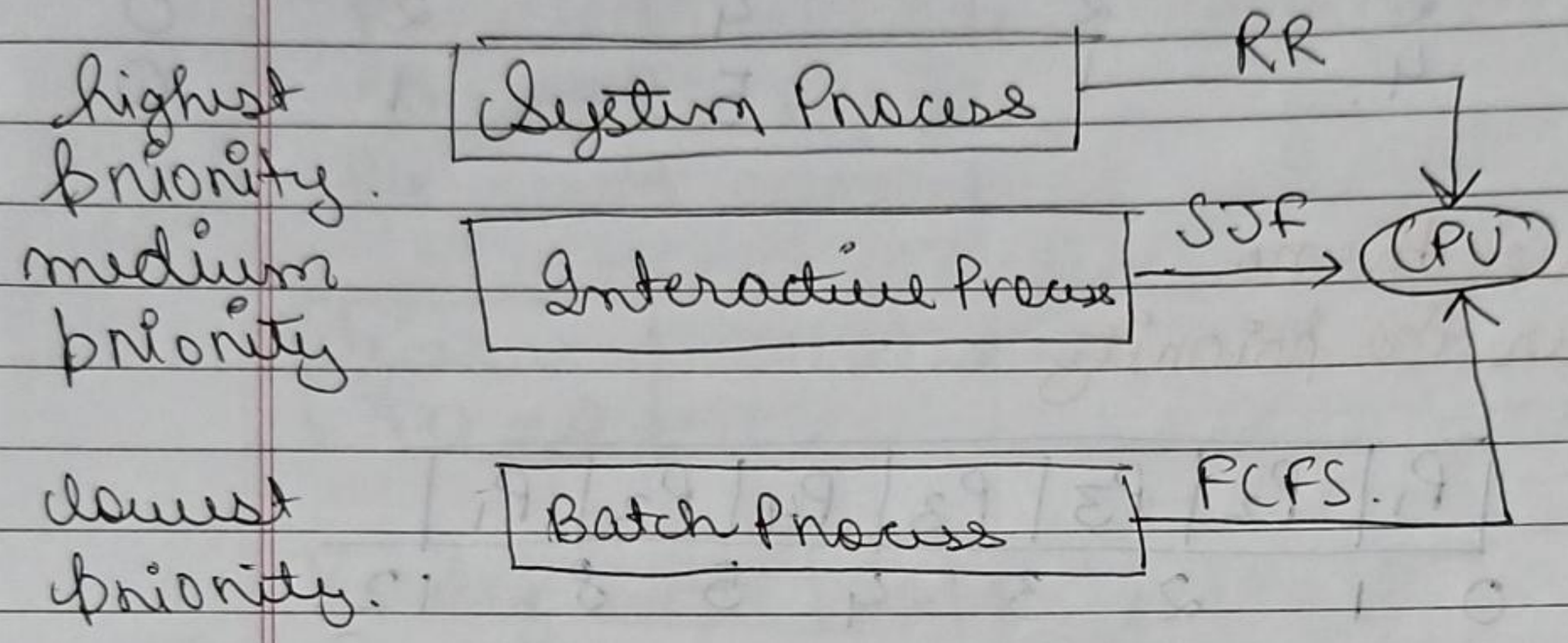
P_1 I/O. P_2 I/O.

14/18 ~CPU USUAGE.

Scanned by TapScanner

Multilevel Queue Scheduling.





It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs.

For example, a common division is a foreground (interactive) process and a background (batch) process.

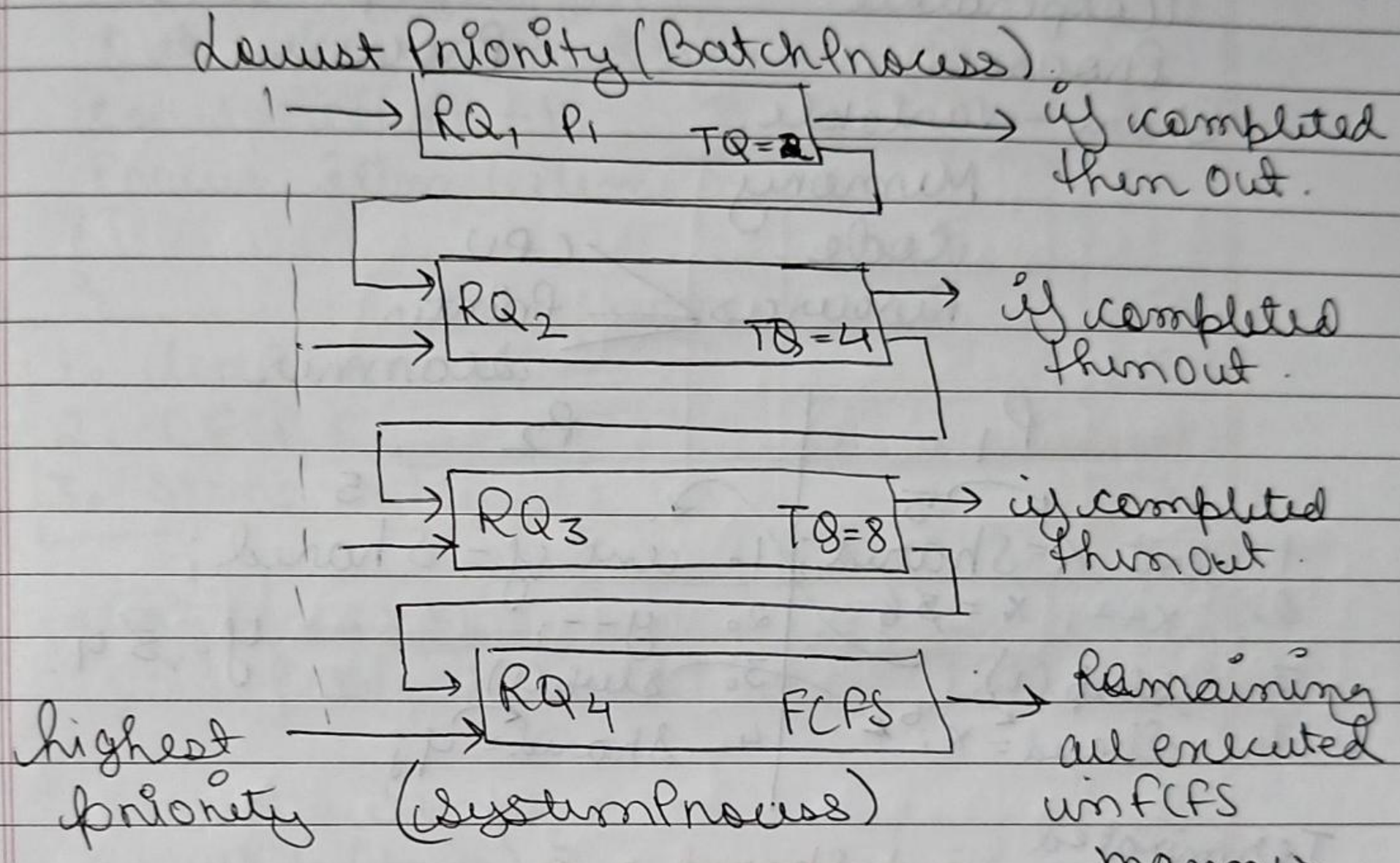
These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.

Disadvantage: lowest priority process might never get their chance.

Scanned by TapScanner

Multilevel Feedback Queue Scheduling (MLFQ) CPU scheduling.

↳ is like Multilevel Queue (MLQ) scheduling but in this processes can move between the queues.



Process synchronization

Process

- Cooperative Processes
  - Share - variable
    - Memory
    - Code
    - Resources ← CPU, Printer, Scanner
- Independent Processes

P_1	P_2
1. int x = Shared; (5)	1. int y = Shared; (5)
2. x++; x = ~~5~~ 6	2. y--; y = ~~5~~ 4
3. sleep(1);	3. sleep(1);
4. Shared = x; (6)	4. Shared = y;

Terminated int Shared = 5

Race condition

When more than one processes are executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition is known as a Race condition.

Scanned by TapScanner

30/05

Producer Consumer Problem

```
void consumer (void) {
int item c;
while(true){
 while (count==0);
 item c = Buffer(out);
 out = (out+1) mod n;
 count= count -1;
 Process_item (items);
 }
}
```

```
1. load Rc, m[count]
2. DECR Rc
3. Store m [count], Rc.
```

$n=8$

Buffer [0 .. n-1]

0	
1	x_2
2	x_3
3	x_4
4	
5	
6	
7	

count

~~3~~ ~~4~~ 2

```
int count=0;
void Producer (void)
{
 int itemp;
 while (true){
 Produce_item (itemp);
 while (count==n);
 Buffer [in]= itemp;
 in = (in+1) mod n;
 count = count+1;
 }
}
```

```
load Rp, m[count];
INCR Rp;
Store m[count], Rp;
```

Producer $I_1 I_2$ consumer $I_1 I_2$ Producer I_3 consumer I_3

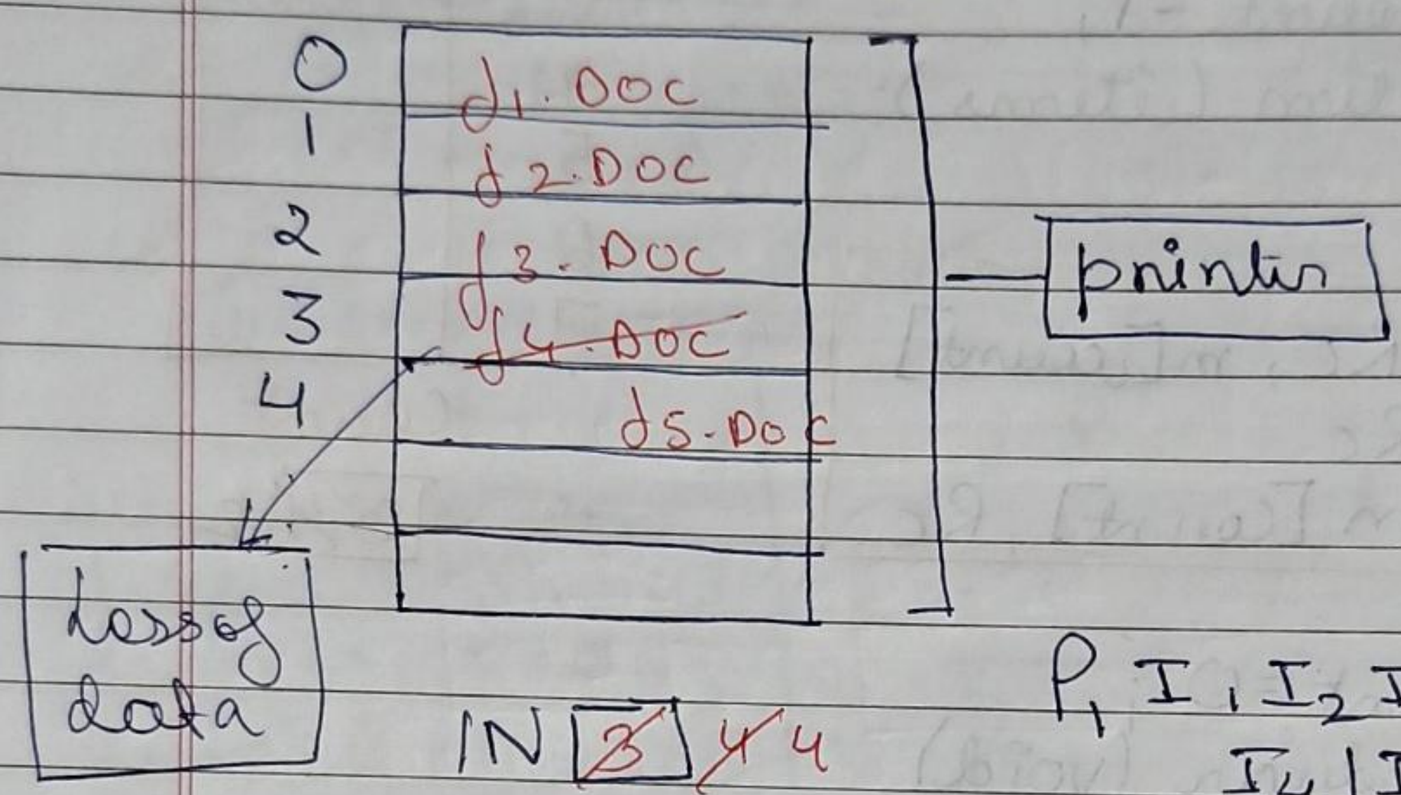
⇒ when the above code is executed in this manner the value of count obtained is 2 but the number of items is 3.

Scanned by TapScanner

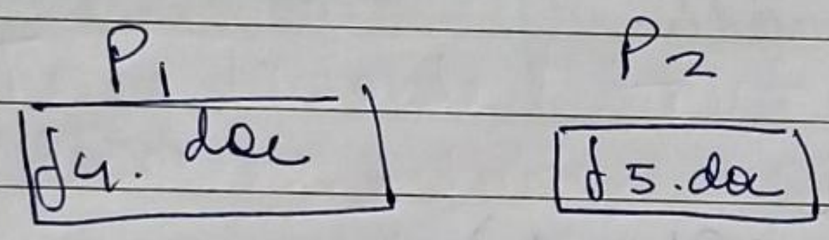
Printer Spooler Problem.

I1. Load Ri, m[in]
I2. Store SD[Ri], "f-N"
I3. INCR Ri
I4. Store m[in], Ri

Spooler Directory.



P_1 I_1 I_2 I_3 | P_2 I_1 I_2 I_3 I_4 | I_4.



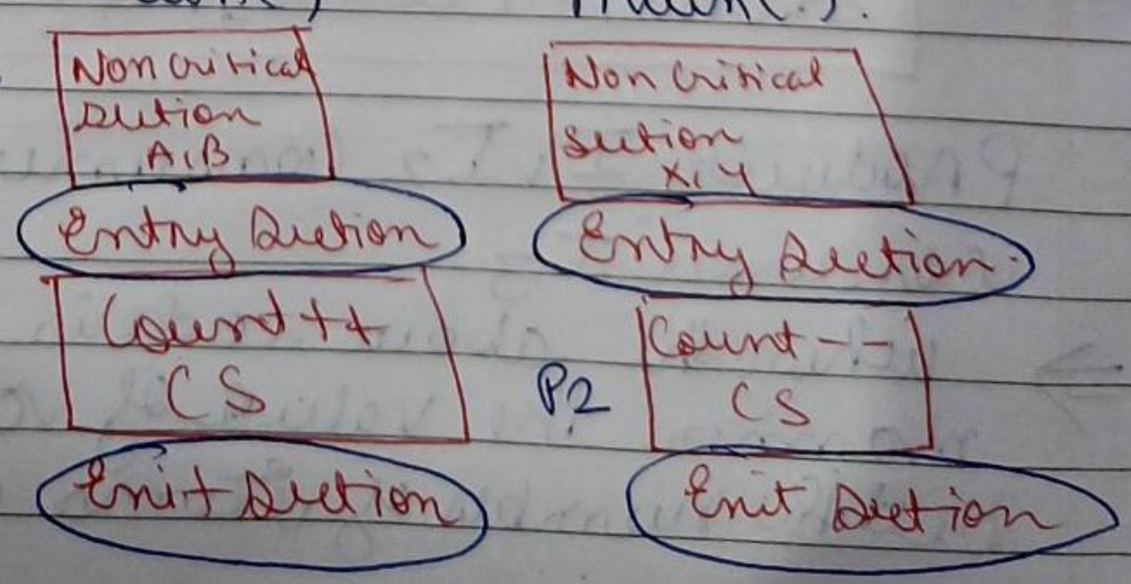
Critical Section Problem.

It's a part of the program where shared resources are accessed by various processes.

Place where shared variables, resources are placed.

```
#include _____ _____
```

→ Place where shared variables, resources are placed.



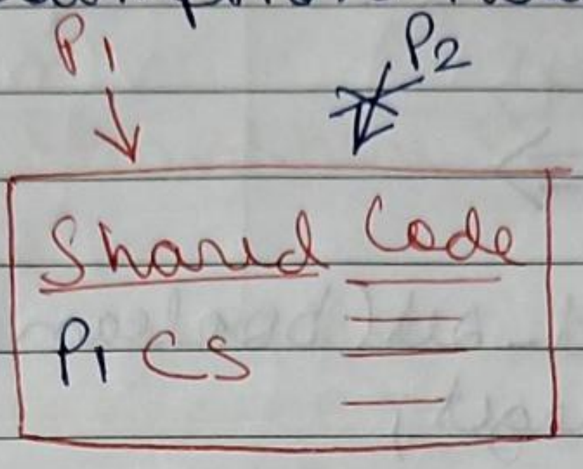
Scanned by TapScanner

Synchronization mechanism.

4 conditions.

Primary { 1) Mutual Exclusion.
2) Progress.

Secondary { 3) Bounded wait.
4) No assumption related to H/W speed.



5/6 Critical Section Solution using "lock"

```
do {
    acquire lock
        CS
    release lock
}
```

Entry Code

in online in test-and-set

1. while (LOCK==1);
2. LOCK = 1

3. Critical Section.

Exit Code

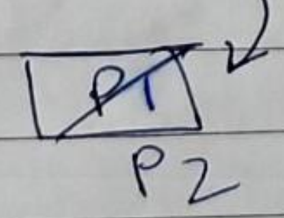
4. LOCK = 0

- Execute in User Mode.
- Multiprocess Solution.
- No Mutual Exclusion Guarantee.

LOCK = 0 → vacant
1 → full.

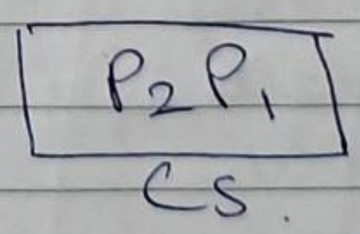
Case 1

P1 P2 ×
1234 | P2 ✓



Case 2

P1	P2
1 ✗ 23	1 2 3 CS.



Scanned by TapScanner

Critical Section Solution using "Test_and_Set" Instruction.

```
While(test_and_set(&lock));
[CS]
lock = false;
```

```
boolean testand_set(boolean *target) {
 boolean r = *target;
 *target = TRUE;
 return r;
}
```

lock	target	r
false	1000	false
1000		

Turn Variable (Strict Alteration).

- 2 process solution.
- Run in User Mode.

	Process "P_0"	Process "P_1"
Entry code	while(turn != 0);	while(turn != 1);
	critical section	critical section.
Exit code.	turn = 1;	turn = 0;

Scanned by TapScanner

Semaphore

Semaphore is an integer variable which is used in mutual exclusive manner by various concurrent cooperative processes in order to achieve synchronization

- Counting ($-\infty$ to $+\infty$)
- Binary (0, 1)

```
Down (Semaphore S)
{
 S.value = S.value - 1;
 if ( S.value <0){
  put process (PCB) in
  suspended list sleep();
 }else
    return;
}
```

```
Up(Semaphore S)
{
 S.value = S.value+1;
 if (S.value ≤0)
 {select a process
 from suspended
 list
  wake up();
 }
}
```

P(), Down, wait.
V(), Up, signal, post, Release.

Binary Semaphore

- 0
- 1

```
Down(Semaphore S){
if(S.value = = 1) {
   S.value=0;
 }else{
  Block this process
  and place in
  suspend list,
  sleep();
 }
}
```

```
Up(Semaphore S) {
if (suspend list is
    empty){
  S.value=1;
 }else {
select a process
from suspend
list and
 wakeup();
 }
}
```

S = X0
successful opⁿ.
S = ∅0 Block.
unsuccessful opⁿ.

Scanned by TapScanner

Practice Question on Binary Semaphore.

Each Process P_i $\{i = 1 \text{ to } 9\}$ execute the follo code.

Process P_{10} execute the foll. code.

```
repeat
  P(mutex)
    [CS]
  V(mutex)
forever.
```

mutex [1]

```
repeat
  V(mutex)
    [CS]
  V(mutex)
forever.
```

What is max. no. of processes that may present in CS at any part of time

$\boxed{P_1\ P_2\ P_3\ P_4\ P_5\ P_6\ P_7\ P_8\ P_9\ P_{10}}$

Solution of Producer consumer Problem using Semaphore.

Counting Semaphore → full = 0
→ Empty = N
→ No. of empty slots

```
Produce_item(item p);
down(Empty);
down(s);
[Buffer[IN] = item p;
 In = (In+1) mod n;]
up(s);
up(full);
```

```
Consumer
down(full);
down(s);
[itemC = Buffer[Out]
 Out = (Out+1) mod n]
up(s);
up(Empty);
```

N = 8

0	a	
1	b	
2	c	
3		
4		
5		
6		
7		

IN [3]

Out [0]

Scanned by TapScanner

Problem Statement

We need to ensure that when a producer is placing an item in the buffer, then at the same time, consumer should not consume any item. In the problem, buffer is the critical section.

Initialization of semaphores:-

mutex = 1

Full = 0 // initially, all slots are empty. Thus full slots are 0

Empty = n // All slots are empty initially

Solution for Producer

```
do{
// produce an item
wait(empty);
wait(mutex);
// place in buffer.
signal(mutex);
signal(full);
}while(true);
```

When producer produces an item then the value of "empty" is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item & thus the value of "full" is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed & consumer can access the buffer.

Solution for consumer:-

```
do{
wait(full);
wait(mutex);
// remove item from buffer
signal(mutex);
signal(empty);
// consumes item
}while(true),
```

As the consumer is removing an item from buffer, ∴ the value of "full" is reduced by 1 and the value in mutex is also reduced so that the producer can't access the buffer at this moment.

Scanned by TapScanner

Now, the consumer has consumed the item. thus increasing the value of "empty" by 1. The value of mutex is also increased so that producer can access the buffer now

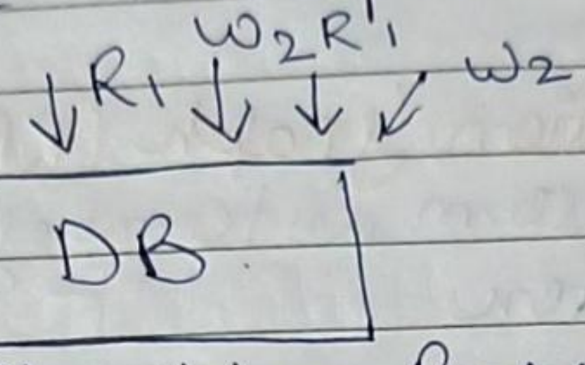
Reader-Writer Problem

```
int rc=0
semaphore mutex=1;
semaphore db=1;
void Reader (void){
while (true)
{
 down(mutex);
  rc=rc+1;
  if(rc==1)then down(db);
 up(mutex)
  [DB]
 down(mutex)
  rc=rc-1;
 if(rc==0)then up(db);
  up(mutex)
  Process-data
}
}
Void writer(void){
 while(true){
  down(db);
     [DB]
  up(db);
}
}
```

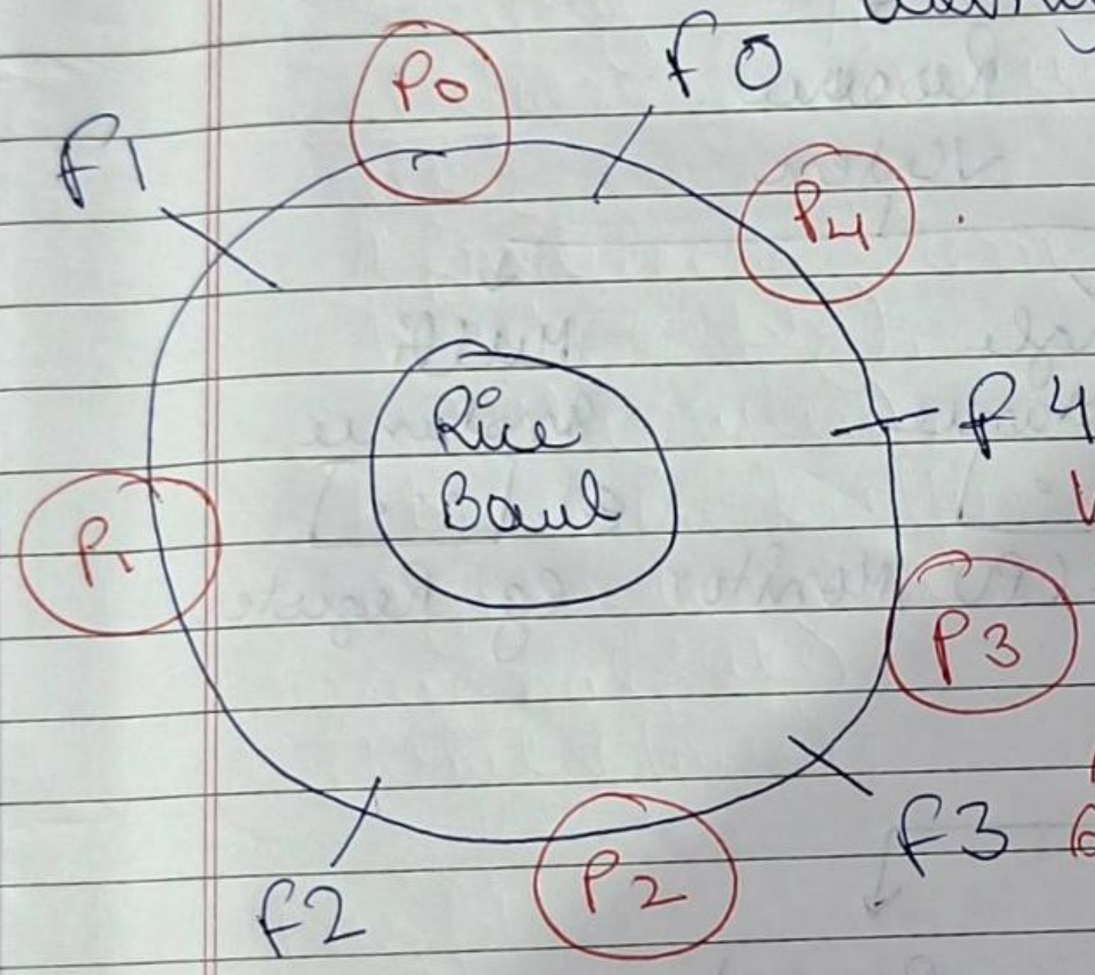
Scanned by TapScanner

Same Data

R - W → Problem
W - R → Problem
W - W → Problem
R - R → No problem



Dining Philosophers Problem and Solution using Semaphore

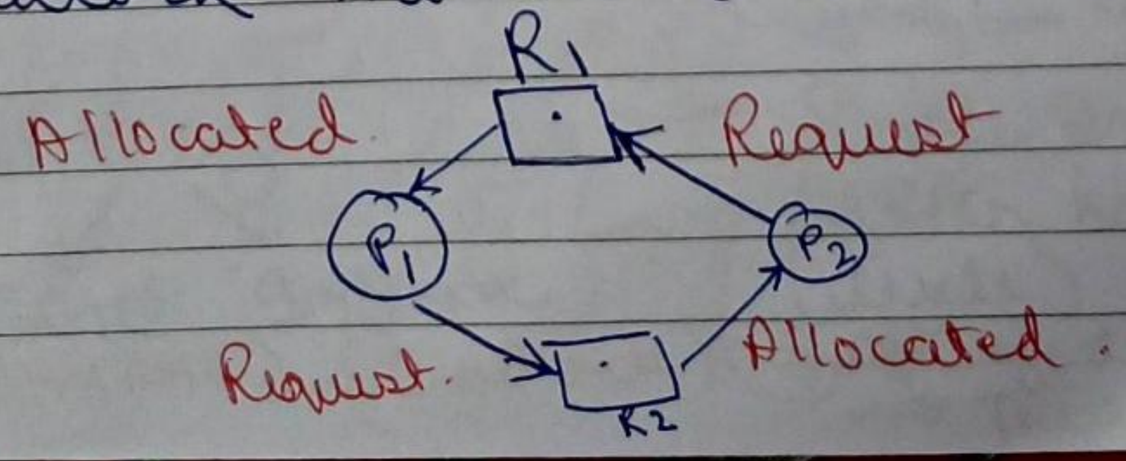



```
void philosopher(void)
{
  while(true){
    Thinking();
    wait(take_fork(i));
    wait(take_fork((i+1)%N));
    EAT();
    signal(Put fork(i));
    signal(Put fork((i+1)%N));
```

'5' Philosophers.
'5' Forks.

Deadlock

If two or more processes are waiting for the happening of some event, which never happens then we say these processes are involved in deadlock then the status called Deadlock



Scanned by TapScanner

Necessary Conditions

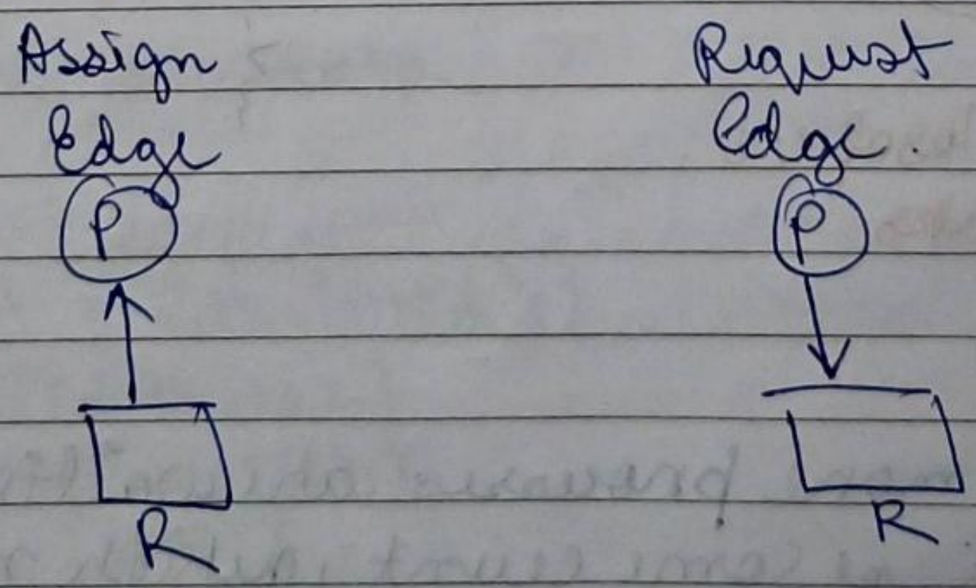
- Mutual Exclusion
- No preemption
- Hold & Wait.
- Circular Wait.

Resource Allocation Graph (RAG).

→ State of the system in terms of processes & resources.

Vertex
- Process Vertex. (Pi)
- Resource Vertex
  - Single Instance — e.g. CPU, Monitor
  - Multi Instance. — eg Register.

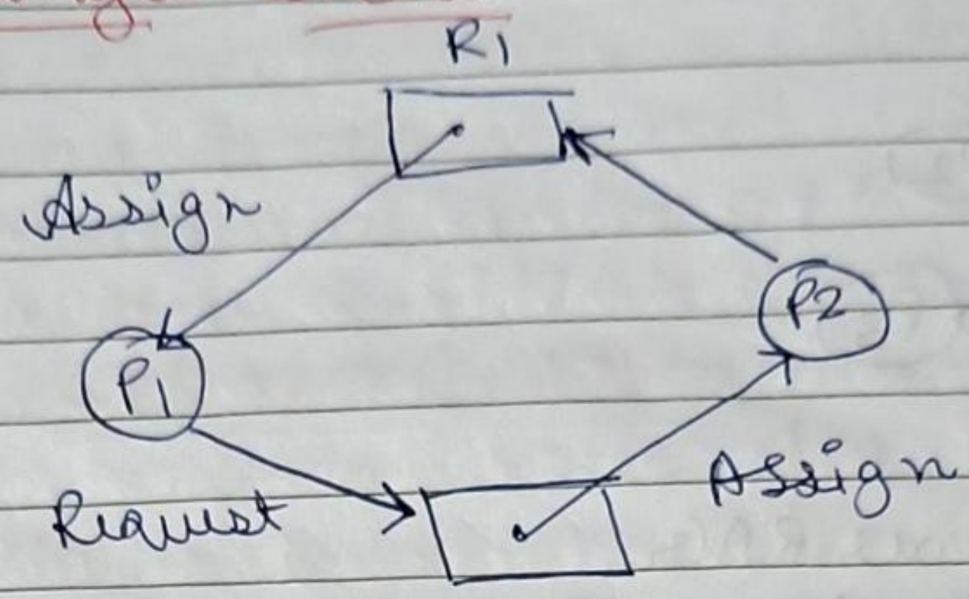
Edge
- Assign Edge
- Request Edge.



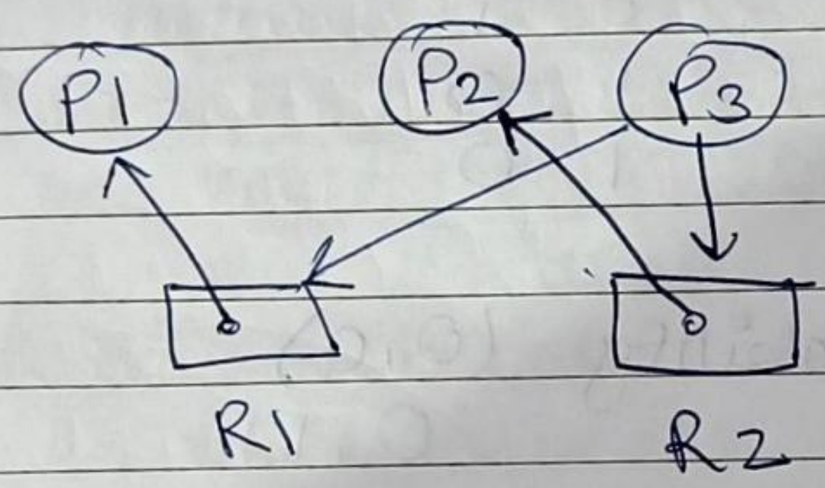
Scanned by TapScanner

6/6

Single Instance



Deadlock



	Allocate		Request	
	R_1	R_2	R_1	R_2
P1	1	0	0	1
P2	0	1	1	0

Availability $\overset{R_1 \; R_2}{(0, 0)}$

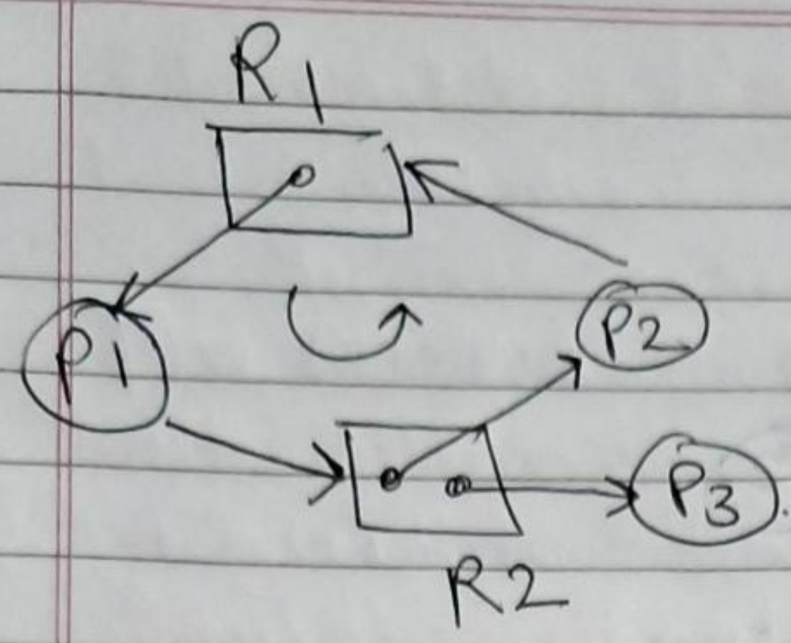
	Allocate		Request	
	R_1	R_2	R_1	R_2
P1	1	0	0	0
P2	0	1	0	0
P3	0	0	1	1

Availability (0,0).
(1,0)
(1,1)

Waiting
- finite (Starvation)
- Infinite (Deadlock)

Single Instance Resource
- if RAG has no cycle then no deadlock.
- if RAG has circular wait (cycle) ⇒ Always deadlock → True

Scanned by TapScanner



Multi Instance RAG.

	Allocate		Request	
	R_1	R_2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

Curr Availability (0,0)
0,1
0,1
0,1
1 0
1 1

Single instance & circular wait
→ deadlock.

Scanned by TapScanner

10/6

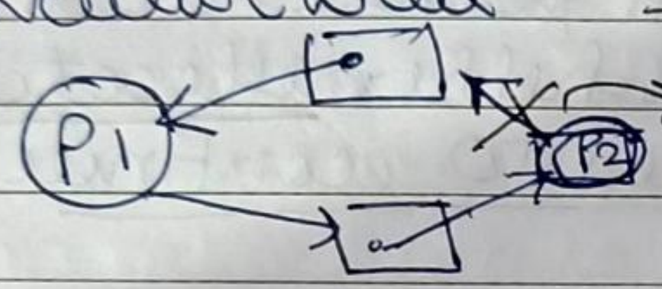
→ Affect Performance (Speed)

Various Methods to handle deadlocks

1) Deadlock ignorance (Ostrich method) – Rare occurance
2) Deadlock prevention
3) Deadlock avoidance (Banker's Algo)
4) Deadlock detection & Recovery
   ↳ Detect ① Kill the processes or process / ② Resource Preemption

Mutual Exclusion
No preemption
Hold & Wait
Circular Wait

→ Remove all or atleast try to remove one.



this request can't be made.

1. Printer
2. Scanner
3. CPU
4. Register

Only req. no. greater than the cur number can be made.

BANKER'S ALGO | Total A=10, B=5, C=7 |

Deadlock Avoidance.

Deadlock Detection.

Process	Allocation (CPU, Memory, Printer) A	B	C	Max Need A	B	C	Available A	B	C	Remaining Need A	B	C	
P1	0	1	0	7	5	3	3	3	2	7	4	3	P1
P2	2	0	0	3	2	2	5	3	2	1	2	2	P2
P3	3	0	2	9	0	2	7	4	3	6	0	0	P3
P4	2	1	1	4	2	2	7	4	5	2	1	1	P4
P5	0	0	2	5	3	3	7	5	5	5	3	1	P5
	7	2	5				10	5	7				

P2 → P4 → P5 → P1 → P3 → Unsafe

Safe Sequence

Current Availability ≥ Remaining Need

Banker's Algorithm in Operating System.

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "S-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Q. A system is having 3 processes each require 2 units of resources 'R'. The minimum no. of units of 'R' such that the deadlock will occur. 4

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

P1	P2	P3
3	4	5
2	3	4

= 9 + 1 = 10

deadlock

without deadlock

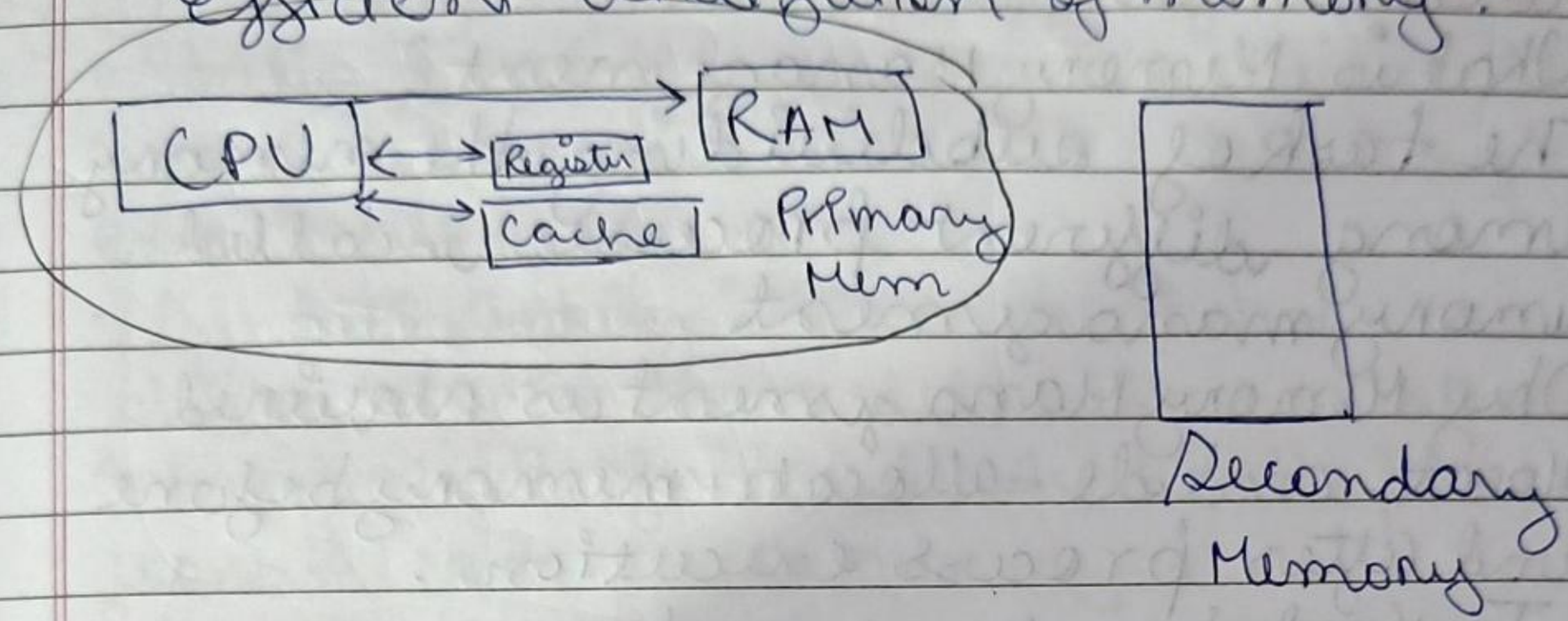
minReq + 1

Memory Management:

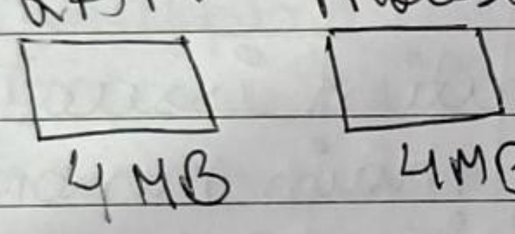
method of managing primary memory.

Goal:

Efficient Utilization of memory.



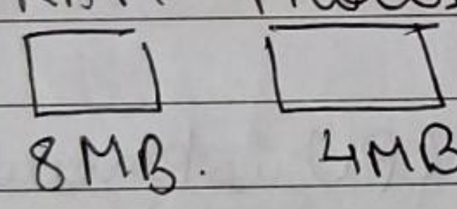
RAM Process.



K = I/O operation (70%).

CPU utilization = (1−K) = 30%.

RAM Process



No. of processes $= \frac{8MB}{4MB} = 2$.

K^2

$K = 70\%$

CPU utilization $= 1-K^2$

CPU $= 1-(0.7)^2$

$= 76\%$

Memory: • Collection of data in a specific format.

- Used to store instructions and processed data.
- The memory comprises of a large array or group of words or bytes, each with its own location.

Scanned by TapScanner

Q. What is Main Memory?
Main memory is a repository of rapidly available information shared by the CPU and I/O devices.
Main Memory → RAM.

Q. What is Memory Management?
The task of subdividing the memory among different processes is called memory management.

Q. Why Memory Management is required.
- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

Logical Address Space :-
- An address generated by the CPU is known as "Logical Address" (virtual address)
- Size of the process.

Physical Address Space :-
- An address seen by the memory unit (i.e. the one loaded into the memory address register of the memory) → "Physical Address".
- The set of all physical addresses corresponding to these logical addresses is known as Physical Address Space.

STATIC AND DYNAMIC LOADING

To load a process into the main memory is done by a loader.
- Static loading - loading the entire program into a fixed address. Requires more memory space
- Dynamic loading - A routine is not loaded until it is called.
  ⇒ large amount of code.

Scanned by TapScanner

Static And Dynamic Linking

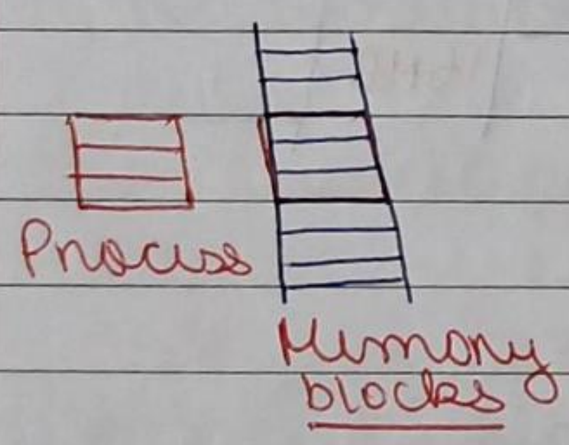
To perform a linking task a linker is used. A linker is a program that takes one or more Object files generated by a compiler and combines them into a single executable file.

- Static linking: Linker combines all necessary program modules into a single executable program.
- Dynamic linking: When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

Swapping

Swapping is a process of swap a process temporarily into a secondary memory from the main memory, which is fast as compared to secondary memory.

Contiguous Memory Allocation :-



Contiguous Memory Allocation.

- We need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory.
- In adjacent memory allotment, each process is contained in a single contiguous segment of memory.

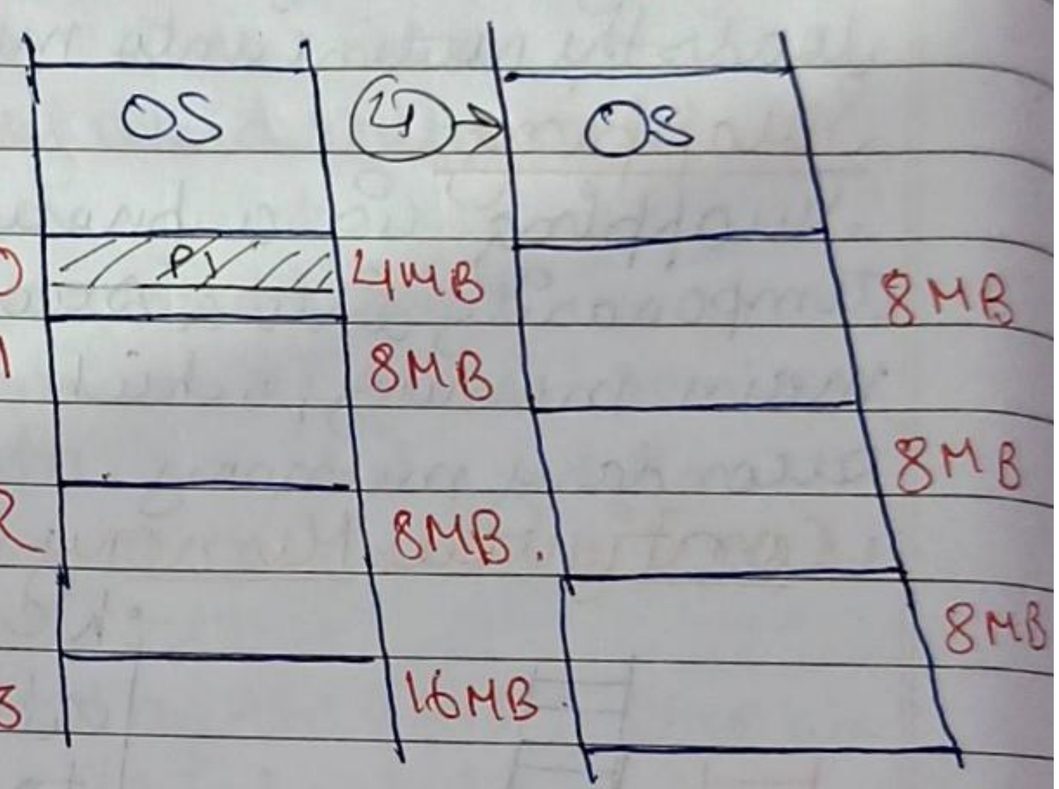
~~Memory Allocation~~

Memory Management Techniques

- Contiguous
  - Static
    - Fixed Partition
  - Dynamic
    - Variable Partition
- Non contiguous
  - Paging
  - Multilevel paging
  - Inverted paging
  - Segmentation
  - Segmented paging

Fixed Partitioning (Static Partition)

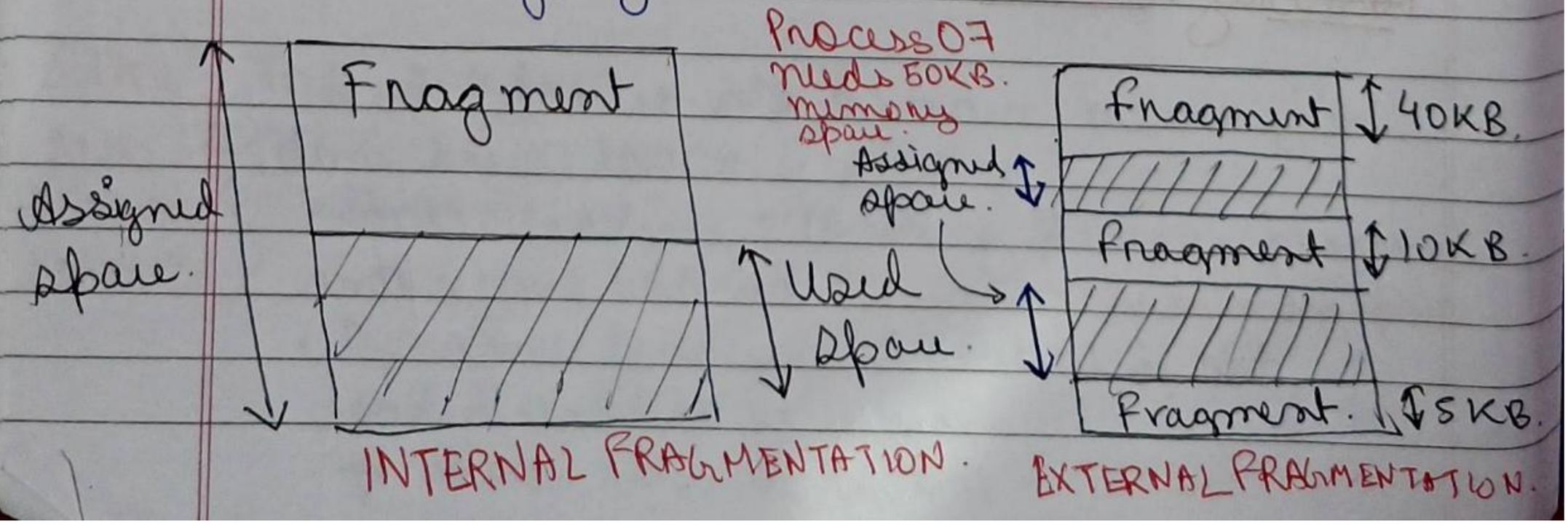
- "No. of partitions are fixed"
- Size of each partition may or may not be same. (4) →
- Contiguous allocation so spanning is not allowed.



Two types of Fragmentation

1. Internal Fragmentation :-

In the case where the memory allotted to the method is somewhat larger than the memory requested, then the difference between allotted & requested memory is called internal fragmentation.



Scanned by TapScanner

→ Compaction.

2. External fragmentation.

- Happens when there's sufficient quantity of area within the memory to satisfy the memory request of a method.
- memory offered → non-contiguous manner.
- When we apply a first fit or best-fit memory allocation strategy it'll cause external fragmentation.

→ Disadvantages
- Internal fragmentation.
- Limit in process size
- Limitation on Degree of Multiplexing
- External fragmentation

Dynamic Partitioning / Variable partitioning.

(hole)

- No internal fragmentation.
- No limitation on no. of processes
- No limitation on process size.

Disadvantages :-
- Allocation / Deallocation
- External Fragmentation.

☆ First-fit : Allocate the first hole that is big enough. (Simple/fast)
(pointer).

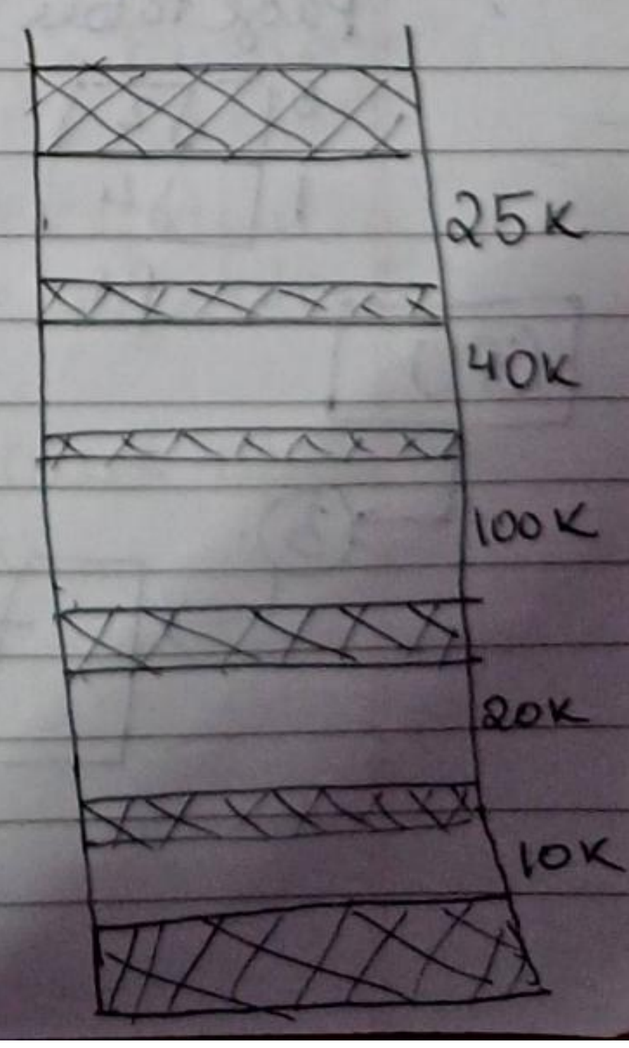
Next-fit : Same as first fit but start search always from the last allocated hole.

Best-fit : Allocate the smallest hole that is big enough.

Worst-fit : Allocate the largest hole.

(slow)

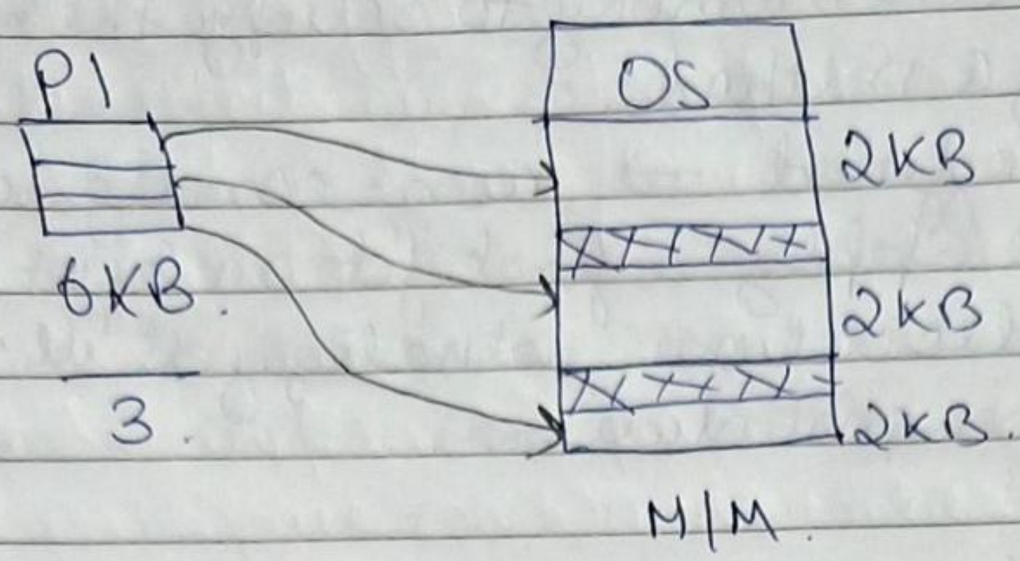
↳ Search in the whole list.



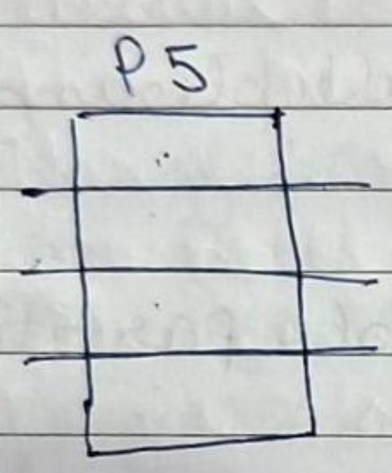
Scanned by TapScanner

11/6

Non-Contiguous Mem Alloc.



Page size = Frame size.

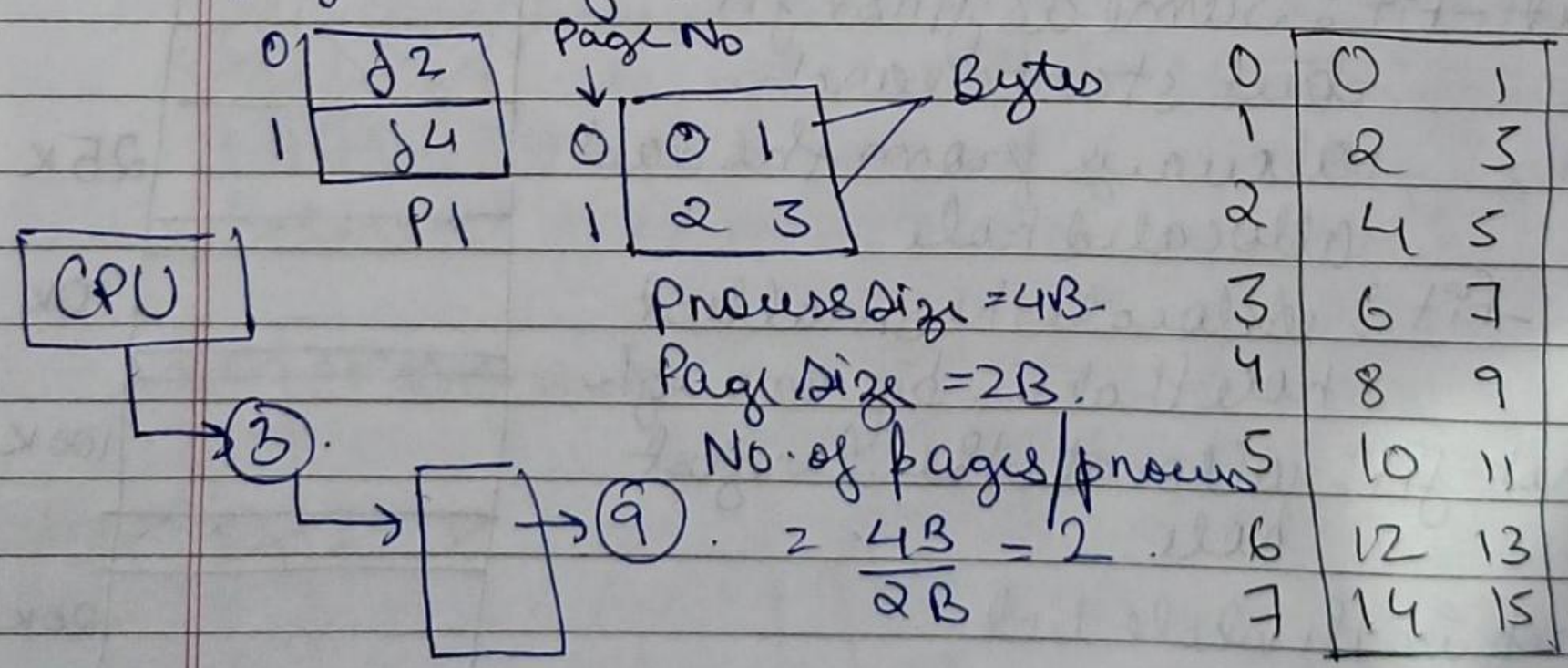


P1	0
P1	1
P5	2
P5	3
P3	4
P3	5
P5	6
P5	7

Page size = Frame size
= 1KB.

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
This scheme permits the physical address space of a process to be non-contiguous.

Page table of P_1



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

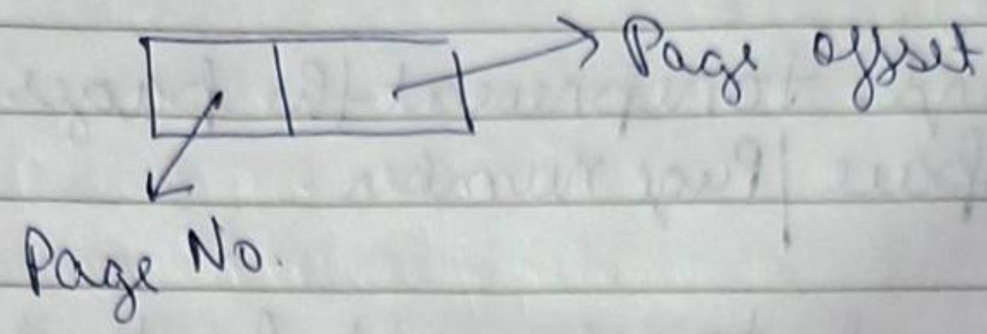
M/M

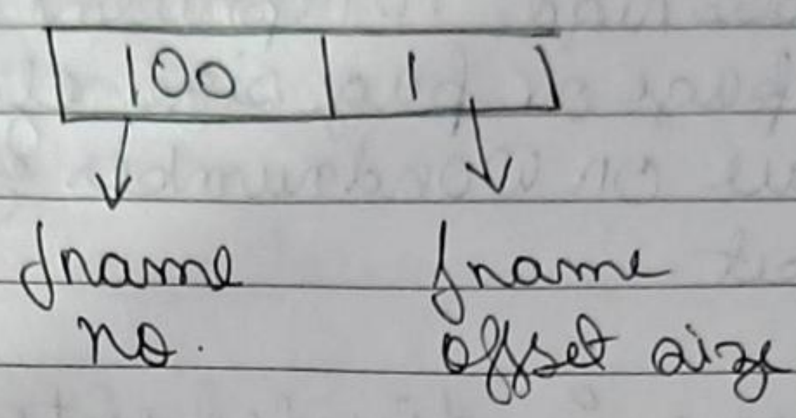
Process Size = 4B.
Page Size = 2B.
No. of pages/process $= \frac{4B}{2B} = 2$.

M/M size = 16B.
Frame size = 2B.
No of frames $= \frac{16B}{2B} = 8$ frames.

Scanned by TapScanner

Logical address





- Logical Address / Virtual Address :-
An address generated by the CPU.
- Logical address Space / Virtual Address Space :-
The set of all logical addresses generated by a program.
- Physical address :
An address actually available on memory unit.
- Physical address Space :
The set of all physical addresses corresponding to the logical address.

- The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device & this mapping is known as paging technique.
- The Physical Address Space is conceptually divided into a number of fixed - size blocks called frames.
- The logical address space is also splitted into fixed size blocks called pages.
- Page Size = Frame size.

Scanned by TapScanner

Address generated by CPU is divided into

- Page number (p):
  Number of bits reqd. to represent the pages in logical Address Space / Page number.
- Page offset (d):
  Number of bits reqd. to represent particular word in a page or page size of logical Address Space or word number of a page or page offset.

Physical Address is divided into:-

- Frame number (f):
  Number of bits required to represent the frame of Physical Address Space or Frame Number.
- Frame offset (d):
  Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

<u>Pagetable entry</u>

FRAME NO	Valid (1) / Invalid (0)	Protection (RWX)	Reference (0 / 1)	Caching	Dirty

← Mandatory Field ← — Optional fields — →

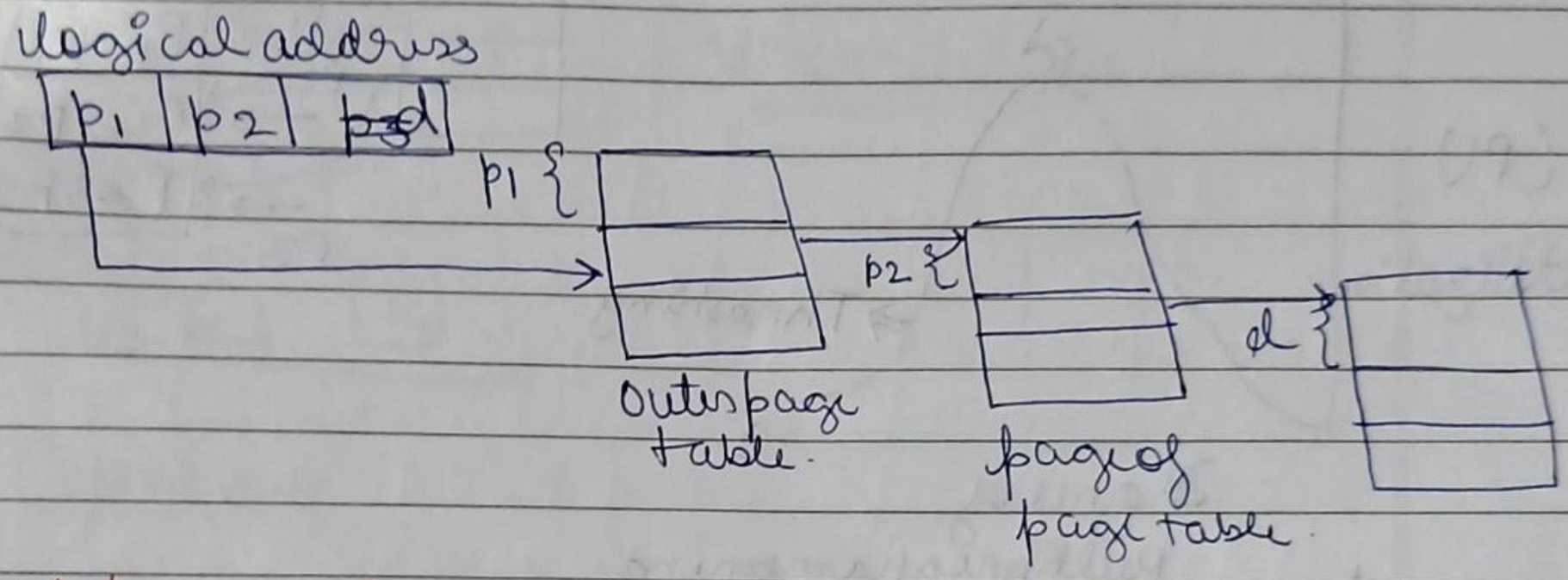
Scanned by TapScanner

Multilevel Paging

- Consist of two or more levels of page tables in a hierarchical manner.

In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory.

Address Translation Scheme

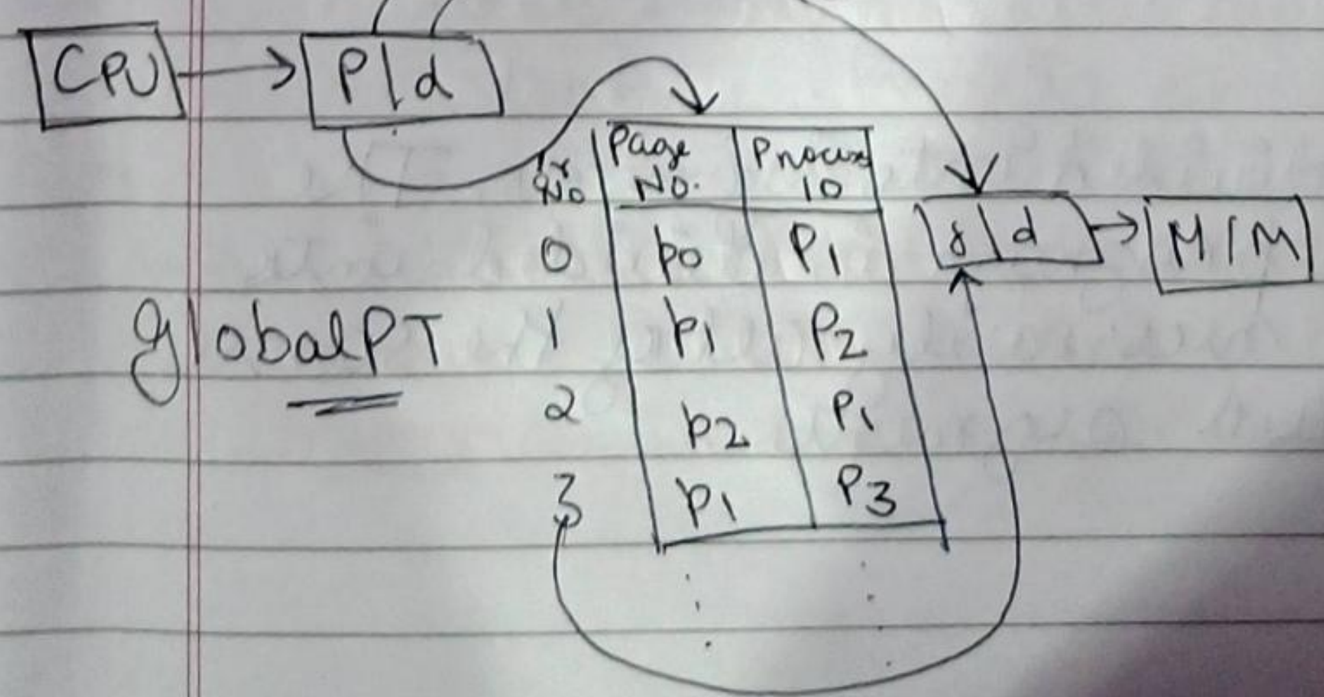


INVERTED PAGE TABLE IN OS.

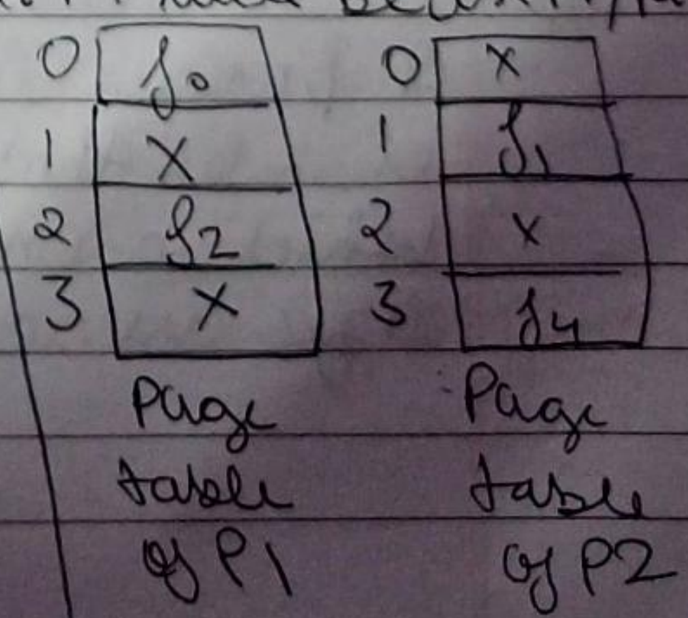
Inverted paging as the indexing is done wrt the frame number instead of the logical page number. Each entry in the page table contains the following fields

- Page No.
- Process id
- Control bits
- Chained pointer.

Inverted paging



1. Each process has its own PT.
2. PT will be in M/M



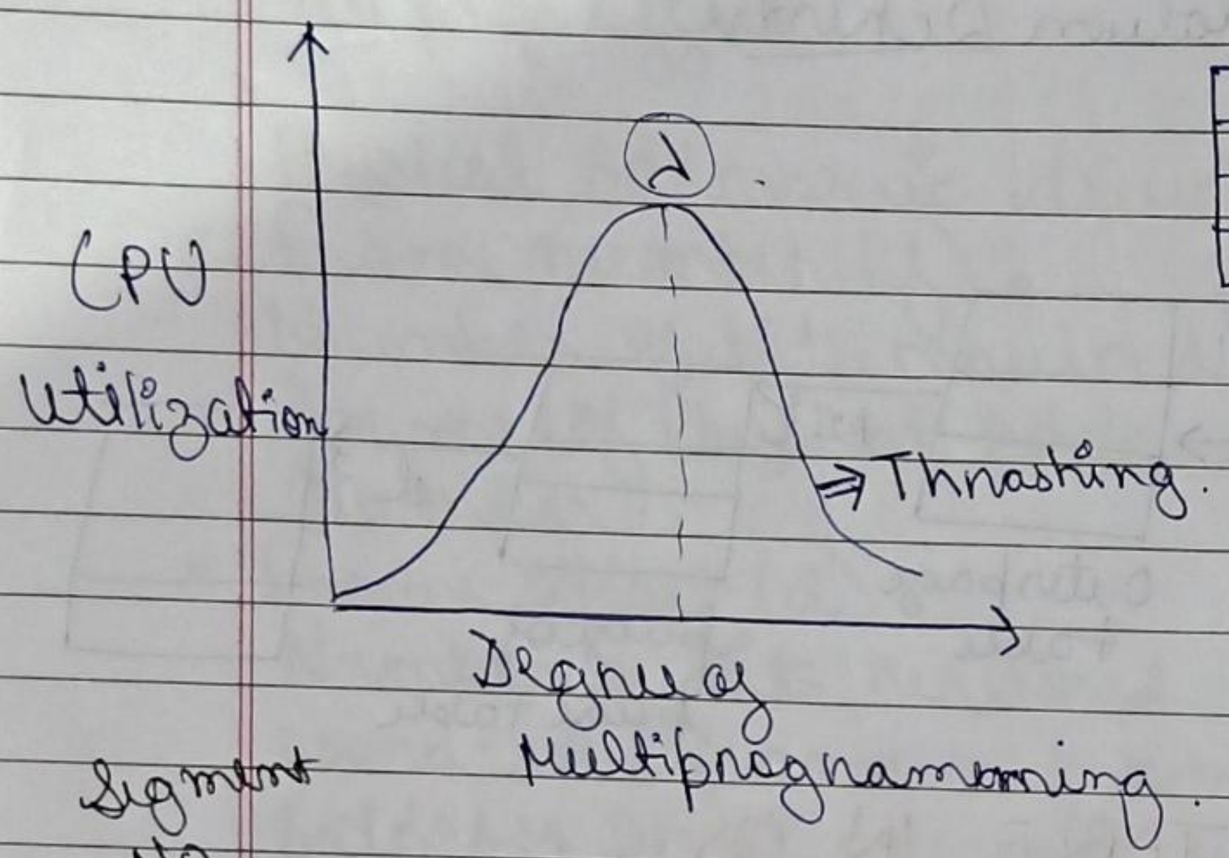
Scanned by TapScanner

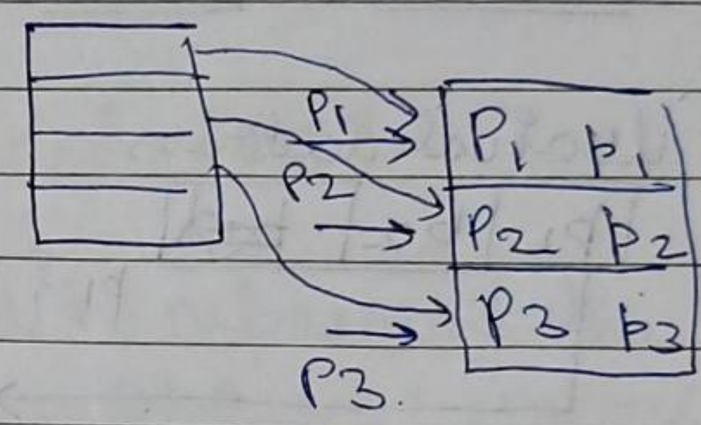
12/06

Thrashing

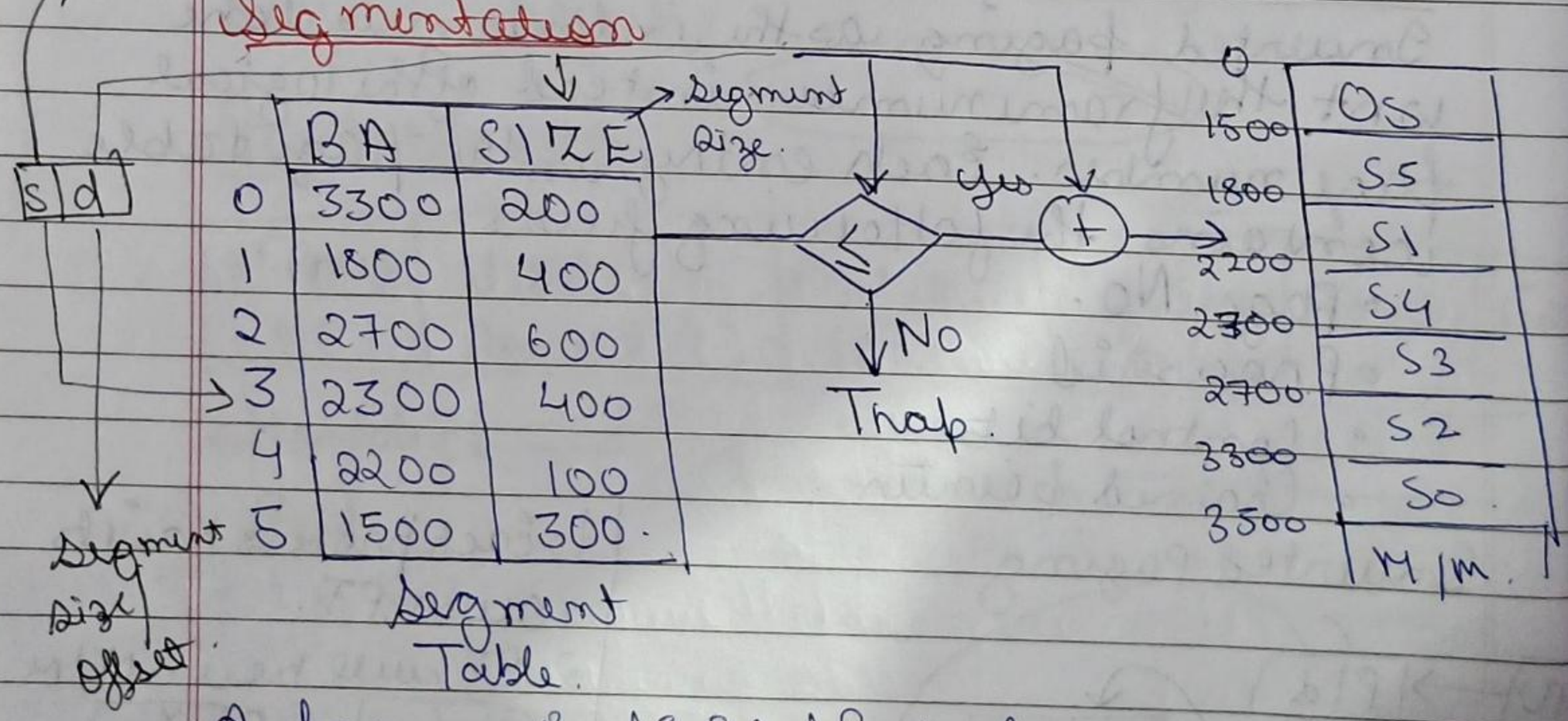
- M/M size increased
- Long term schedular.

Thrashing is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.





Segmentation



	BA	SIZE
0	3300	200
1	1800	400
2	2700	600
3	2300	400
4	2200	100
5	1500	300.

Segment Table.

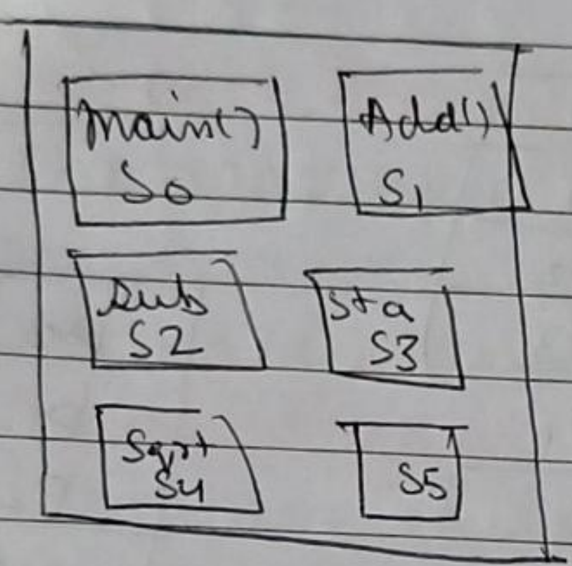
A process is divided into segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments.

Scanned by TapScanner

Types of segmentation :-

1. Virtual Memory segmentation :-
Each process is divided into a number of segments, not all of which are resident at any one point in time.

2. Simple Segmentation :-
Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously

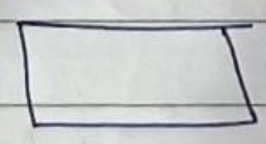


VARIOUS SIZE

Overlay

Pass 1	Pass 2
80KB.	90KB.
30KB.	30KB.
20KB	20KB
10KB	10KB
140KB.	150KB.

80KB.
90KB.
30KB.
20KB.
10KB
X 230KB.



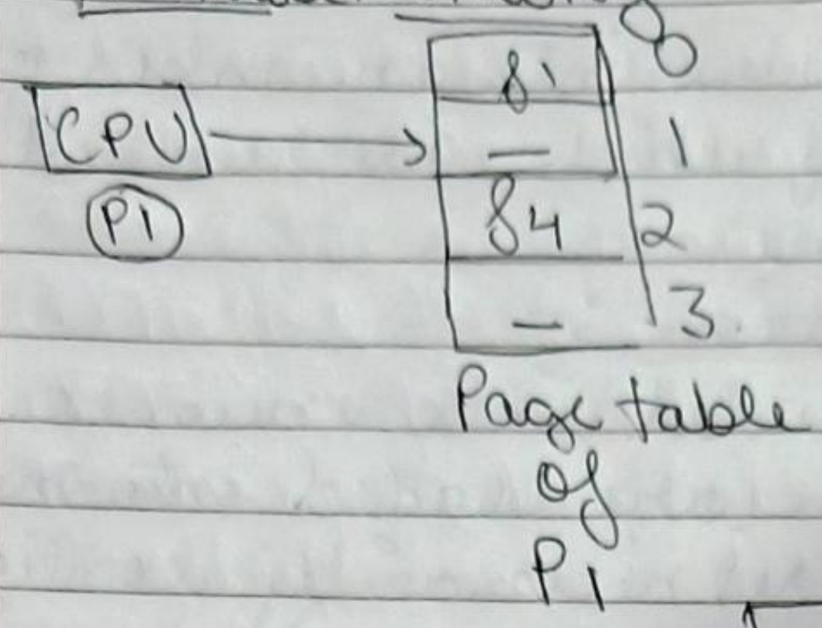
150KB.

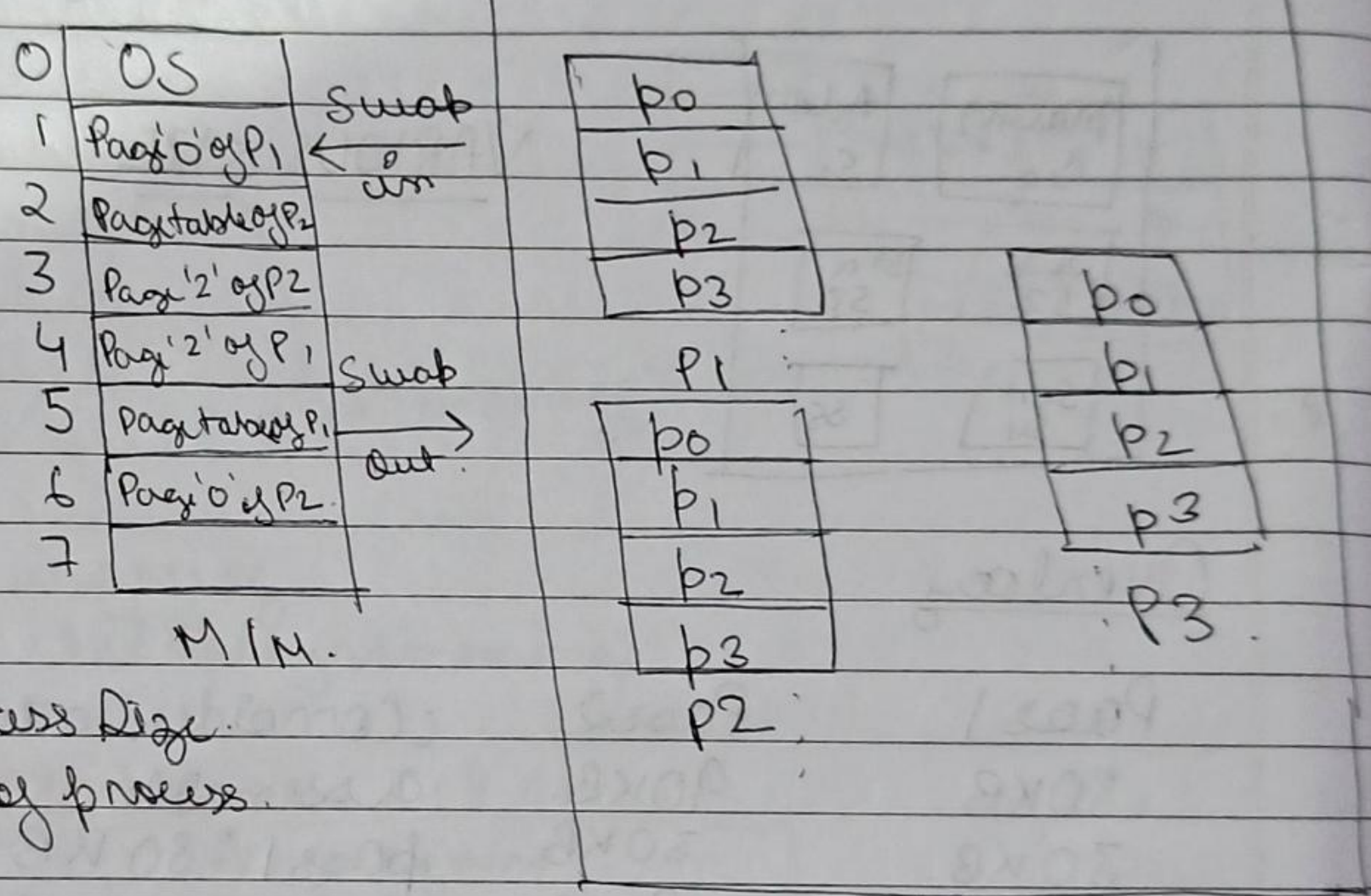
Consider a two pass assembler
pass 1 : 80 KB.
pass 2 : 90 KB.
Symbol table : 30KB.
Common routine : 20KB.
At a time only one pass is in use.
What is min partition size reqd. if ~~only~~ overlay driver is 10KB size?

→ The process of transferring a block of program code or data into internal memory, replacing what is already stored.

Scanned by TapScanner

Virtual Memory





- Process Size.
- No. of process.

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

Scanned by TapScanner

- All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time.
- A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

Demand Paging:

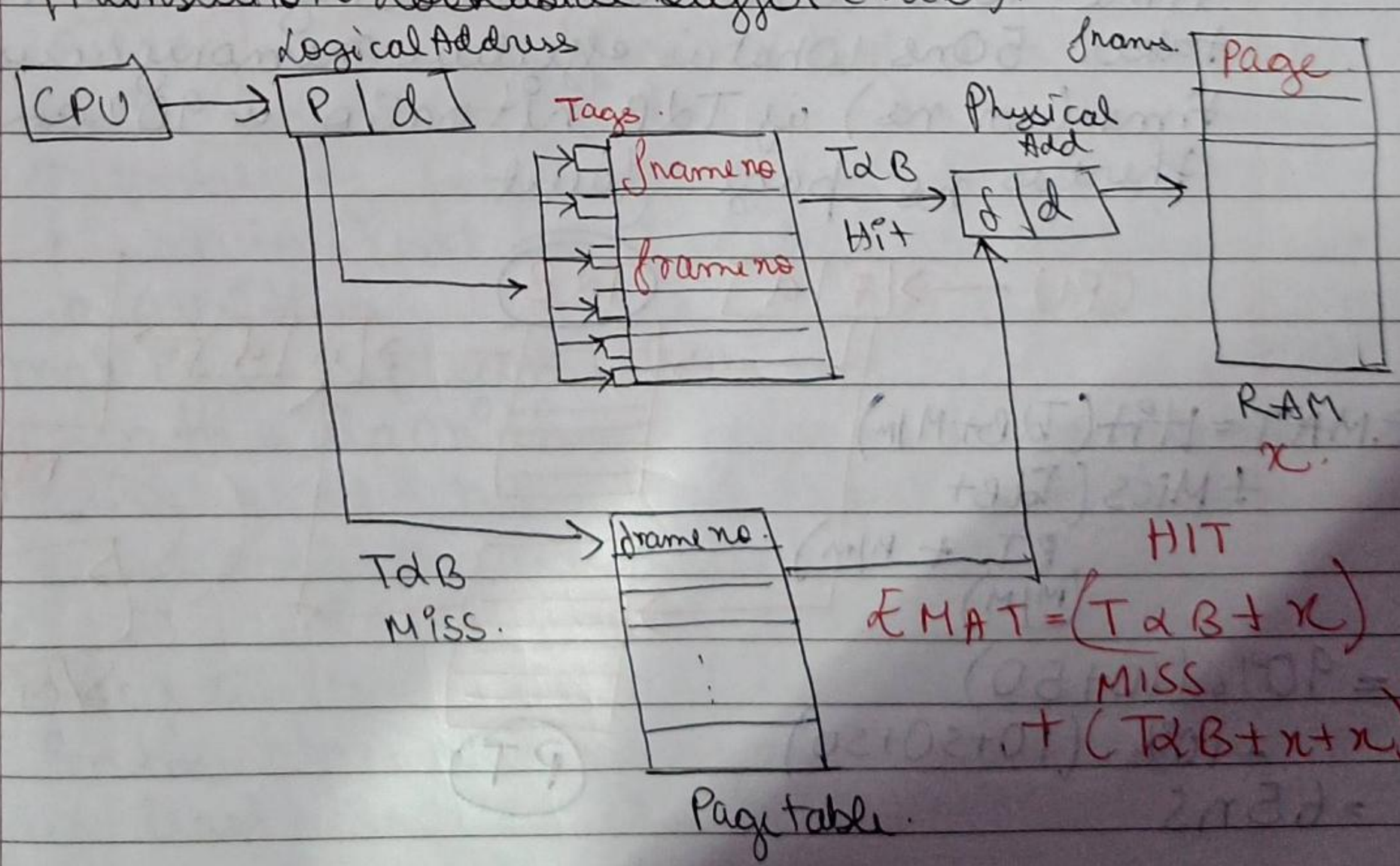
The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

Swapping:

Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the normal page replacement process.

When a process is busy swapping pages in and out then this situation is called throshing.

Translation Lookaside Buffer (TLB).



Scanned by TapScanner

To overcome the size issue, the entire page table was kept in main memory, but the problem here is two main memory references are required:-

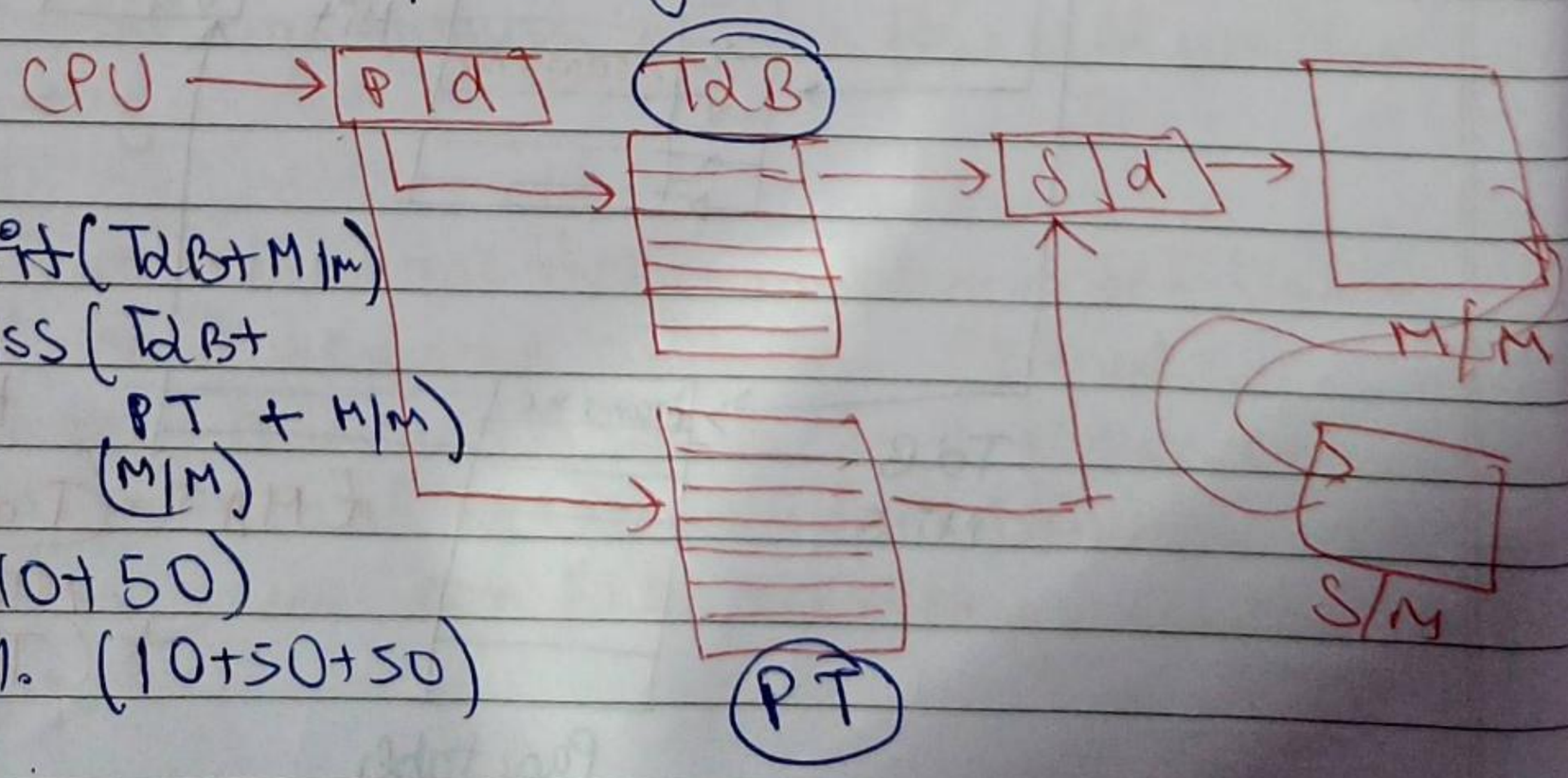
(i) To find the frame number.

(ii) To go to the address specified by frame no.

- To overcome this problem a high speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB)
- Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions.
- TLB contains page table entries that have been most recently used.
- TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

Translation Lookaside Buffer (TLB).

A paging scheme using TLB. TLB Access Time 10ns and main memory access time takes 50ns. What is effective memory access time (in ns) if TLB hit ratio is 90% and there is no page fault.



EMAT = ~~Hit~~ (TLB + M/M)
+ Miss (TLB + PT + M/M)
(M/M)

= 90% (10+50)
+ 10% (10+50+50)
= 65 nS.

Scanned by TapScanner

15/06

Page Replacement Algo.
1) FIFO.
2) Optimal Page Replacement.
3) Least Recently Used (LRU)

FIFO:-

f_3			1	1	1	1	0	0	0	3	3	3	3	2	2
f_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
					Hit							Hit			Hit

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

Page Hit = 3
Page Fault / Page Miss = 12

Hit Ratio = $\frac{\text{No of hits}}{\text{No of References}}$

In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Belady's anomaly:-

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First out (FIFO) page replacement algorithm.

Optimal Page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Least Recently Used

In this algorithm, page will be replaced which is least recently used.

Scanned by TapScanner

classmate

Date
Page

16/06 Replace the page which is not used in longest Dimension of time in future.

Optimal Page Replacement

f4			2	2		
f3		1	1	1		
f2	0	0	0	0		
f1	7	7	7	7	7	3
	LRU				Hit	*

Replace it with 3.

Ref String :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Most recently used.

Optimal Page Replacement

at last.

Least Recently Used

(Replace the least recently used page in past)

f4			2	2	2	
f3		1	1	1	1	
f2	0	0	0	0	0	
f1	7	7	7	7	7	3
	*	*	*	*	Hit	*

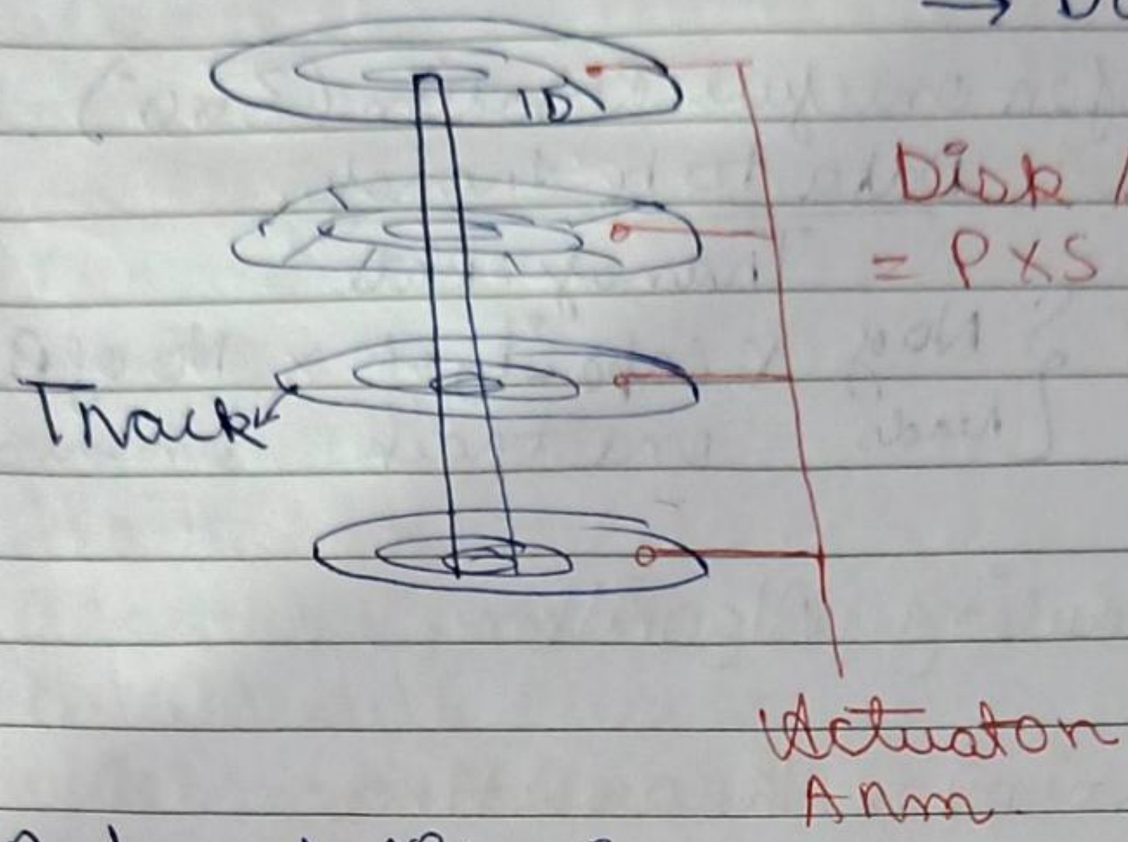
Most Recently Used

f4				2	2	2
f3			1	1	1	1
f2		0	0	0	0	3
f1	7	7	7	7	7	7
	*	*	*	*	Hit	*

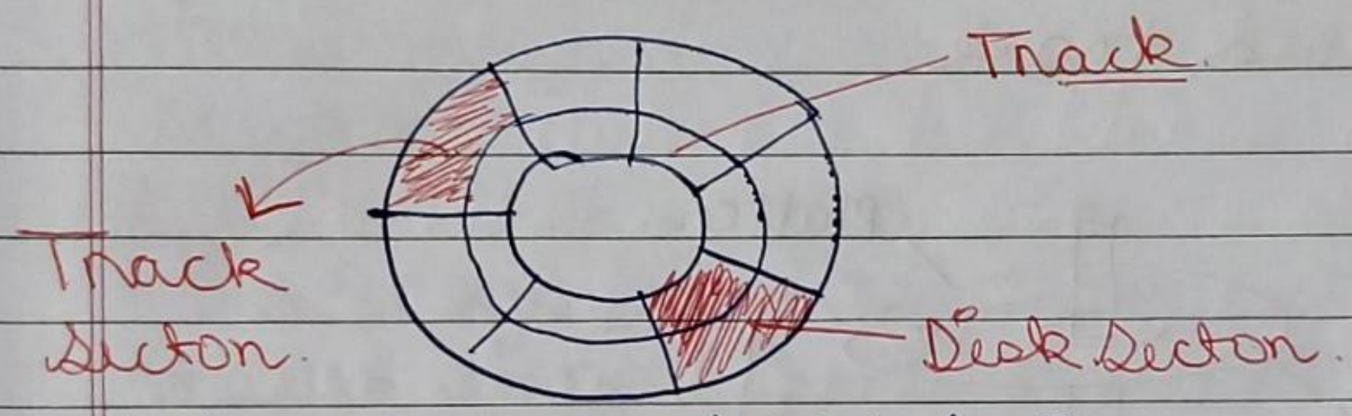
Scanned by TapScanner

Disk Architecture

platters → surface → track → sectors
→ Data



- A hard disk is a memory storage device that looks like this:



The disk is divided into tracks. Each track is further divided into sectors.

Read-Write (R-W) head moves over the rotating hard disk.

Disk Access Time - To perform a read or write operation on a memory location, we need to place the R-W head over that position.

1. **Seek time**: The time taken by the R-W head to reach the desired track from its current position.
2. **Rotational Latency**: Time is taken by the sector to come under the R-W head.
3. **Data transfer time**: Time is taken ~~by the~~ to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller Time**: The processing time taken by the controller.

Scanned by TapScanner

5. Average Access Time:
Seek time + Average Rotational Latency + data transfer time + controller time.

6. Rotation Time:
Time taken for one full Rotation (360°).

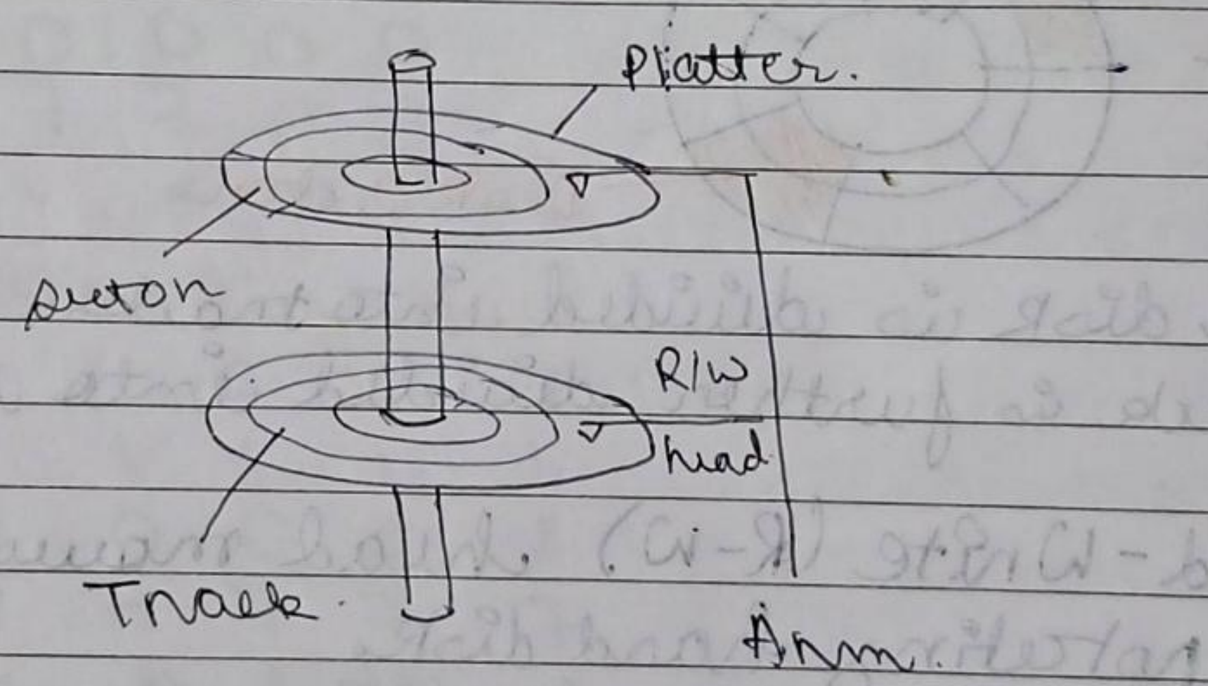
7. Transfer Time: $\frac{\text{Data to be transfer}}{\text{Transfer Rate}}$

8. Transfer Rate: (Data Rate) { No of heads × capacity of one track × No of Rotations in one sec }

18/06 Disk Scheduling Algorithm.

Goal: To minimize the seek time.

Seek Time: Time taken to reach upto desired track.



- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN
- LOOK
- CSCAN (Circular Scan)
- CLOOK (Circular LOOK).

Scanned by TapScanner

classmate
Date
Page

FCFS: FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

SSTF: In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.

- As a result, the request near the disk arm will get executed first.
- SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

SCAN:

- In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.
- So, this algorithm works as an elevator and hence also known as elevator algorithm.
- As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

CSCAN:

- In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction.

Scanned by TapScanner

So, it may be possible that two many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

- These situations are avoided in CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there.
- So, the disk arm moves in a circular fashion and this algorithm is similar to SCAN algorithm and hence it is known as C-SCAN (circular SCAN).

LOOK:

- Similar to SCAN disk scheduling algorithm.
- Disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.
- Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

CLOOK:

In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request.

Scanned by TapScanner

- RSS:
  - It stands for random scheduling and just like its name it is nature.
  - It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm gets perfect.
  - Which is why it is usually used for and analyses and simulation.
- LIFO:
  - newest jobs are serviced before the existing ones.
- N-STEP SCAN:
  - In this a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go.
  - Also once the buffer is full no new requests are kept in this buffer and are sent to another one.
  - Now, when these N requests are serviced, the time comes for another top N requests and this way all get requests get a guaranteed service.
- FSCAN: This algorithm uses two sub-queues. During the scan all requests in the first queue are serviced and the new incoming requests are added to the second queue.
  All new requests are kept on hold until the existing requests in the first queue are serviced.

Scanned by TapScanner

classmate
Date
Page

19/06

File System in OS.
↓
Software
How data will be stored and fetched

User → file → folder/ → file system
Directory

- A file is a collection of related information that is recorded on secondary storage.
- On file is a collection of logically related entities.
- From user's perspective a file is the smallest allotment of logical secondary storage.

The name of the file is divided into two parts as shown below:-
- name
- extension, separated by a period.

Operations on files.
1. Creating
2. Reading
3. Writing.
4. Deleting.
5. Truncating
6. Repositioning.

File Attributes.
1. Name.
2. Extension (Type)
3. Identifier.
4. Location.
5. Size.
6. Modified date, Created date.
7. Protection/ Permission.
8. Encryption, compression.

Scanned by TapScanner

File Directories :-

- Collection of files is a file directory.
- The directory contains information about the files, including attributes, location and ownership.
- Much of this information, especially that is concerned with storage is managed by the operating system.
- The directory is itself a file, accessible by various file management routines.

Advantages: Efficiency, Naming, Grouping.

- Single level Directory
- Two level Directory.
- Tree structured Directory.

Allocation Methods

- Contiguous Allocation
- Non Contiguous Allocation
  - → Linked List Allocation
  - → Indexed Allocation.

1. Continuous Allocation :-

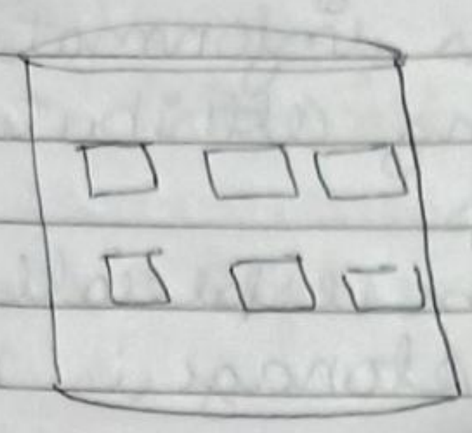
- A single continuous set of blocks is allocated to a file at the time of file creation.

- Advantages
  1) Easy to Implement.
  2) Excellent Read Performance.
- Disadvantages.
  1) Disk will become fragmented.
  2) Difficult to grow file.

- Here, the file allocation table needs just a single entry for each file, showing the starting block and the length of the file.
- Pre allocation strategy, using variable size portions.

Scanned by TapScanner

Linked List Allocation.



Data
Pointer

Directory.

file	start
A	2.

Pointer.

Advantages:
- No External fragmentation.
- File size can increase.

Disadvantages:
- Large seek time.
- Random Access / Direct Access Difficult
- Overhead Pointers.

↳ Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain.
- Again the file table needs just a single entry for each file, showing the starting block and the length of the file.

Indexed Allocation.

The file allocation table contains a separate one-level index for each file: This index has one entry for each block allocated to the file.

Advantages
- Support direct access
- No External fragmentation.

Disadvantages
- Pointer Overhead
- Multilevel Index.

Directory

File	Index Block
A	6

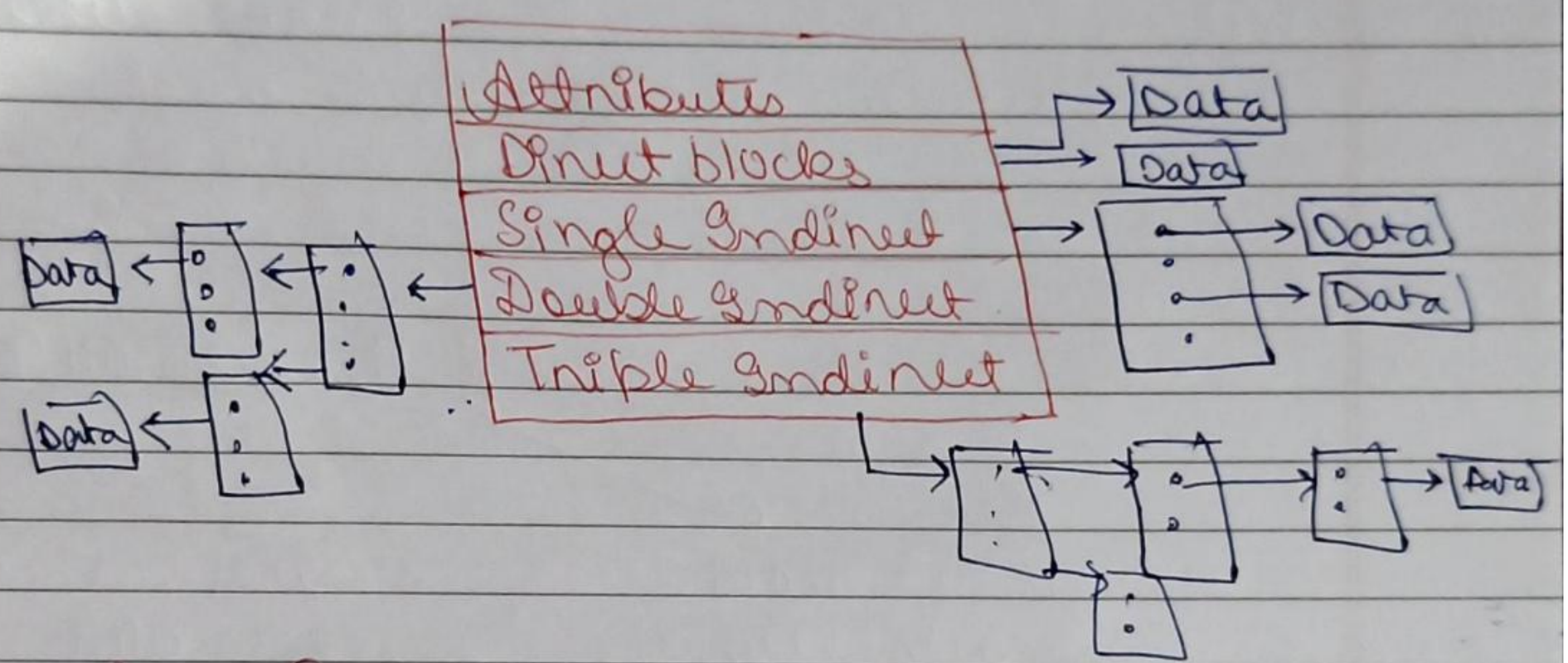
Scanned by TapScanner

22/06

Inode in Operating System

- In Unix based operating system each file is indexed by an Inode. Inode are special disk blocks they are created when the file system is created.
- The number of Inode limits the total number of files / directories that can be stored in the file system.

Unix Inode Structure



	Security	Protection
Basic	Security grants the system access to appropriate users only.	While protection deals with the access to the system resources.
Type of threats involved of the system	In security, external threats are involved	While in protection, internal threats are involved.
Queries handle	In security, more complicated queries are handled.	Whereas in protection, simple queries are handled.
Policy	Security illustrates that which person is granted for using the system.	Whereas protection determines that what files can be accessed or permitted by a special user.

Scanned by TapScanner

5. Mechanism

In security, encryption and certification (authentication) mechanisms are used.	Whereas in protection, authorization mechanism is implemented.

Scanned by TapScanner