Operating System

Date / /
Page

I. Introduction and Background :

II. Process Management :
- → Process Concept
- → CPU scheduling
- → Process Synchronisation
- → Concurrent programming
- → Deadlocks and Threads

III. Memory Management
- → RAM chip - Implementation.
- → Loading - Linking and Address Binding
- → Techniques
- → Paging
- → Multilevel paging
- → Inverted page Table
- → Segmentation
- → Segmented Paging.
- → Virtual Memory

IV. File and Device Management.

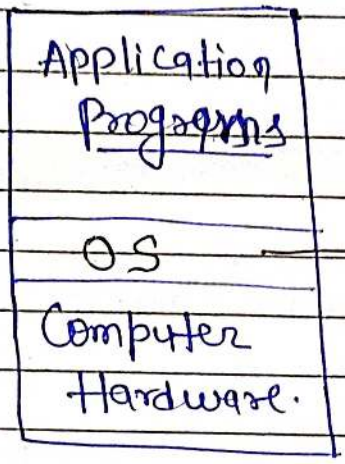
V. Protection and Security.

Processor unit ⇒ cycles/sec = Hertz

Date / /
Page

RANKA

I. Introduction and Background –



→ Operating System is also known as System Software.

→ It provides an Environment.

→ Processor executes the instructions of a Programs in a sequential manner.

2.2 GHz

2.2×10^9 Hz = 2.2×10^9 cycles/sec.

2.2×10^9 cycles ⟶ 1 sec.

1 cycles ⟶ $\frac{1}{2.2 \times 10^9}$ sec.

≅ ns

1 cyc ⇒ 1 micro instructions Execution.

Phases of Execution of instructions -

IF. – fetch

ID – Decode

OF – operand fetch

Execut. –

WB. –

MS – Secondary memory
MS – Main memory
RS – Registers

Date / /
Page

→ Program in Execution is known as a process.

→ Program → passive Entity → S. Memory
Process → Active Entity → M. Memory

- Types of operating System-

① Batch operating System
② Multiprogramming OS
③ Multitasking OS
④ Multiprocessor OS
⑤ Real time OS.

① Batch Operating System →

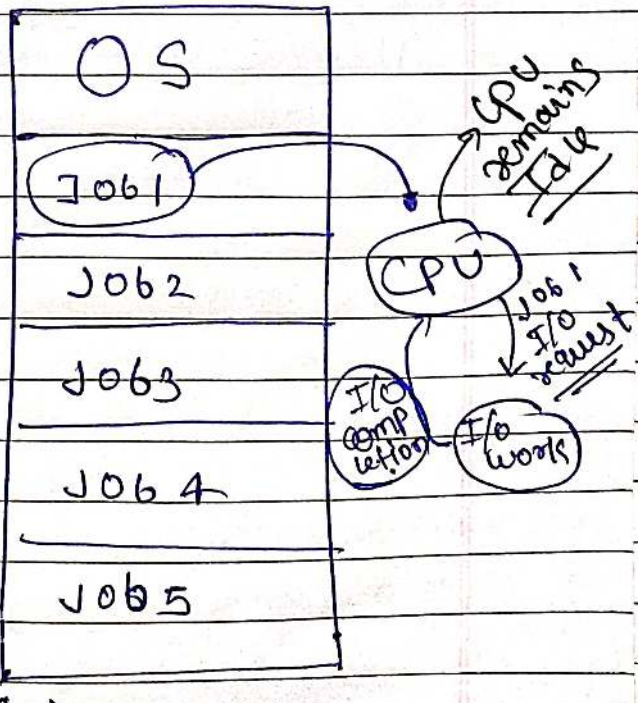
→ Efficiency = Total no. of jobs done / Total time taken.

→ Processor System → Unique Processor
→ Multiprocessor

ROM → Bootstrap Loader (NIC)

→ Similar types of Batch are punched at a time.

CPU + I/O work work



Main memory

Date / /
Page

RANKA

- System call -

→ It is a request made by the user program to get the service of an operating system.

```
int main()
{ int a, b, c;
  scanf("%d %d", &a, &b);
  c = a+b;
  printf("%d", c);
  return 0;
}
```

→ read()

→ write()

System call

→ Disadvantage

→ CPU remains Idle at the time of I/O work.

→ ~~only job of the~~

→ CPU performs only one job at a time then ~~it~~ only it can move to next job

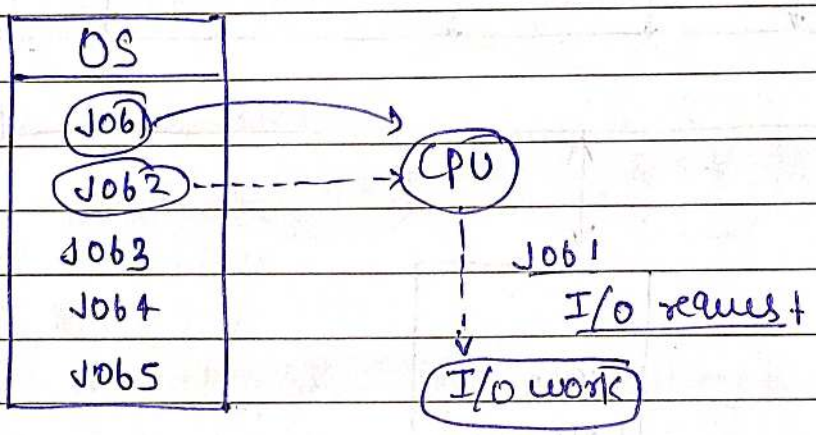
→ .

✡ Multiprogramming OS → (Unique Processor)

- **Degree of Multiprogramming →**

→ The number of programs residing in the main memory at any point of time is known as Degree of Multiprogramming.

→



→ Atleast 1 program is present in Main memory, the CPU can never be Idle.

→ In the

✡ Multitasking OS → (Unique processor)

→ It is an Extension of Multiprogramming OS.

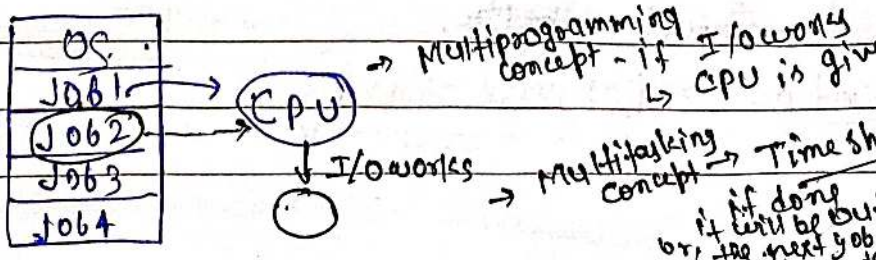
→ (Extend) keyword → Inheritance
↳ parent class property → Child class

→ Code reusablity - Everything of Multiprogramming is inherited to Multitasking.

→ CPU is shared b/w the programs in time Sharing manner.
(Time Quantum) (TQ) = 2ms. - e.g.

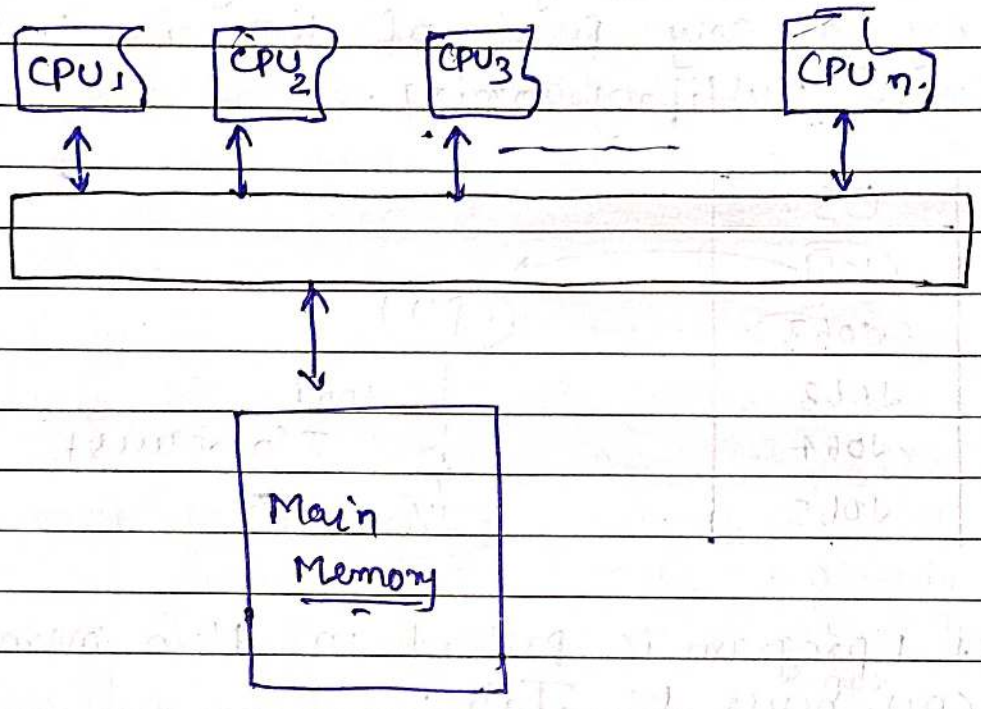
→ Switching is fast => So seems to be Multi program at one time.

→



→ Maximum duration of time a process can hold the CPU in Multitasking OS is Time Quantum.

✡ Multiprocessor OS -



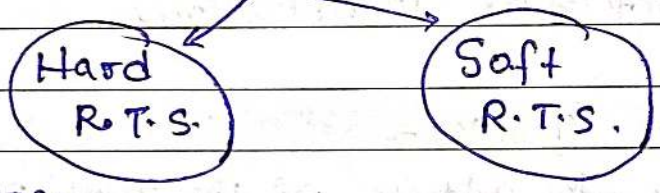
→ Multiple processor.
→ All other resources will be same
→ Resources will be shared with all these CPUs.
→ No. of jobs → distributed → Efficiency ↑
→ Advantage - Efficiency increased
→ fault tolerant — if one CPU gets fault, work trans. to other.
→ Disadvantage
↳ competition & contention in Accessing the Resources from Main memory.
→ Parallelism → Can be achieved.
↳ parallel work →
↳ Multiple process → Execution.
→ ~~Eco~~ Economical → ~~Costly~~.
→ Contention → Resource Sharing

→ 10 processor → Same time of work.
in 1 system
→ 10 System → Same time of work.

✡ Real time OS →

→ Realtime application → Strictly Time bounded System

Real time System



→ Missile Launching.

Hard R.T.S

↳ if 100 attempts, then 100 attempts must be Succeeded.

→ No, unsuccessfull is allowed.

Soft R.T.S

↳ 100 attempts, → 90 - Successfull. } If it's fine.
10 - Unsucessfull. }

→ Unsucessfull is allowed.

→ online Transaction OTP..

• Process Concepts →

Programs → Process

Sec. Mem. Primary Memory.

→ Processes → User Process
↳ ~~Sys~~

→ Process → ID assigned → PID → Process Identification number.
↳ number, with the help of this, OS identifies the Process. → Uniquely identify

Register → GPR - General purpose Register - $R_0, R_1 \dots R_n$
→ SR. - special Register
→ (PC), AC, MAR
IR, DR.

Date / /
Page
RANKA

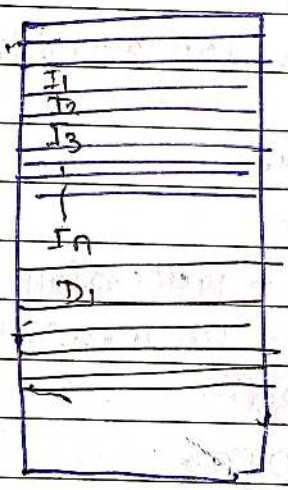
- **Attributes Of a Process -**

(Context)

1. Process ID
2. Process State
3. Process Counter
4. Priority
5. General Purpose Register.
6. List of open files.
7. List of open Devices
8. Protection information.

- **Program Counter -**

→ It holds the address of the next ~~Programs executed~~ instructions.



→ Always holds the address of next instruction that will be executed by processor.

- **Accumulator -** (Add - Binary)

↳ One of the operands present in Accumulare
↳ one in General purpose Register.
↳ Stores Results.

- **MAR :** (Memory Address Register)

→ holds the ~~current~~ Address of the current instruction that is being executed by the processor.

Control of CPU

Date / /
Page

RANKA

△ IR → (Instruction Register)

→ It holds the current instruction that is being Executed by the process.

✡ DR → (data Register)

. It holds the Data.

✡ TR →

→ It holds the Temproary Data

→ Unique - Processor → Multitasking

↳ process.

↳ Every process has Process Counter

↓

Software

Program Counter

↳ Hardware

Stores the address Of Last Instruction done

→ Maintain Switching

• Priority -

↳ Process → Priority assigned -

Java - setPriority ( ) .

OS - assign - by default Priority

Algorithm } - Priority

• General purpose Register -

↳ Attached with CPU. → Hardware.

↳ Process → Register - Allocated

R0
R1
R2
⋮
Rn

Store Temporary Data

• PSW → (Program state word)

Date / /
Page

RANKA

- Protection information

↳ Importance → Protection assigned to importance
↳ Security Provided

Process → OS → Assign Buffer → Block - Small
↳ 2KB / 4KB

PCB → Process control Block

→ Every Process will have its own PCB
↓
all information Stored

All the information related to a process is present in PCB

Every Process has its PCB

Priority Based Scheduling → Priority → PCB
taken from

- ~~xxxx~~ ~~xxxx~~

Date / /
Page

RANKA

✡ Process - State Diagram -

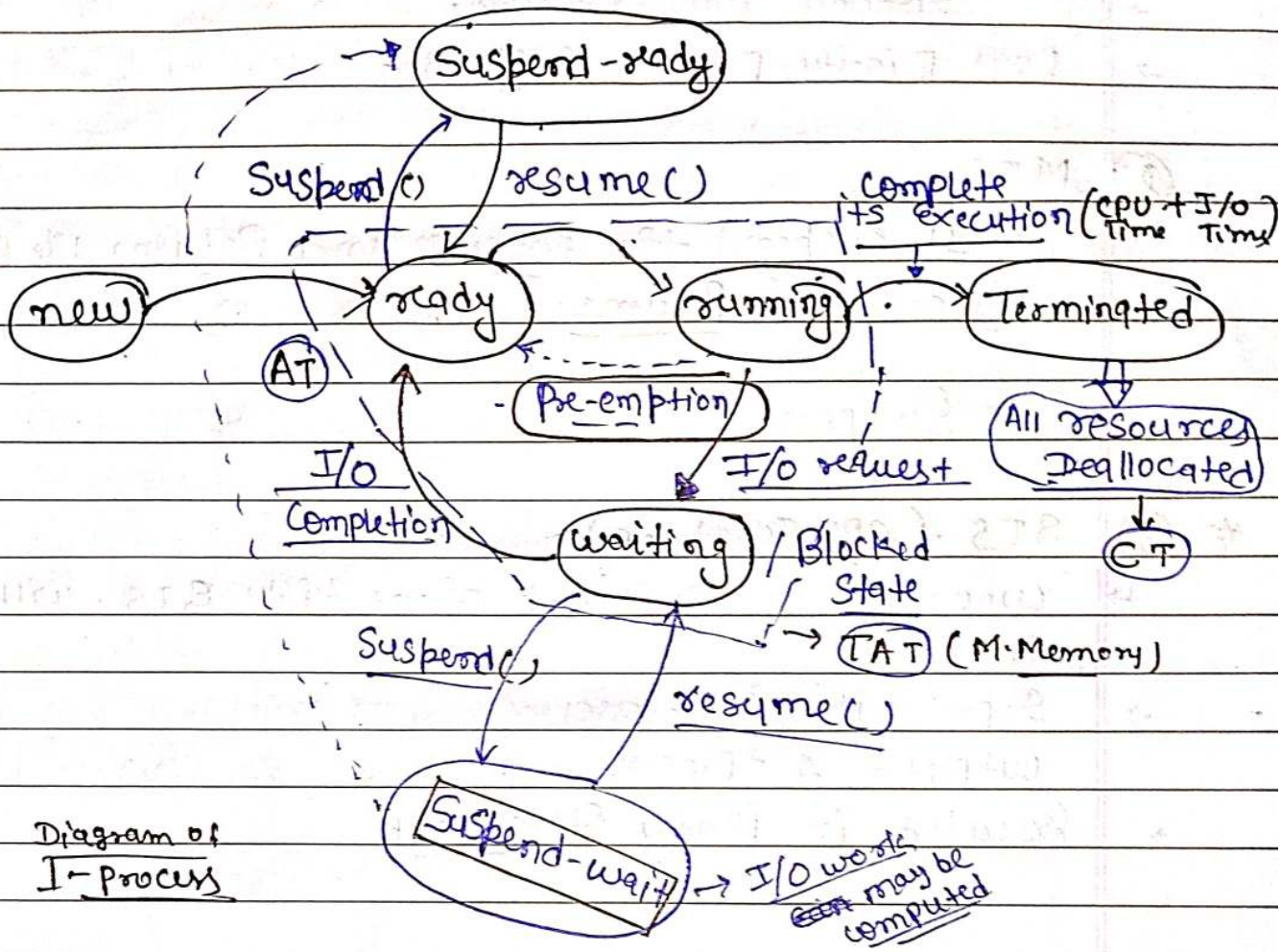


Diagram of I-Process

1. Long - term scheduler (LTS)
2. Mid - term Scheduler. (MTS)
3. Short - term Scheduler. (STS)

→ Sub-routine

? Or (CPU Scheduler)

→ They are the modules of the operating systems.
→ fixed program – Cannot be changed.

① LTS -

→ It will bring the program from the secondary Memory to Main memory.
→ Degree of Multiprogramming → (LTS) will alway increase the Degree of Multiprogramming.
→ Each time LTS, Executed, ~~new program will be~~

→ Memory Shortage.
→ ↳ Suspend some Processes.
→ Low Priority process suspended.

(2) MTS →
↳ It suspend the process and Resume the process
Suspend & Resume.

e.g - Swapper

* (3) STS (CPU Scheduler)
↳ Who will get the control of the CPU, STS will decide it.
→ Input - No. of processes.
Output - 1 - Process.
→ Executed in Ready State only

• Dispatcher -
↳ It Dispatches the CPU ~~shift~~ between the processes.
→ Provide CPU
→ Context Switching is also done by the Dispatcher.

Date / /
Page

- Execution Time -

Burst time = CPU Time + I/O Time

The ~~time~~ duration of time that is required by a process to complete its CPU work.

The duration of time that is required by a process to complete its I/O works.

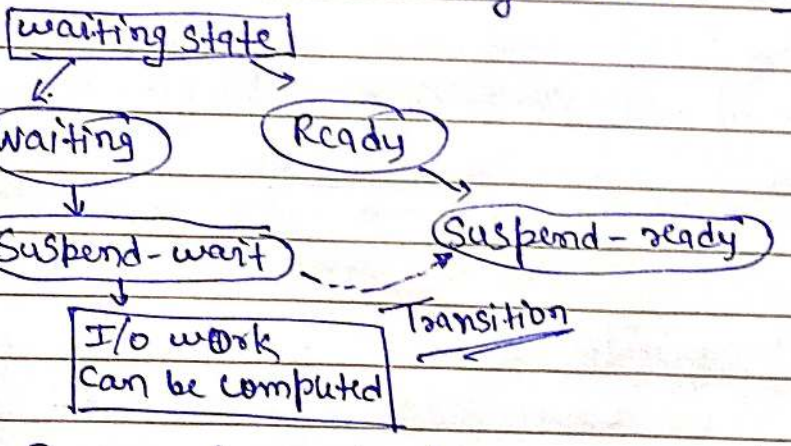
- Ready, waiting → waiting to get the control of the CPU is the waiting state.
  - waiting → for I/O work completion

- Pre-emption -

Pre-empt → because of some Priority.
↳ जबरदस्ती

Non-pre-empty → No-Priority

→ Process which is waiting will be Suspended.



→ If Process have to take CPU, it has to come to ready state to take control of CPU..

Date / /
Page

Q:- Consider there are n- processor in a System. and maximum m- Processes can be present in the System. What could be the minimum and maximum no. of Processes that could be present in the ready, Running and Waiting State.

	min	max
ready	0	m
running	0	n
Waiting	0	m

(n) - Proc.
n- Proces → Ready
m- Proc - Maxin

m = 10
n = 3

No. of Processors - Can restrict = Maximum processes in Running State.

Can't Say - Depends on the Size of Ram
↳ if maximum is not given.

Date / /
Page

- **Arrival time (AT)**

↳ The time when the process arrives in the ready queue is known as Arrival time of a process. - Exact time.

↳ Every process has its own arrival time.

- **Burst time (BT) (CPU Time)**

→ The amount of CPU time required by a process for its completion is known as Burst time of a process.

↳ Duration of time ~~for~~ for Execution of process.

- **Completion time (CT).**

→ The time when the process completes its Execution. is known as completion time of a process.

→ CPU work + I/O work - Time.

→ When I/O time is 0 (negligible), then Execution time is equal to Burst time.

- **Turn Around Time (TAT)**

↳ The time difference between the completion time & arrival time of a process is known as Turn around time of a process.

↳ i.e. $TAT = (CT - AT)$

↳ Time = Ready + Running + waiting.

- **waiting time - (WT)**

→ The time difference b/w T.A.T and Burst Time of a process is known as waiting time of a process.

i.e.

$$W.T = TAT - BT$$

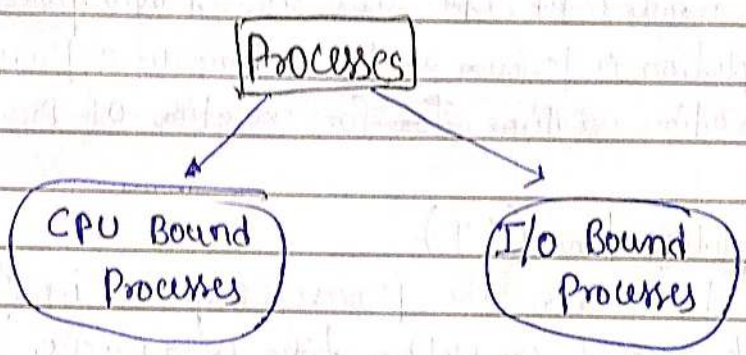
Ready + running + waiting — running

Date / /
Page

- **Response Time (RT)**

↳ The time difference b/w the 1st response and Arrival time of a process is known as Response Time of a process.

(1st response) means the time when the process gets the control of the CPU for the 1st time.

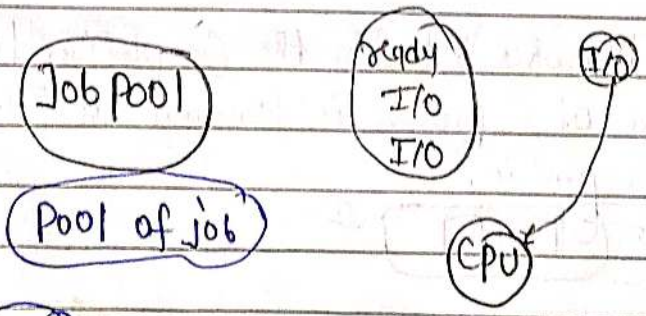


① **CPU Bound**

→ Those processes who spent more time doing the CPU work.

② **I/O Bound -**

→ Those processes who spent more time doing the I/O work.



(LTS) → Choose the process of I/O and CPU work both, for that Efficiency will be greater.

Date / /
Page

RANKA

- **Context Switching** –

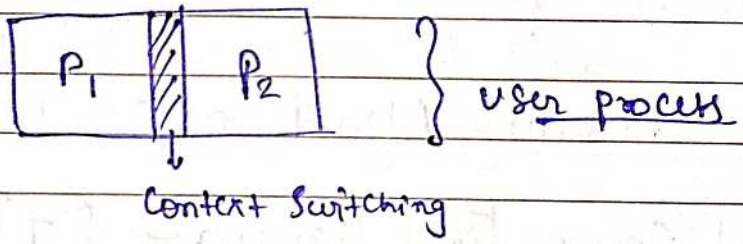
↳

P_1	P_2
~~running~~	~~ready~~
waiting	running

→ The control of the CPU is given to another process by context switching from one process to another.

→ whenever, there is context switching, Cremerally, two process will be involved in context switching.
- ↳ state change
- ↳ ~~At~~ changes in the Attributes.

↳ It is one type of (overhead) for the system.




```
CPU Scheduler ( )
{
   .
   .
   .
   dispatcher( );
}
```

```
dispatcher( )
{
   .
   .
   .
}
```

Date / /
Page

RANKA

• CPU Scheduling –

who :- (STS) or (CPU Scheduler)

where :- (Ready Queue) / State

when :- (1) (i) (Running) → Terminated
(ii) Running → waiting
(iii) Running → Ready
Pre-emption

} होगा ही !
CPU Take back

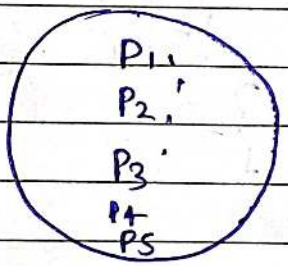
(2) (i) new → ready
(ii) waiting → Ready

} हो सकता है !

CPU Scheduling Algorithms :—

(1) First Come First Served – (FCFS)

Criteria :- (Arrival Time (AT))



$P_1 \leftarrow P_2 \leftarrow P_3 \leftarrow P_4 \leftarrow P_5$

Tie- (PID) → Lowest – 1st
Older Process → 1st in queue

mode :- (Non-premptive mode)

Date / /
Page

Q.

P.N	A.T	B.T	CT	T.A.T	W T
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6
				34/5	19/5

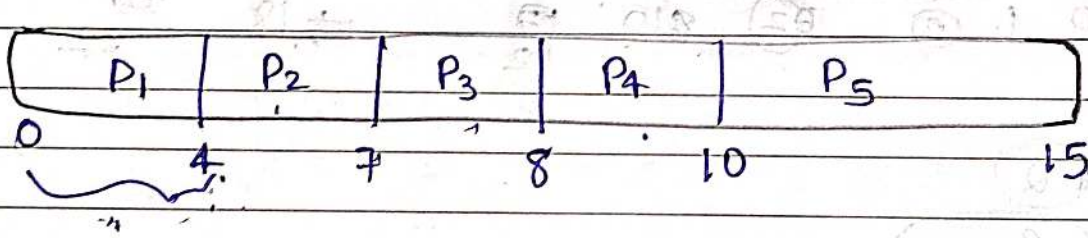
Avg. TAT = ?

Avg. WT = ?

RQ

P1
P3
P4
P5

Graph chart →

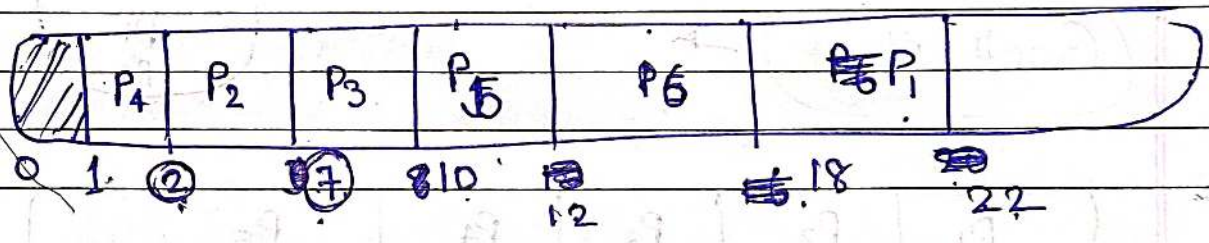


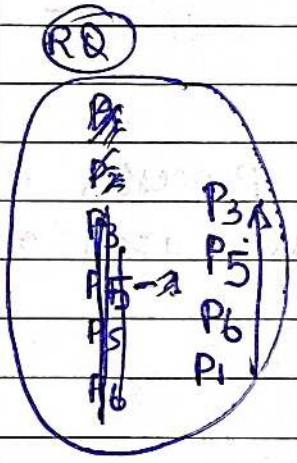
Note - if the arrival time of the processes are same, then Schedule the process based on the Lowest Process ID.

Q.

P.N	A.T.	B.T	CT	TAT	WT
1	6	4	22	16	12
2	2	5	7	5	0
3	3	3	10	7	4
4	1	1	2	1	0
5	4	2	12	8	6
6	5	6	18	13	7
				50/6	29/6

Gnatt chart -



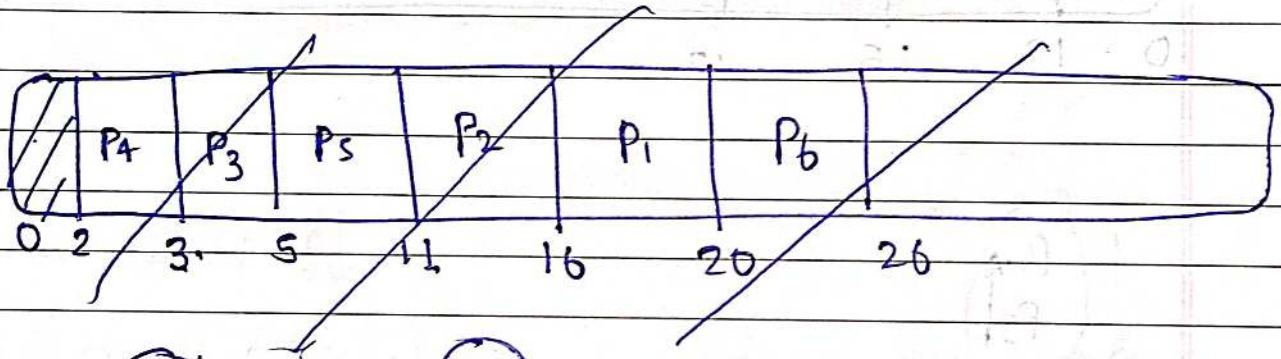


Date / /
Page

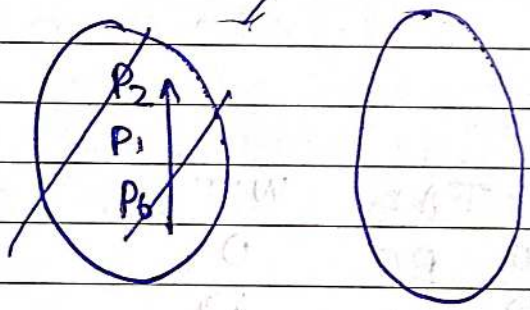
Q.

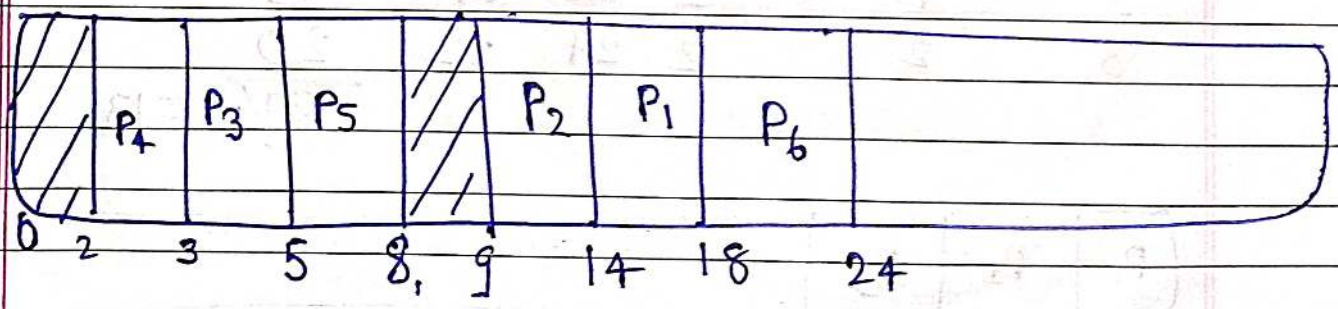
P.N	AT	BT	CT	T.AT	W.T
1	10	4	18	8	4
2	9	5	14	5	0
3	3	2	5	2	0
4	2	1	3	1	0
5	3	3	8	5	2
6	11	6	24	13	7
				34/6	13/6

Gnatt chart -



34/6
39/6

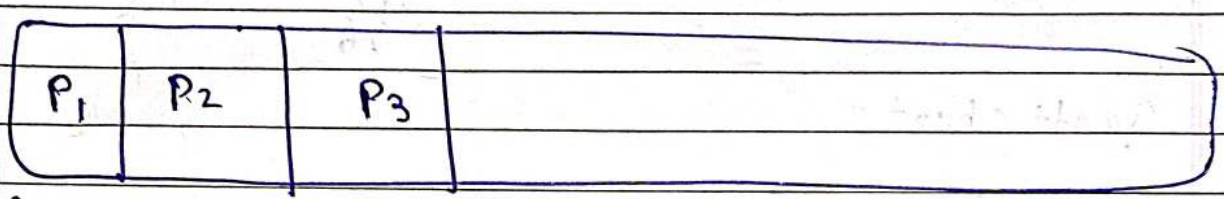




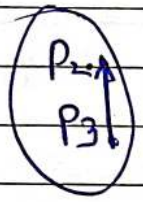
Date / /
Page

Q.

P.N	A.T.	B.T	CT	T.A.T ~~W.T~~	W.T
1	0	13	13	13	0
2	1	2	15	14	12
3	2	2	17	15	13
				42/3	25/3
				= 14	= 8.33

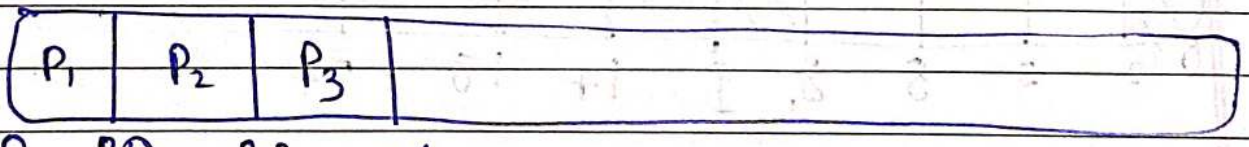


0 13 15 17



Q.

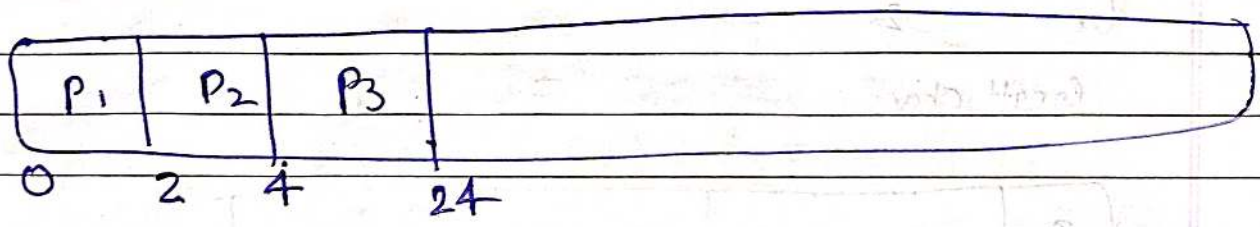
P.N	A.T	B.T	CT	T.A.T	W.T
1	0	20	20	20	0
2	1	2	22	21	19
3	2	2	24	22	20
					39/3 = 13



0 20 22 24

Date / /
Page

P.N	A.T	B.T	C.T	T.A.T (CT-AT)	W.T (TAT-BT)
1	0	2	2	2	0
2	1	2	4	3	1
3	2	20	24	22	2
				27/3	3/3 = (1)



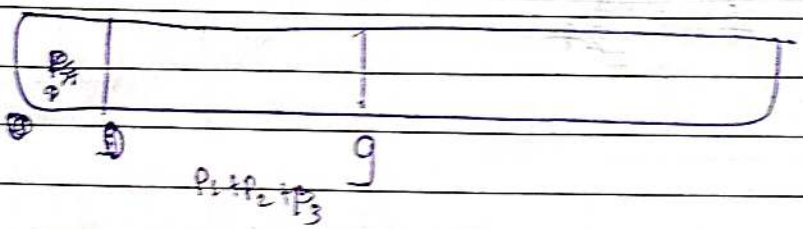
- **Convoy Effect →**

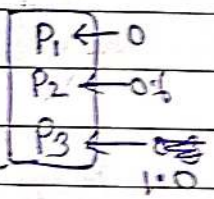
if the FCFS, if the B.T of the 1st process, is very very large, then it will have a huge impact on the average waiting time of the processes. This is known as Convoy Effect.

Date / /
Page

Q.	Job Id	BT	CT	Job	A.T	BT	WT?
	P	4		1	0.0	9	
	Q	1		2	0.6	5	
	R	8		3	1.0	1	
	S	1					
	T	2					

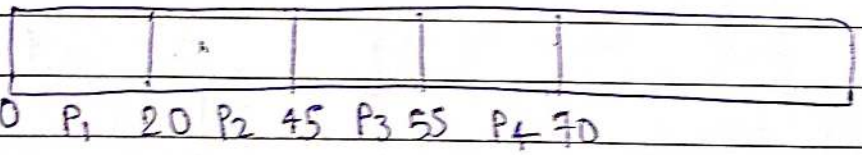
Gnatt chart





Q. SRT- Algo ?

Process	Ex. T	A.T
P_1	20	0
P_2	25	15
P_3	10	30
P_4	15	45



waiting time of P_2 = 5 sec.

Date / /
Page

RANKA

- **Starvation** -

→ Indefinite postponement of the Execution of the Processes is known as Starvation.

→ when → Control of CPU → processes ? given.

Deadlocks.

✯ Shortest job first (SJF) :-

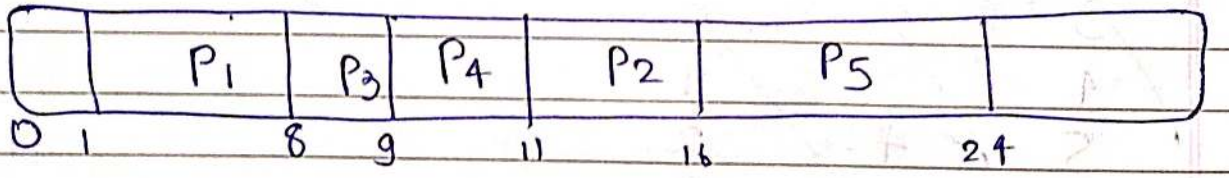
Criteria - (Burst Time) }
Mode - (Non-premptive) }

Small Burst Time ! given priority

Q.

Process	AT	BT	CT	TAT (CT-AT)	WT (TAT-BT)
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11
				53/5	30/5 = 6ms.

Gnatt Chart -



NOTE - if the B.T of the Processes are matching then Schedule the process Based on the Arrival Time.

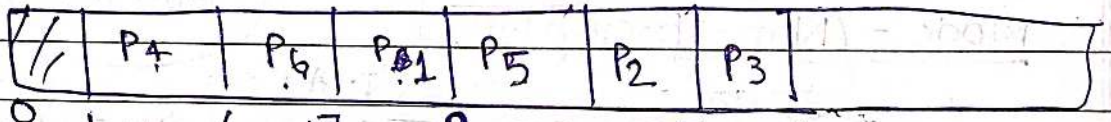
CT - AT

Date / /
Page

Q.

PID	AT	BT	CT	TAT	WT (TAT-BT)
1	6	1	8	2	1
2	3	3	13	10	7
3	4	6	19	15	9
4	1	5	6	5	0
5	2	2	10	8	6
6	5	1	7	2	1
				42/6 = 7	24/6 = 4

Gr. C



P4 RQ

P1, P2, P3
P4, P5

Q.

PID	AT	BT
1	0	7
2	1	5
3	2	3
4	3	1
5	4	2
6	5	1

Date / /
Page

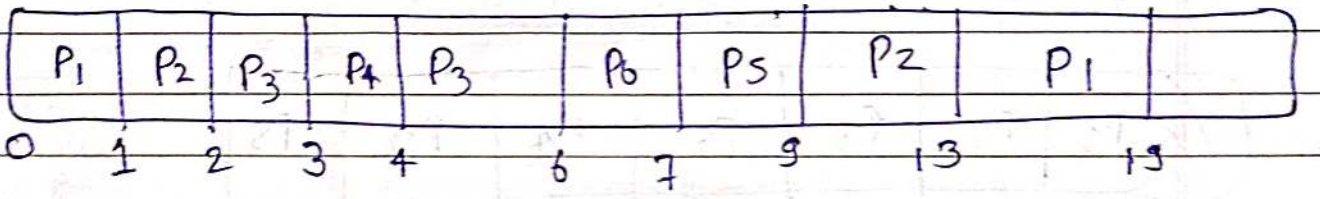
☆ Preemptive Shortest Job first – (SRTF)

(Shortest Remaining Time first)

Criteria ⇒ (Burst Time)

Mode ⇒ (Preemptive)

Gnatt Chart



Q.

PN	A·T	BT	CT	TAT (CT-AT)	WT (TAT-BT)	RT
1	0	6	19	19	12	0-0
2	1	8	13	12	4	1-1 = 0
3	2	3	6	4	1	2-2 = 0
4	3	1	4	1	0	3-3 = 0
5	4	2	9	5	3	7-4 = 3
6	5	1	7	2	1	6-5 = 1
				43/6	24/6 = 5	= (4)

48.

Date / /
Page

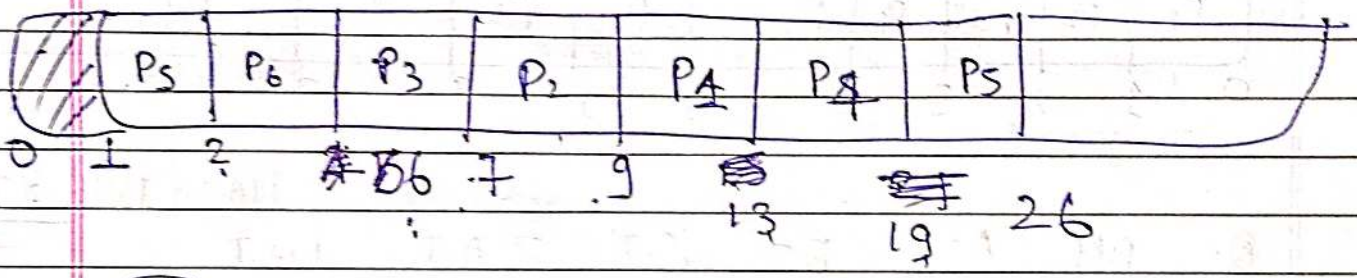
RANKA

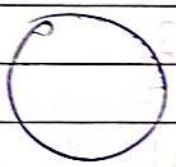
CT-RJ

Q.

PID	AT	BT	CT	TAT	WT	TAT − BT
1	3	4	23	10	6	
2	4	2	9	5	3	
3	5	1	7	2	1	
4	2	6	19	17	11	
5	1	87	26	25	18	
6	2	4 3 2 1 0 6		4	0	
				63/6	39/6	

Gnatt ch.





P3 ← P2 ← P1 ← P4 ← P5

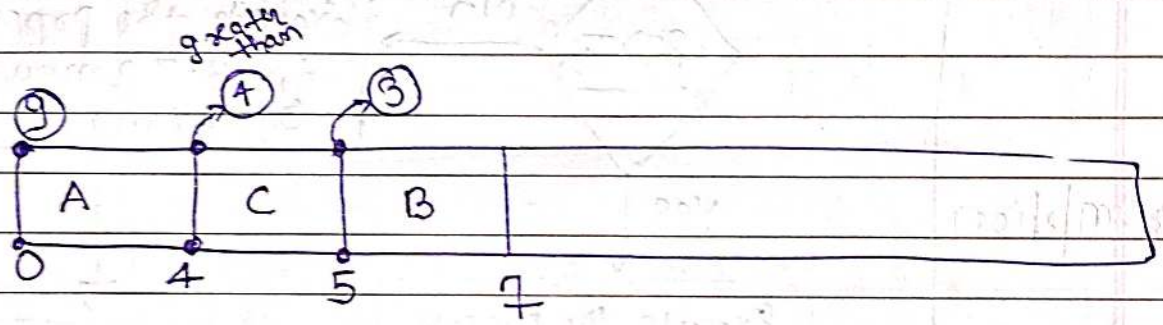
Date / /
Page

RANKA

Q. Consider three processes - A, B, and C known to be Scheduled as per SRTF Scheduling Algorithm. A has run for 4 units of Time, then process C Arrives, when C has run for 1 unit of time, then process B Arrives and it completed its Execution running 2 units of time. what is the minimum Burst time of A and C respectively.

Remind

PID	AT	BT
A		
C		
B		2 0



Minimum B.T of A and C

Remaining B.T of C > B.T of B

BT of C > 2

B has prempted C

minimum B.T = 3

⟹ Remaining B.T of A > B.T of C

B.T of A > 4 → minimum

greater than 4 → = 5

5 + 4 = 9

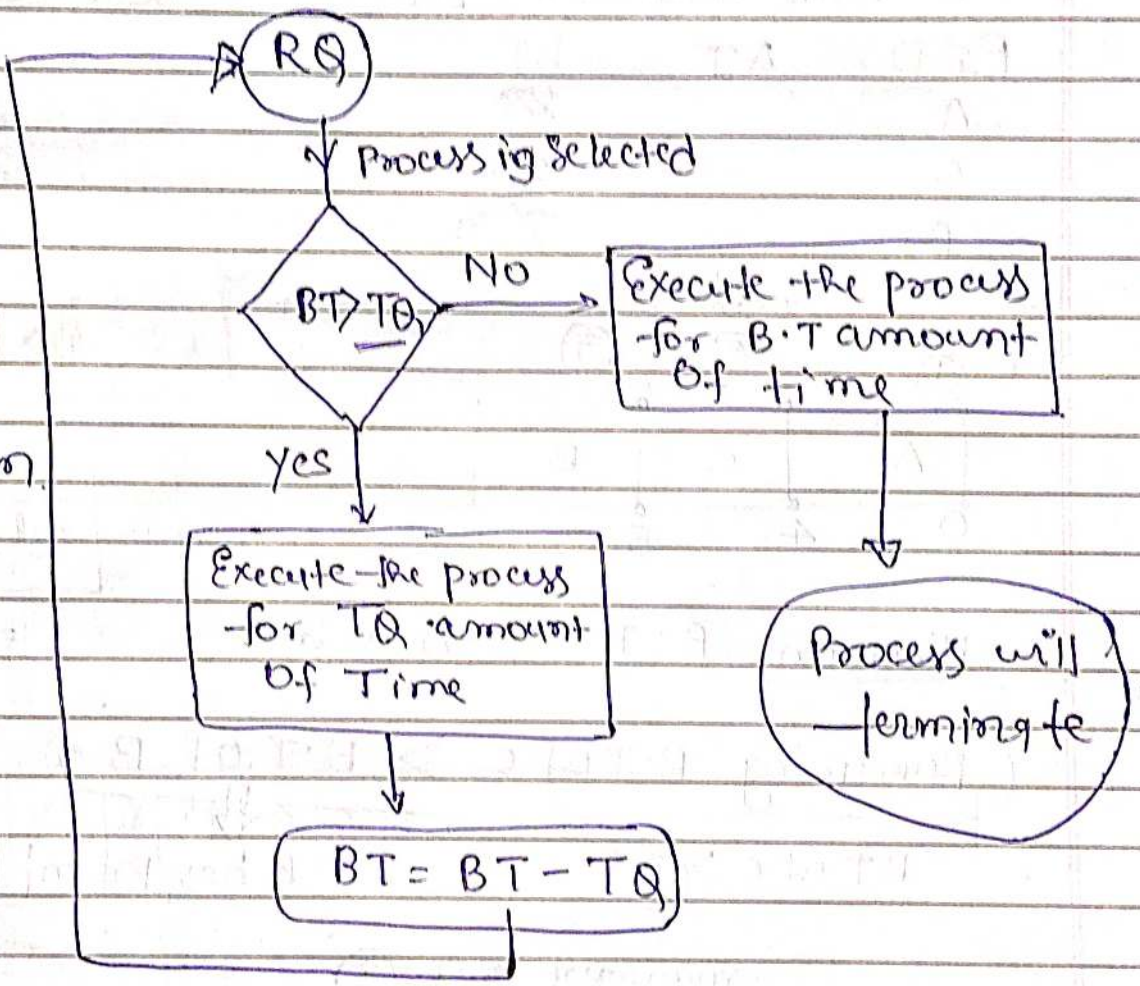
Date / /
Page

Scheduling

☆ Round Robin Algorithm -

Criteria → (Arrival time / Time Quantum)

Mode → (Preemptive)



→ Less TQ ↓

→ It is used for Interactive System / Application

Date / /
Page

Q.

PID	AT	BT
1	0	~~4~~ 2 0
2	1	~~5 3~~ 1 0
3	2	2 0
4	3	1 0
5	4	~~6 4~~ 2 0
6	6	3 1 0

TQ = 2.

Ready Queue

~~P2~~ ← ~~P3~~ ← ~~P1~~ ← P4 ← P5 ← P2 ← P6 ← P5
2 2 2 3 4

P1	P2	P3	P1	P4	P5	P2	P6	P5	P6	P6	P5	

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

* Always New ~~process~~ process's first priority in the Ready queue.

New to Ready → more priority than Running to Ready

Overhead - wastage

Date / /
Page

Q.

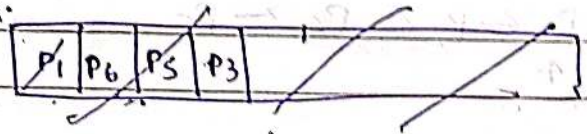
PID	AT	BT	CT	TAT	RT
1	3	~~20~~			
2	2	~~20~~			
3	6	~~1~~			
4	8	10			
5	4	~~1~~			
6	3	~~20~~			

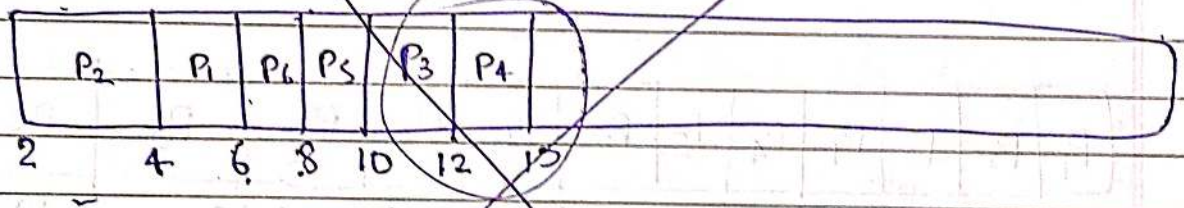
TQ = 2

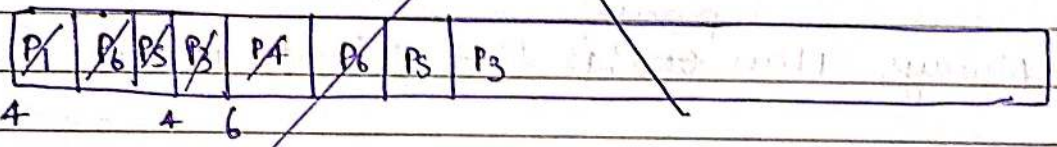
Round Robin

Avg. TAT? Avg. RT? Avg. WT?

RQ.



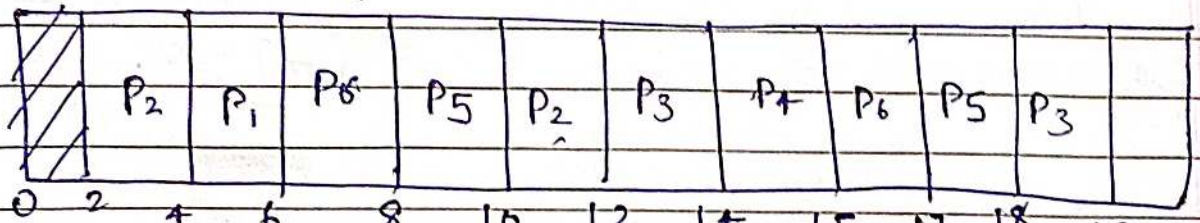




RQ

$P_1 \leftarrow P_6 \leftarrow P_5 \leftarrow P_2 \leftarrow P_3 \leftarrow P_4 \leftarrow P_6$

4 4



Date / /
Page

→ if TQ ↓, then no. of Context switches increases ↑

→ if TQ ↑ then no. of Context switches decrease ↓

→ if TQ ↓ then average response time ↓ decrease

→ if TQ ↑ then average response time ↑ increases.

→ If TQ is very very Large, it will behave like a FCFS (first come first serve Algorithm).

✡ Priority based Scheduling →

a) Non-preemptive priority Scheduling.

Criteria :- (priority No.)

Mode :- (Non-preemptive)

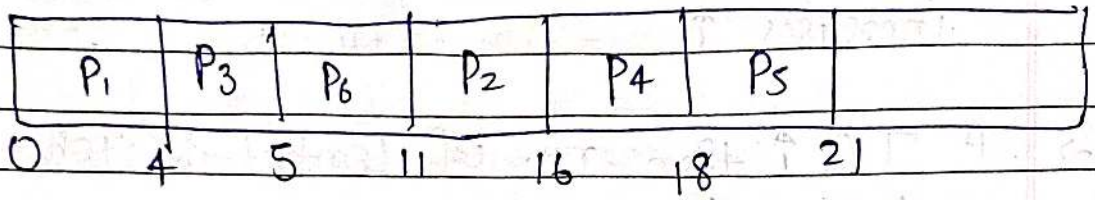
e.g -

	Priority	P.N	AT	BT
	4	1	0	4
	5	2	1	5
highest ←	7	3	2	1
	2	4	3	2
Lowest ←	1	5	4	3
	6	6	5	6

P2, P3, P4, P5 ← 4 unit of Time

P3 ← P2 ← P4 ← P5 ← 4 Unit

P6

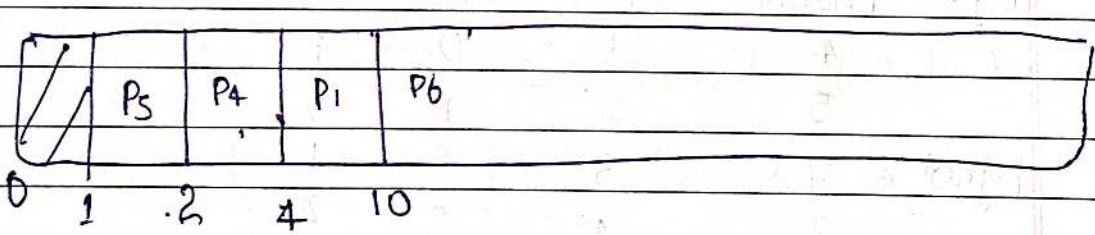


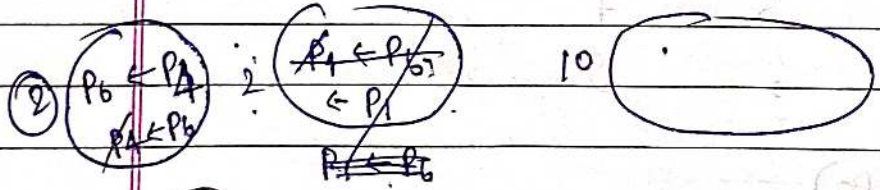
Note →

If the priority of the processes are matching then Schedule the process based on the arrival Time.

Q.

Priority	PID	AT	BT
4	1	4	6
5	2	6	3
6	3	3	4
6	4	2	2
High ← 7	5	1	10
Low ← 3	6	2	2





$P_4 \leftarrow P_3 \leftarrow P_6 \leftarrow P_1$

Date / /
Page

RANKA

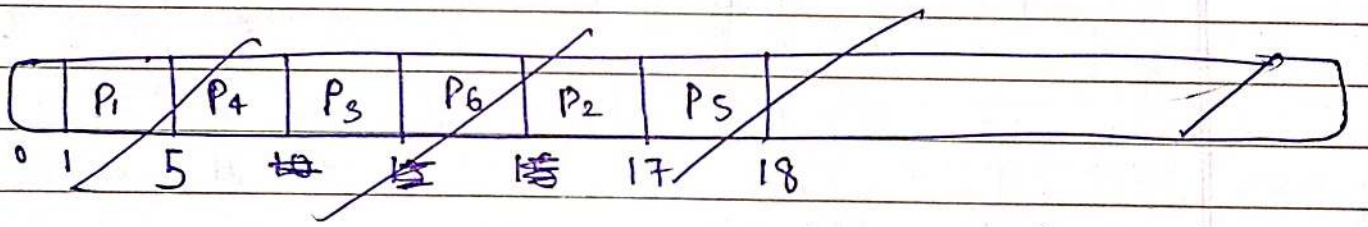
(6) Preemptive priority Scheduling -

Criteria - Priority NO.
Mode - Premptive

Q.

	Priority	PID	AT	BT
Low ←	4	1	1	4 3 0
	5	2	(2)	2 0
	7	3	2	3 2 0
High ←	8	4	3	8 0
	5	5	3	1. 0
	6	6	4	2 0

$P_4 \leftarrow P_3 \leftarrow P_6 \leftarrow P_2 \leftarrow P_5 \leftarrow P_1$

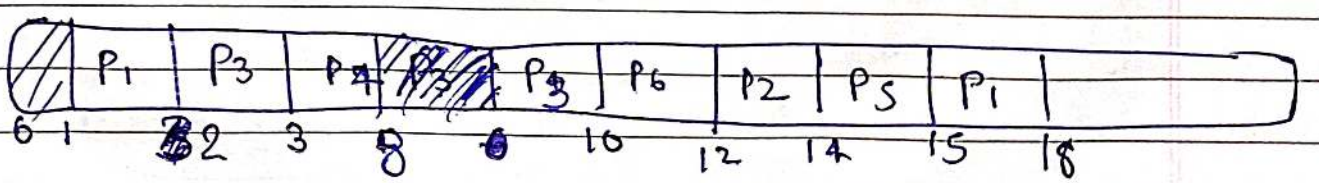


RQ ← ~~$P_1 \leftarrow P_2 \leftarrow P_3 \leftarrow P_4 \leftarrow P_5 \leftarrow P_6$~~ $P_3 \leftarrow P_5$ | ← $P_4 \leftarrow P_5$ | P_6

~~$P_2 \leftarrow P_3 \leftarrow P_4 \leftarrow P_5 \leftarrow P_6$~~

2 3 4

Sec. ~~$P_1 \leftarrow P_3 \leftarrow P_6 \leftarrow P_2 \leftarrow P_5$~~



Date / /
Page

RANKA

At $i=0$ RQ = $P_3 \leftarrow P_2$

$i=3$ RQ = $P_4 \leftarrow P_3 \leftarrow P_2 \leftarrow P_5$

$i=4$ RQ = ~~$P_3 \leftarrow P_2 \leftarrow P_5$~~ ~~$P_4 \leftarrow$~~ $P_4 \leftarrow P_3 \leftarrow P_6 \leftarrow P_2 \leftarrow P_5$

~~$P_3 \leftarrow P_4 \leftarrow P_6 \leftarrow P_2 \leftarrow P_5$~~

$P_4 \leftarrow P_3 \leftarrow P_6 \leftarrow P_2 \leftarrow P_5$

OS-Lab

Date / /
Page

(1) Program of FCFS -

Input
- n - processes — input — Array
- AT
- BT

Gnatt chart
↳ Computation

AT - Scheduling
Array
Temp no ~ BT : till
Remove

Process
Compute
↳ Array
↓
CT → Store
↓ Temp. regis
Input

Select - AT → BT amount of time
↳ Add + BT

TAT ?
WT ?

FCFS

```
#include <iostream>
using namespace std;
int main()
{   int AT
```

(7)

Date / /
Page

RANKA

```
#include <stdio.h>
#include <stdlib.h>
#define max 30
void main()
{ int i,j, n, bt[max], at[max], wt[max], tat[max]
    temp[max];
float avgtat = 0, avgwt = 0;
printf("Enter the no. of processes");
Scanf("%d", &n);
printf(" Enter the burst time of process");
for(i=0; i<n; i++)
        Scanf("%d", &bt[i]);
printf(" Enter the arrival time of process");
    for(i=0; i<n; i++)
      Scanf("%d", &at[i]);
temp[0] = 0;
printf("Process \t Burst time \t Arrival time \t
        waiting time \t Turn around time \n");
for(i=0; i<n; i++)
{ wt[i] = 0;
  tat[i] = 0;
 temp[i+1] = temp[i] + bt[i];
  wt[i] = temp[i] - at[i];
 tat[i] = wt[i] + bt[i];
 avgwt = avgwt + wt[i];
 avgtat = avgtat + tat[i];
printf("%d \t %d \t %d \t \t %d \t \t
      %d \n", i+1, bt[i], at[i], wt[i], tat[i];
}
avgwt = avgwt/n;
avgtat = avgtat/n;
```

```
printf("average waiting time %f \n", avgwt);
printf("average turn around time %f", avgtat);
}
```

- **Longest Job First** (LJF)
  - Criteria = BT
  - Mode = Non-Preemptive

(Longest BT)

Q.

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

Gantt:

P1	P4	P5	P3	P2	
0	3	8	14	18	20

Q.

PID	AT	BT	CT	TAT	WT
1	3	3	18	15	12
2	1	1	2	1	0
3	2	3	15	13	10
4	4	2	20	16	14
5	6	6	12	6	0
6	2	4	6	4	0
				55/6	36/6 = 6

	P2	P6	P5	P3	P1	P4	P2	
0	1	2	6	12	15	18	20	26

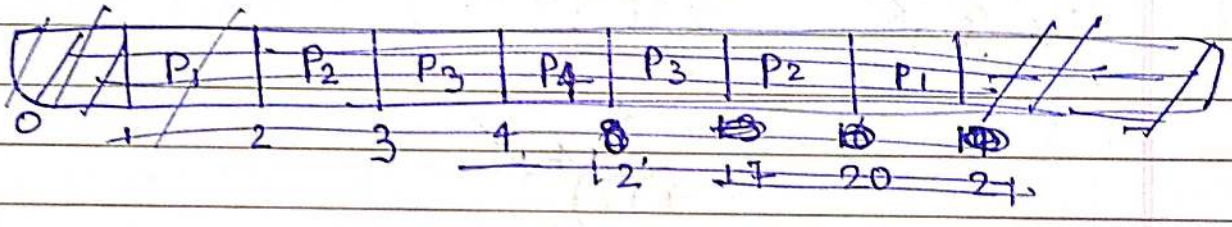
Date / /
Page

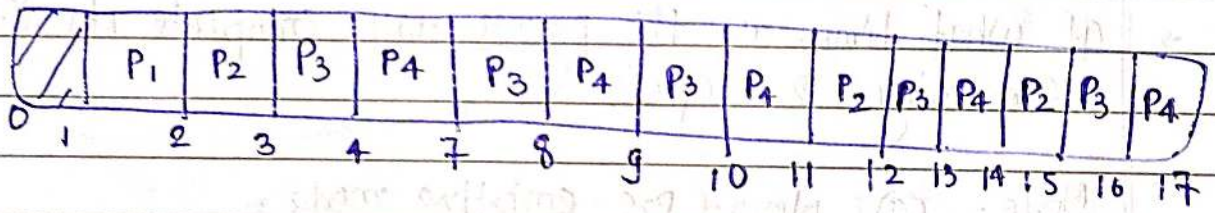
RANKA

- **Preemptive LJF (or) LRTF →**
- **Criteria – BT**
  Mode – Preemptive

Q.

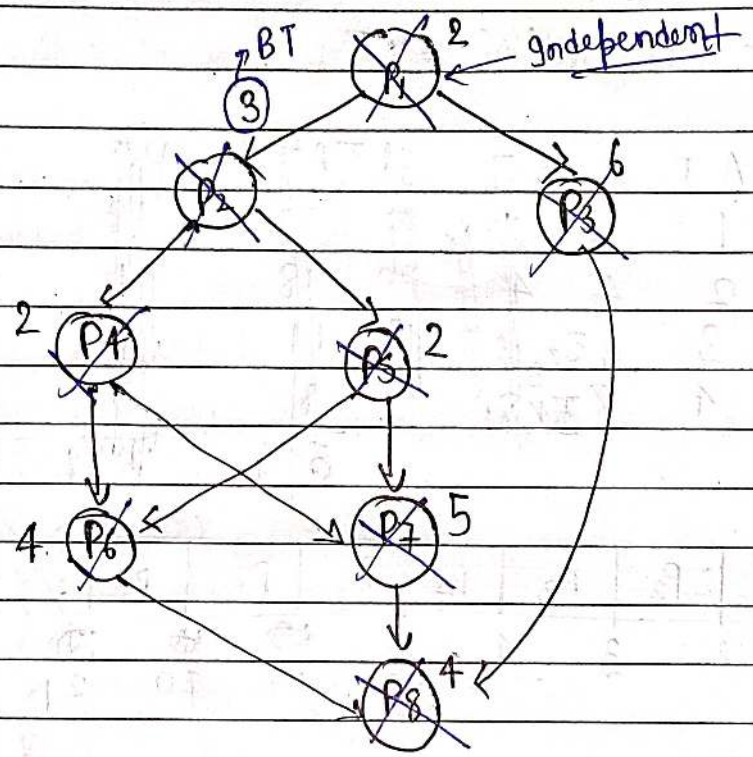
PID	AT	BT
1	1	2
2	2	4
3	3	6
4	4	8





9+4

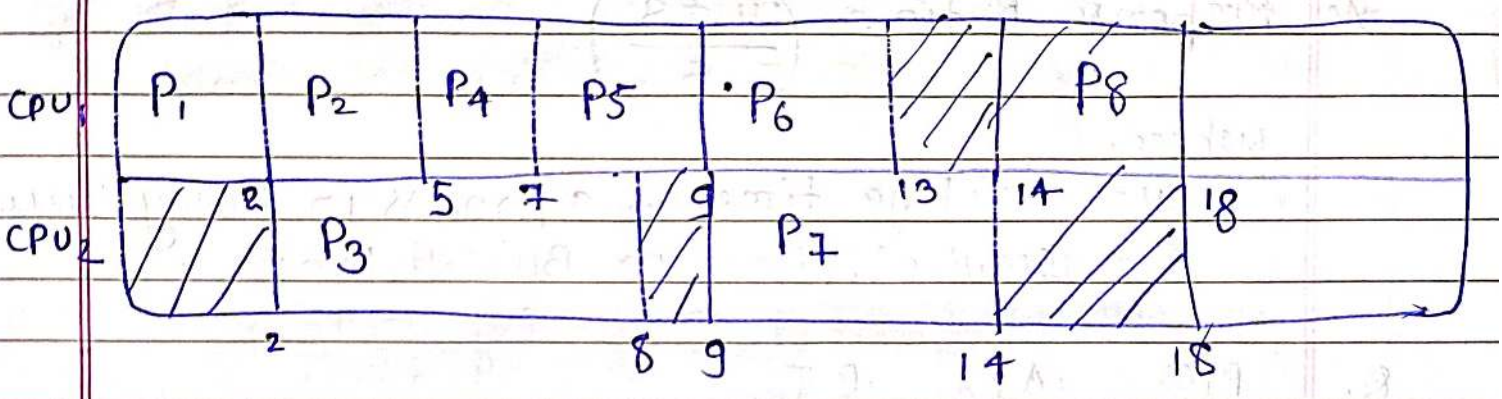
Q. Consider the following dependency graph :—



→ At what time, all the process will complete its Execution assuming 2 - CPU's ?

Note- (a) Non-pre-emptive mode.
(b) A process can't be Executed on both the CPU's

	CPU$_1$	CPU$_2$
at T=0	P_1	
T=2	P_2	P_3


CPU1
CPU2
P1
P2
P4
P5
P6
P8
P3
P7
2
5
7
8
9
13
14
18
2
8
9
14
18

Date / /
Page

• Highest Response Ratio Next (HRRN)

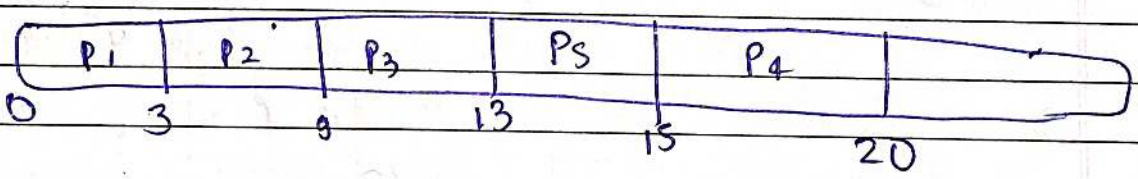
⇒ Criteria :- Response Ratio (RR)
⇒ Mode :- Non-premptive.

✲ Response Ratio = $\frac{w + s}{s}$

where,
w = waiting time of a process in Ready queue
S = Service time or Burst time

Q.

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

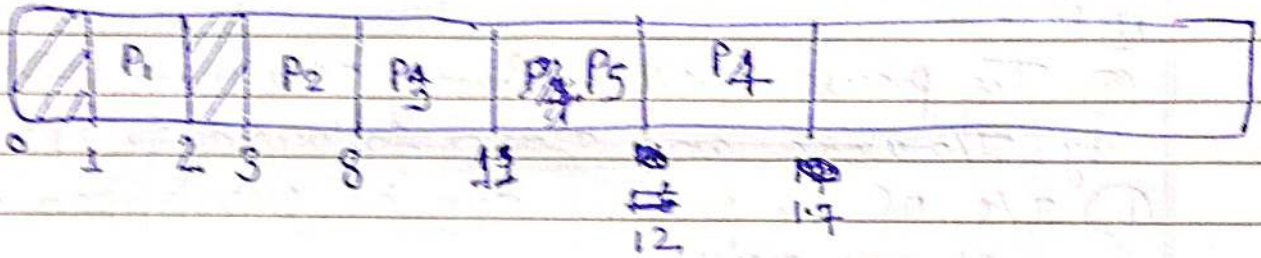


At 9
R.R of P3 = 5+4/4 = 9/4
R.R of P4 = 3+5/5 = 8/5
R.R of P5 = 1+2/2 = 3/2

At 13
R.R of P4 = $\frac{7+5}{5}$ = 12/5
R.R of P3 = $\frac{5+2}{2}$ = 7/2

Q.

PID	AT	BT
1	1	1
2	3	2
3	4	3
4	6	5
5	8	1



at 8

R.R of $P_3 = \frac{4+3}{3} = 7/3 = 2$

R.R of $P_4 = \frac{2+5}{5} = 7/5$ ✓

R.R. of $P_5 = \frac{0+1}{1} = 1$

at 12 :

R.R of $P_4 = \frac{5+5}{5} = 2$ ✓

R.R. of $P_5 = \frac{3+1}{1} = 4$ ✓

CPU I/O CPU

14/09/22

Date / /
Page

RANKA

Q.

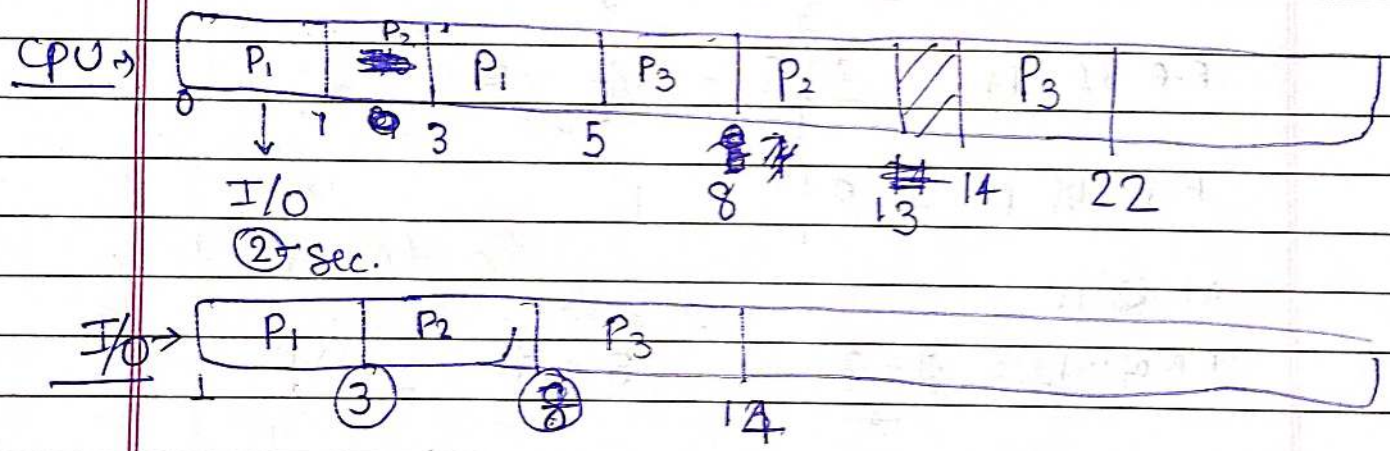
PN	AT	BT	I/O	CPU
P1	0	4	2	2
2	1	2	4	5
3	2	3	6	8

What is the completion time of the processes P_1, P_2 and P_3 using SRTF scheduling algorithm?

Note -

(a) The process first performs CPU work, followed by I/O work and again followed by CPU work.

(b) I/O of the processes can be overlapped as much as possible.



Optimal → SJF Practical → (RR, SJF)

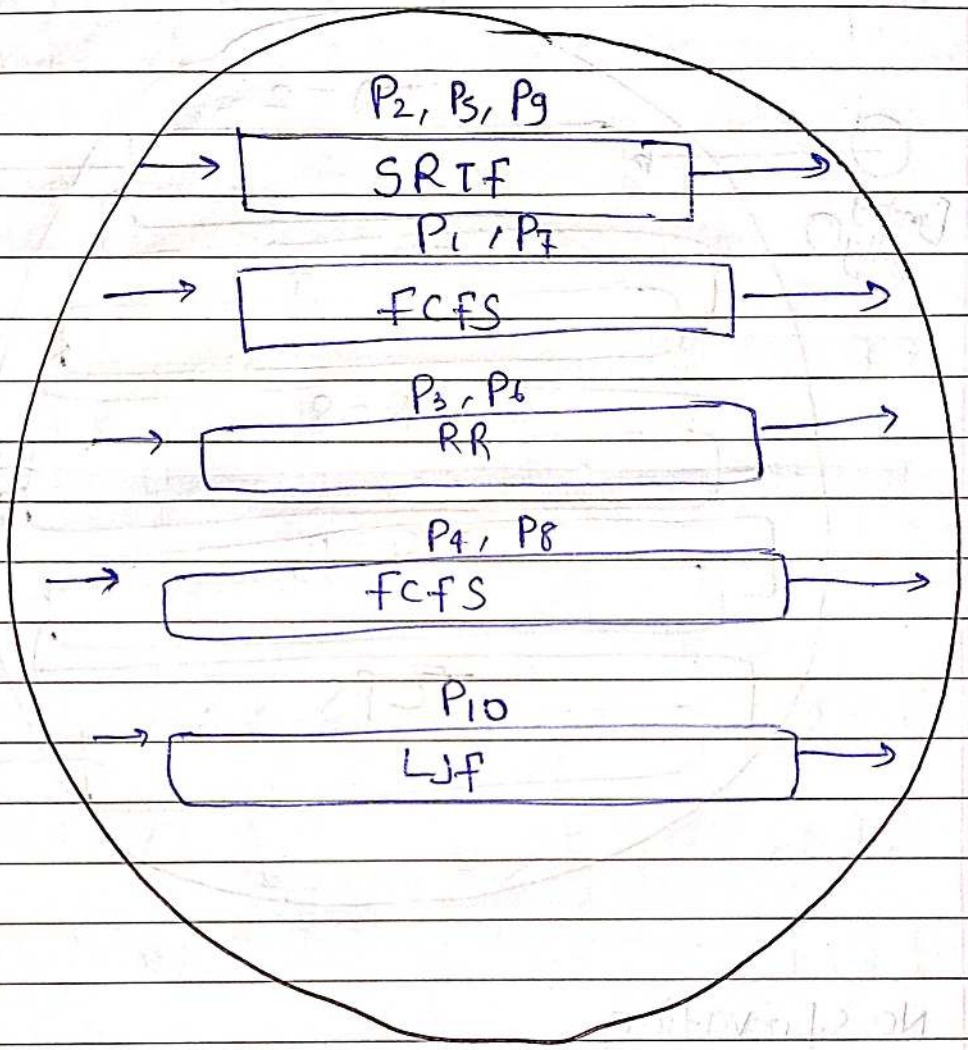
avg tat ↓ Eff ↑
avg wt ↓

Date / /
Page

RANKA

✡ Multilevel Queue Scheduling Algorithm -

$P_1 \rightarrow 2$
$P_2 \rightarrow 1$
$P_3 \rightarrow 3$
$P_4 \rightarrow 4$
$P_5 \rightarrow 1$
$P_6 \rightarrow 3$
$P_7 \rightarrow 2$
$P_8 \rightarrow 4$
$P_9 \rightarrow 1$
$P_{10} - 5$

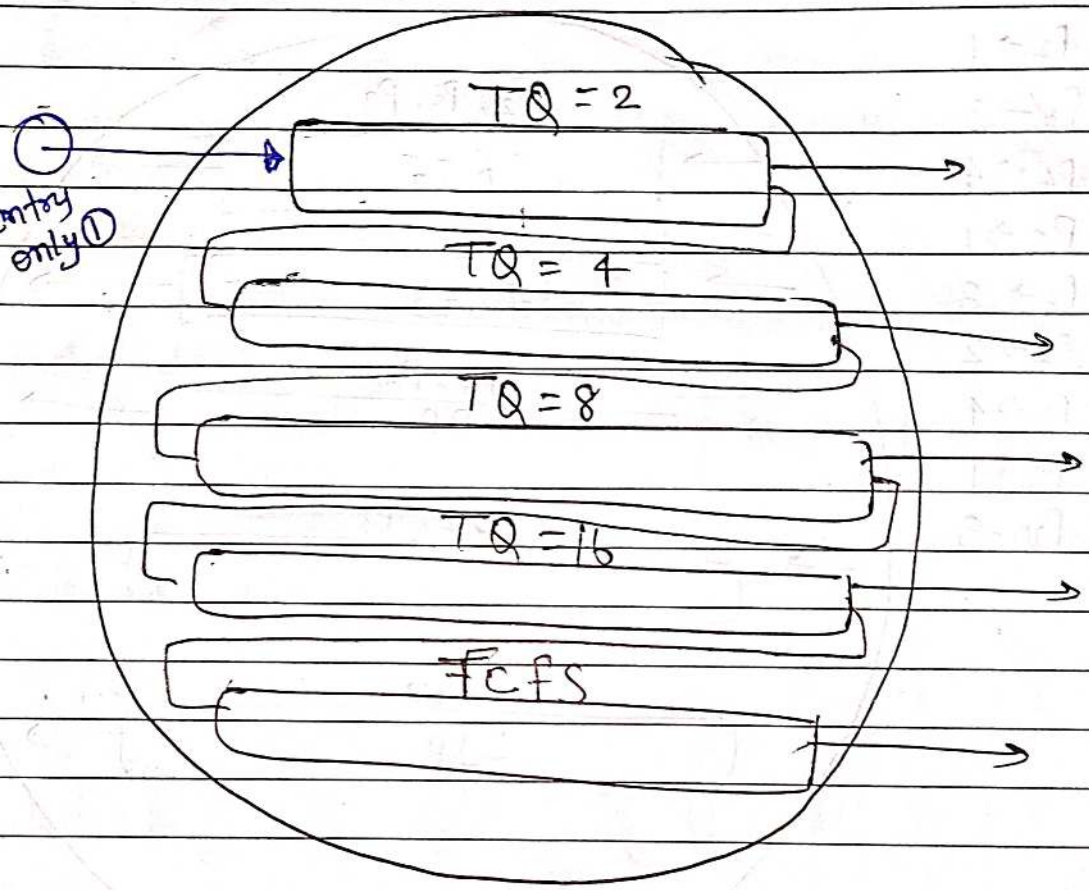


→ Starvation of Lowest priority queue.
→ Suffering from Starvation.
→ Transitions not allowed.

Date / /
Page

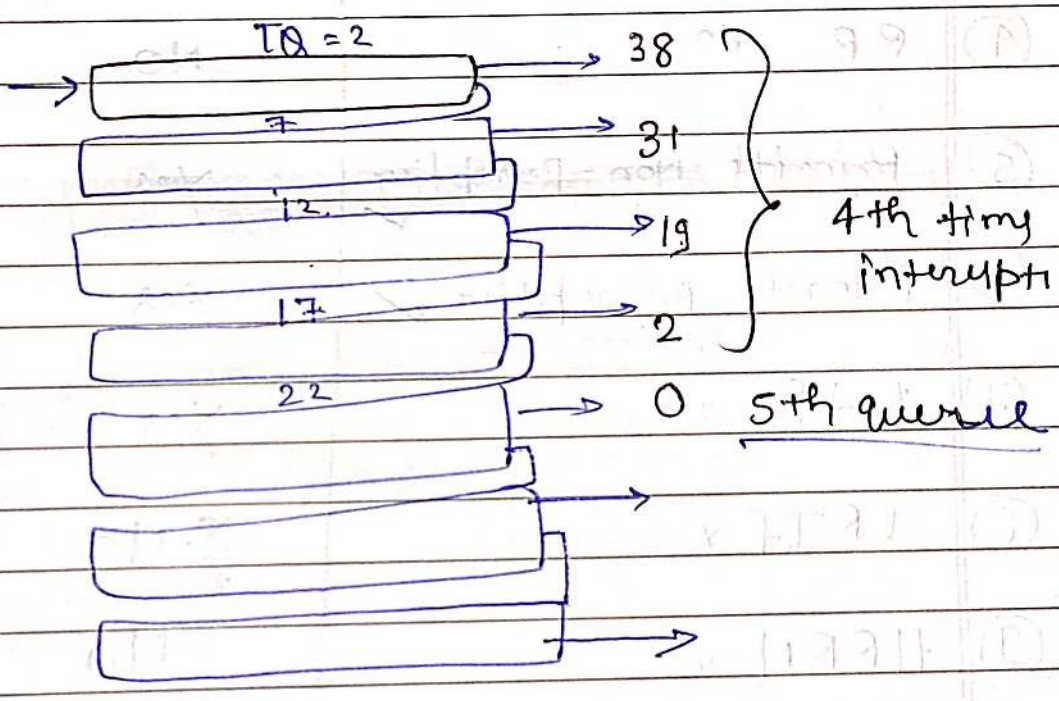
RANKA

Multilevel feedback queue scheduling Algorithm →



→ No starvation
→ Transition Allowed.

Q Consider a system, which has a CPU bound process with BT of 40 sec., the Multilevel feedback queue scheduling Algo is used and the queue time quantume is 2 sec, and in each level it is incremented by 5 sec. How many times the process will be interrupted, and in which queue the process will complete its execution.



#	Name of Scheduling Algo.	Starvation
(1)	FCFS	NO
(2)	SJF	yes
(3)	SRTF	yes
(4)	RR	NO
(5)	Priority Non-premptive	yes
(6)	Priority premptive	yes
(7)	LJF	yes
(8)	LRTF	NO
(9)	HRRN	NO
(10)	Multilevel Que. Sch. Algo	yes
(11)	Multilevel feedback Que Sch. Algo.	NO

Indefinite postponement of the execution of the process

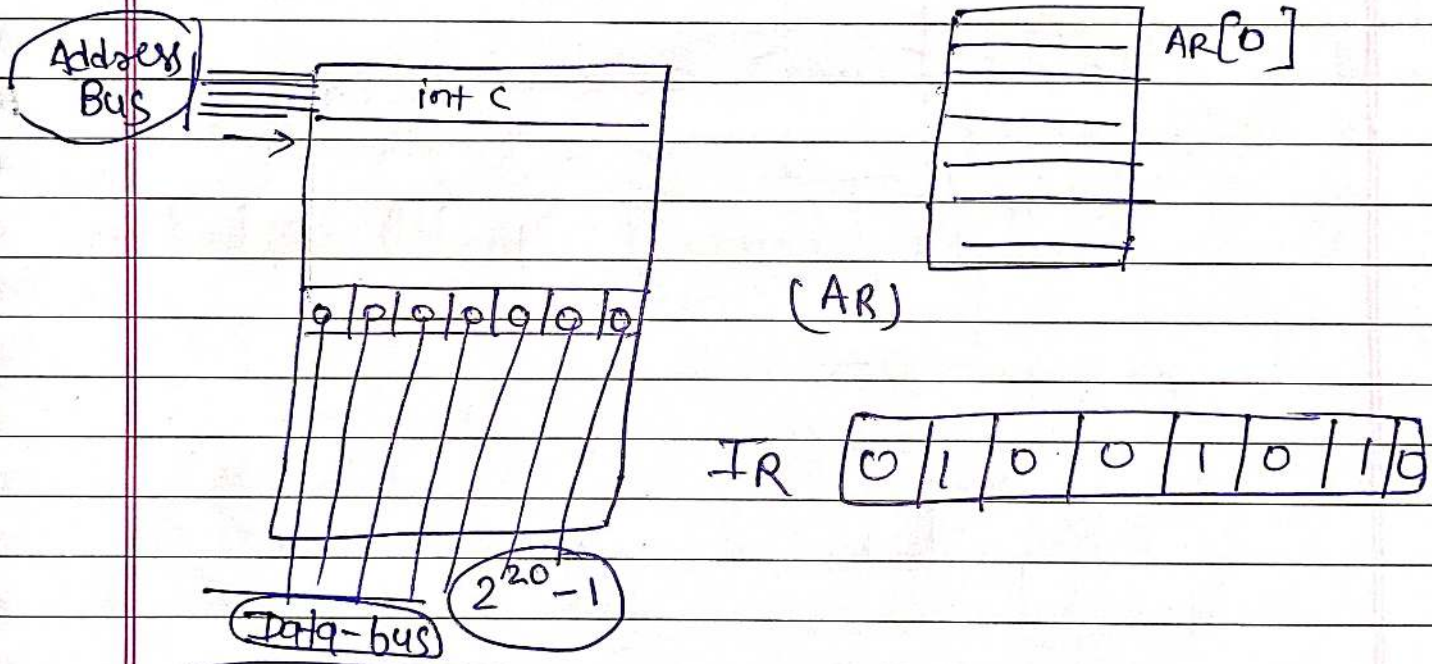
Date / /
Page

RANKA

flip-flop- Smallest storage of the memory.
↳ either 0/1.
↳ 1 bit storage

Many flip flop ⟶ Register
Many Register ⟶ RAM

{Read & write} {Load & Store}
⇓ ⇓
from the Memory to the Register
form the Register to the Memory

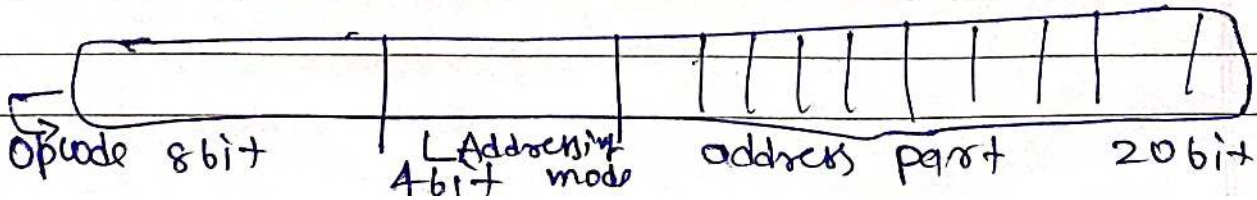


If, ID, Of, Ex, WB

$T_0 \rightarrow$ AR ← PC
$T_1 \rightarrow$ IR ← M[AR]
PC ← PC + 1;
} Microinstructions

$T_2 :\rightarrow$

32 bit



Date / /
Page

RANKA

Count = Count + 1 ;

Load Rp , M [count]
Rp ← Rp + 1
Store M [count] , Rp

} Many Microinstructions

Sequence of Microinstructions

Date / /
Page

- **Process Synchronization →**

Co-operating processes

→ Those processes that can affect the Execution of other Processes, and could be affected by the Execution of other Processes are known as Co-operating Processes.

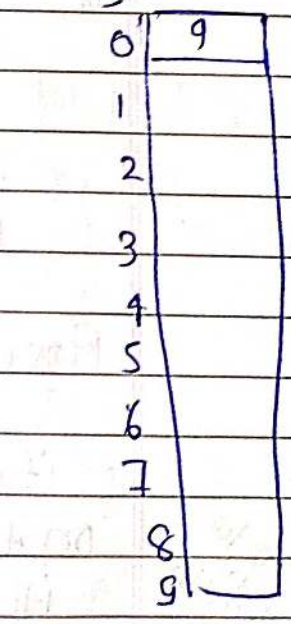
* → Something will be shared between processes.

Independent Process

→ Those processes that can not affect the Execution of other Processes, and could not be affected by the Execution of other processes are known as Independent processes.

- **Producer Consumer Problem** – (Bounded Buffer)

```
int in=0, out=0;
int Buffer[N];
int count=0;
void Producer()
{ int itemp;
itemp = Produce();
Buffer[in] = itemp;
in = (in+1) mod N;
count = count+1;
}
void Consumer()
{ int itemc;
itemc = Buffer[out];
out = (out+1) mod N;
Count = count-1;
} Consume(itemc);
```




```
while(1)
{ — —
  — —
  — —
}
```

no infinite Loop, it Can breaks.

Date / /
Page

RANKA

→ while (count == N);

⇓

while (count == N)

{

}

→ }

while (count == 0);

Synchronisation

→ Micro instruction → Atomic (indivisible)

→ Instruction → Non-Atomic)

Producer	Consumer
T_0 Load R_p, M[count]	T_1 Load R_c, M[count]
T_2 $R_p \leftarrow R_p + 1$	T_3 $R_c \leftarrow R_c - 1$
T_4 Store M[count], R_p	T_5 Store M[count], R_c

④ , ⑥ ⑤

T_0	T_1	T_1	T_0	T_0	T_3
T_2	T_3	T_3	T_2	T_1	T_4
T_4	T_5	T_5	T_4	T_2	T_5

count = ~~5~~ 6 4

Race - Condition →

→ if the output of the instructions is dependent on the ~~types~~ order of Execution of the processes. then it is called Race condition.

Order of Execution / Output unpredictable //

→ types of Execution of Microinstructions.

→ Sequence of order of Execution

Date / /
Page

- **Universal Assumption** – (To solve Synchronisation Problem)

→ The Running process can get preempted at any point of time after completing the current instruction / Micro instruction.

- **Moto** → **Consistency !**

21/09/22

→ Following three conditions should be satisfied simultaneously in order to achieve Synchronisation –

1. Mutual Exclusion
2. Progress
3. Bounded waiting

Program (P)

```
do{
    [Entry Section]
    [Critical Section]
    [Exit Section]
    [Remainder Section]
} while (true);
```

Note –

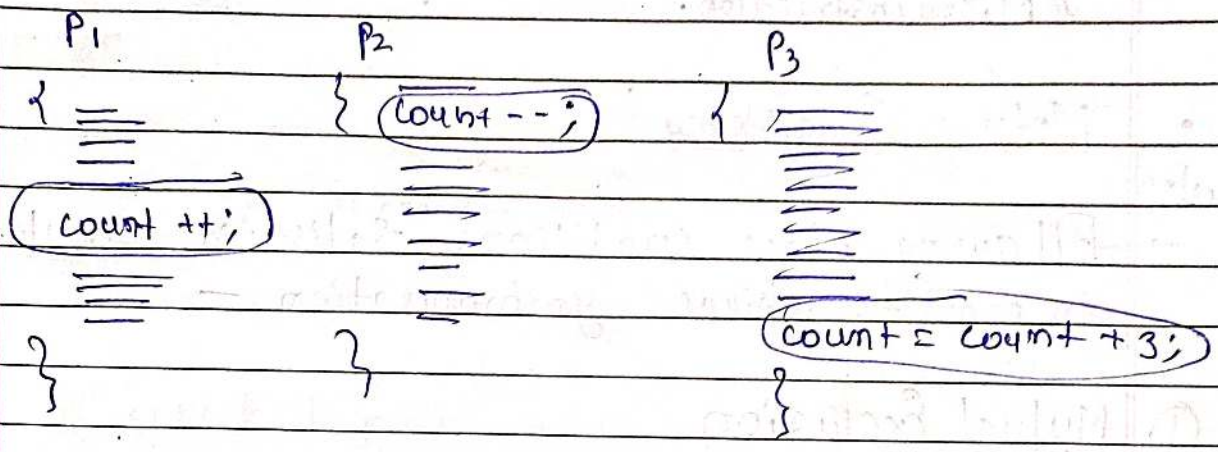
→ Any program can be divided in four sections – and critical is the most important one.

Date / /
Page

RANKA

- **Critical section -**

→ The portion of program text, where the shared variables are kept is known as Critical section.



- **Remainder section -**

→ The portion of program text, where independent variables are kept is called Remainder section.

✡ • **Mutual Exclusion -**

→ The process should Enter inside the critical section in a mutually Exclusive manner. That means No two process should be present simultaneously in the critical section.

✡ **Progress -**

→ The processes, those who are interested to enter inside its critical section, must be present in their respective Entry section. Among those processes, one of the process must be allowed to enter inside the critical section immediately, if the critical section is free.

Progress - leads to Deadlock

Date / /
Page

→ The decision that who will Enter inside the critical section should be taken by those Processes, who are Present in their respective Entry section and this decision should not be postponed indefinitely.

→ Not interested processes should not block the processes who are interested to enter inside the critical section.

✡ Bounded waiting →

→ There must Exist a bound on the no. of times that other processes are allowed Enter inside their Critical Section after a process has made a request to Enter inside its critical section and before its request is granted.

e.g -

P_i — flag = 1 → interested

```
while (1)
{ while (turn == j);
      C.S
  turn = j;
}
```

P_j — flag = 0

```
while (1)
{ while (turn == i);
      C.S
  turn = i;
}
```

turn = i

turn = i → Bound. W. = ✗

turn = j → Remainder → Non-interested

M.E = ✓
Progress: ✗
B.W = ✓

$P_i = 3$
$P_j = 1$

⇒ Progress is not satisfied

Above given -

⇒ This algorithm is known as "Strict Alternation Algorithm" / Backer's Algorithm.

- e.g.

Pi

boolean flag[2];

f	f
0	1

Pj

```
while (1)
{ flag[i] = true;
while (flag[j] == true);
        C.S
flag[i] = flase;
}
```

```
while (1)
{ flag[j] = true;
while (flag[i] == true);
        C.S1
flag[j] = false;
}
```

M.E = ✓

P = ✗ ⇒ | T | T |

B.W = ✓

↳ Deadlocks

i j — TT

Date / /
Page

RANKA

e.g. - bool flag [2] ;
bool turn ;

P_i

```
while (1)
{ flg[i] = true;
  turn = j;
while (flag[j] == true && turn == j);
```

C.S.

```
flag[i] = false;
}
```

P_j

```
while (1)
{ flag[j] = true;
  turn = i;
while ( flg[i] == true
        &&
        turn == i);
```

C.S.

```
flag[j] = false;
}
```

M.E = ✓
P = ✓
B.W = ~~✗~~ ✓

} all satisfied

turn = i ~~j~~ i

∅	∅
i	j

⇒ This algorithm is known as "peterson's Algorithm"
→ (peterson's correct solution).

1 0 1 0

Date / /
Page

RANKA

Q.

Ans

P1

while (S1 == S2);

CS

S1 = S2

P2

while (S1 != S2);

CS

S2 = ! S1

S2 = S2

S1 S2 S.

S2 S2

S2

S1	S2	
T	T	→ P2
T	F	→ P1
F	T	→ P1
F	F	→ P2

22/09/22

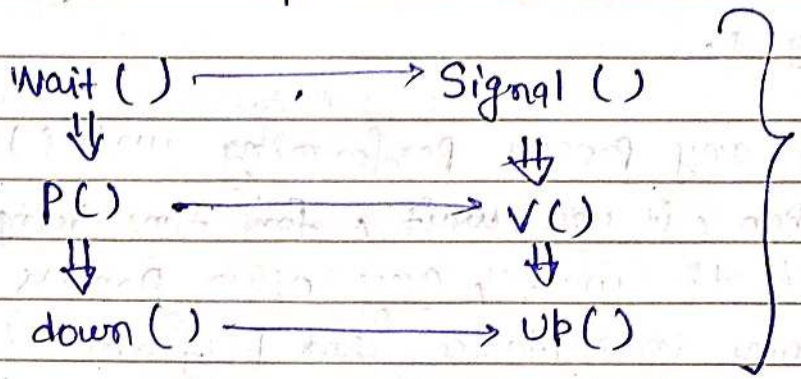
Date / /
Page

RANKA

☆ Semaphore - (User-defined Data-type)

→ Semaphore is an integer variable that apart from initialization is accessed through atomic operations, i.e. wait() and signal().

Atomic - indivisible - Preemption not possible.



⇒ wait (S);

S → Semaphore

```
{ while (S ≤ 0);
  } S--;
```

⇒ Signal (S);

```
{ S++;
  }
```

⇒ if the value of Semaphore is less than or equal to zero, the process which is performing wait() operation ~~may~~ will wait until the value of Semaphore get changed to greater than zero.

→ Value will be changed by - either, signal operation is performed on the same semaphore, by ~~the~~ any other process. ~~is performing Signal op. on the same Semaphore~~. Or, the value gets changed to positive value.

Busy waiting

Date / /
Page

→ If any process is in wait condition, it may wait, but Signal & operation will never wait.

→ If any process performing wait() on any semaphore, the value of the semaphore will be decremented by 1. and, if any process performing signal() on any semaphore, the value of the semaphore will be incremented by 1.

→ If any process performing wait() on any semaphore then, it will wait, for this busy, for getting of the wait(), any other process, must have to come in either one wait or signal() on the same semaphore.

- Semaphore →

→ Binary Semaphore

{ 0, 1 }

only

Counting Semaphore

(−∞ to ∞)

any integer

- Define - Semaphore -

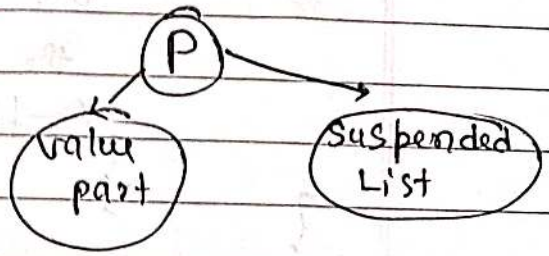
typedef short Semaphore
{ int value;
short Process *L;
};

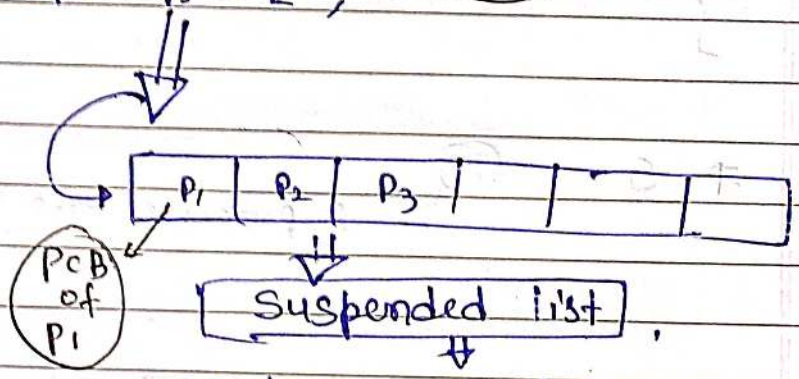
Date / /
Page

→ Every instance of semaphore will have 2 - part, one integer - value and one - pointer.

→ int value, will contain the value. and Pointer part will be pointing to a suspended list.

```
Semaphore P, Q, R;
typedef struct semaphore
{ int value;
  struct process *L;
}
```





→ to store the info of suspended processes.

→ Every semaphore has their own Suspended list.

• Counting Semaphore →

```
Wait (Semaphore S)
{ S.value = S.value - 1;
  if (S.value < 0)
  { suspend() the process and place its
    PCB in the suspended list.
  }
}
```

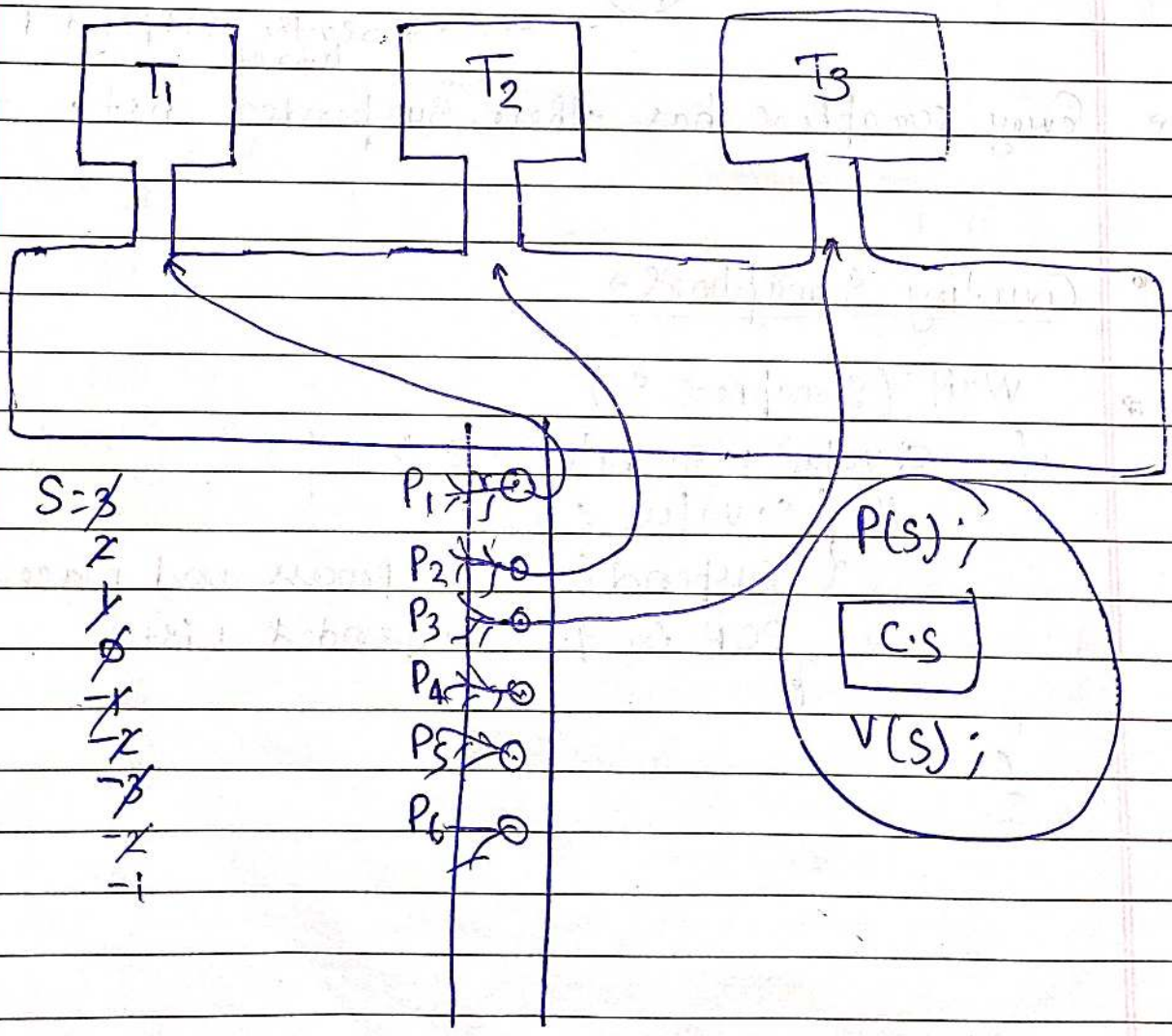
Date / /
Page

RANKA

* Signal (Semaphor S)
{
S.value = S.value + 1;
if (S.value ≤ 0)
{ wakeup() one of the processes from the Suspended list so, that it can Continue its further Execution
}
}

* S = -2 → 2 processes are in suspended list.

24/09/22



S=0 ①

Date / /
Page

- **Binary Semaphore** (S = 1/0)

1 → +ve
0 → -ve

(2n)

```
void wait (Semaphore S)
    {
        if (S.value == 1)
            S.value = 0;
        else
        { Suspend () the
            Process and place its PCB in the
            Suspend list.
        }
    }

void signal (Semaphore S)
    {
        if (Suspended list is empty)
            S.value = 1;
        else
        { wakeup() one of the process from the
            suspend list
        }
    }
```

Note :- if (S = 0), we can't predict how many processes are suspended or not, we have to see the Suspended list for ~~the~~ getting the no. how many processes are Suspended)

Date / /
Page

RANKA

⇒ S = 0

P(S) ⇒ (S = 0) ⇒ Suspended.

⇒ S = 1

P(S) ⇒ (S = 0) ⇒ gets out of program

⇒ S = 1

V(S) ⇒ (S = 1) ⇒ Suspended list. empty

↳ (S = 1)

S = 0

V(S) ⇒ (S = 1) ⇒ Suspended list empty

(S = 1)

⇒ S = 0

V(S) ⇒ S = 0 ⇒ Suspended list is not Empty

wake up()

(S = 0)

⇒ S = 1

V(S) ⇒ S = 1 ⇒ Suspended list is not Empty

wake up()

(S = 1)

⇒ Binary Semaphore provides Mutual Exclusion

Date / /
Page

Q. (2.6) P_i, $i = 1 \ldots 9$. (mutex) – Binary

mutex = 1

repeat

P(mutex) / V(S)

{ Critical Section }

V(mutex)

forever.

P_{10} = identical → V(mutex) – in place P mutex)

largest no. of processes

$i = 1$ (2) ✓ .

(0) $P_{(i)}$ $V_{(i)}$ (2)

(1)

P_1	P_2	P_3 – – – – – – – –	P_9	P_{10}
while(1)	while(1)	while(1)	while(1)	while(1)
{ P(mutex)	{ P(mutex)	{ P(mutex)	{ P(mutex)	{ V(mutex)
C.S	C.S	C.S	C.S	C.S
V(mutex)	V(mutex)	V(mutex)	V(mutex)	V(mutex)
}	}	}	}	}

mutex = ~~1~~ 0

C.S | P_1 | P_2 | P_{10} | P_3 | P_4 | P_{10} | P_5 | P_6 | P_{10} | P_7 | P_8 | P_{10} | P_9 |

S.L (Suspend List) | P_2 | P_3 | P_4 | P_5 | P_6 | P_7 | P_8 | P_9 |

P_1 & P_{10} → active

=> 10 Maximum processes at any time.

Date / /
Page

RANKA

P Q S T

```
while(1)        while(1)
{ w;            { y:
print'0';       Print '1';
print'0';       Print '1';
x:              z:
}               }
```

Date / /
Page

- Producer Consumer Problem with the help of Semaphore -

```
Semaphore Mutex = 1;
Semaphore Empty = N;
Semaphore Full = 0;

Void Producer()
{ int itemp;
  itemp = Producer();
  wait (Empty);
  wait (mutex);
  Buffer [in] = itemp;
  in = (in+1) mod N;
  Signal (mutex);
  Signal (Full);
}

Void Consumer()
{ int itemc;
  wait (Full);
  wait (Mutex);
  itemc = Buffer[out];
  out = (out+1) mod N;
  Signal (Mutex);
  Signal (Empty);
  consume (itemc);
}
```

⇒ Mutex is a binary / semaphore that will be used by the Producer & Consumer to access the Buffer in a Mutually Exclusive manner.

⇒ Empty is a counting semaphore that will keep track of the no. of Empty slots inside the Buffer at any Point of time.

⇒ Full is a counting Semaphore that will keep track of the no. of filled slots inside the Buffer at any point of time.

Hw Scheduling qn Synchronisation qn

Date / /
Page

	0	g	← in, out	0	9	← in, out
first	1	6	← in ← out	1	6	← out
consumer	2			2	c	
CPU	3			3	d	
	4			4	e	
	5			5	f	
	6			6	g	
	7			7	h	

Mutex = 1 0 1 0 1

Empty = 8 7 8

full = 0 1 0

rmutex = 1 0 1

Empty = ~~8 7~~ 0 1 0

full = 8

→ Semaphore is a synchronisation tool.

Q. what will happen, if we interchange wait(empty) & wait(mutex) in producer's code

(1) Some of the items produced by the Producer will be lost
(2) No problem the code still works fine
(3) Both the Consumer will access the Buffer simultaneously
(4) There ~~will be~~ may be possiblity of Deadlock.

mutex = 1 0

empty = 8 0 1 2 3

full = 8 7

Q. what will happen, if we interchange signal(mutex) & signal(full), in producer's code.

~~skip~~ mutex = 1 0 1

Empty = 8 7

full = 0 1

2.35 P(S): S = S - 1

if S < 0 then wait

V(S): S = S + 1

if S ≤ 0 then wakeup a proc.

P_b, V_b - wait and signal - Binary

X_b, Y_b - Binary - P(S), V(S)

P(S): $P_b(X_b)$;
S = S - 1;
if (S < 0)
{ $V_b(X_b)$;
$P_b(Y_b)$;
}
else
$V_b(X_b)$;

V(S): $P_b(X_b)$;
S = S + 1;
if (S ≤ 0);
$V_b(Y_b)$;
$V_b(X_b)$;

$X_b = 1$

$Y_b = 0$

S = 0 = -1

This portion is necessary, because if it is not there, this process will not be waked up again. due to wait operation in V(S). → no other process can wake-up it.

→ No two ~~signal~~ process can perform wait and signal simultaneously.

Date / /
Page

RANKA

⑧ Reader - writer problem →

→

```
mutex = 1 ;
wrt = 1 ;

Void Writer()
{ wait(wrt);

  C.S

  Signal(wrt);
}
```

(Reader favourable Code)

```
int Readcount = 0;
Void Reader()
{ wait(mutex);
  Readcount ++;
  if (Read count == 1)
    wait(wrt);
  Signal(mutex);

  C.S

  wait(mutex);
  Readcount --;
  if (Readcount == 0)
    Signal(wrt);
  Signal(mutex);
}
```

HW ⇒ write a writer favourable code ?

Q. (2.10) P - C Problem

Shared buffer size = N

empty, full, mutex

0 N 1

P, Q, R, S — either empty or full

wait()

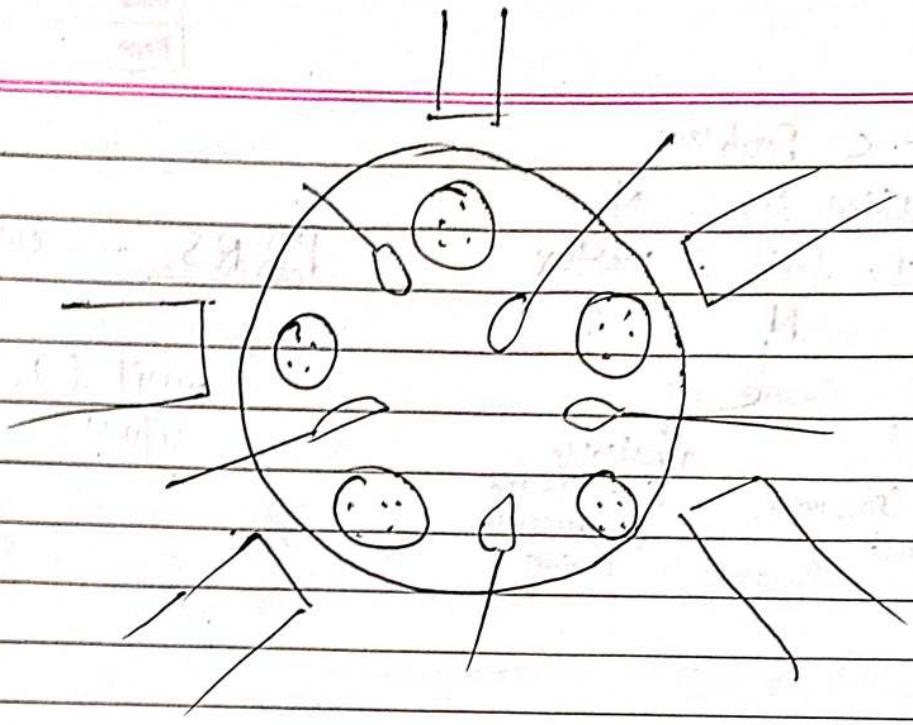
signal.

empty → available slots for the consumer to read

full → available slots for the producer to write

P N-1 Q.

Q.



5 - forks
5 - philosopher

} Dining philosopher problem

fork - resource

1st - right fork
2nd - left fork

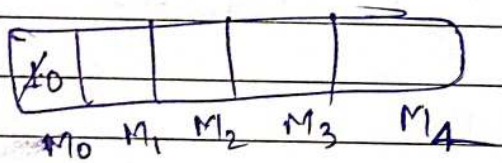
} ⇒ Deadlock

- P(Left) ←
  P(right)

[C-S]

V( )
V( )

int M[5] - M - semaphore



Date / /
Page

- **Concurrent Programming →**

S_1 : $a = b * c$;
S_2 : $d = e / f$;
S_3 : $g = a / d$;
S_4 : $h = g * i$;

writeset = { a, d, g, h }

readset = { a, b, c, d, e, f, g, i }

→ we can execute S_i and S_j statements parallely or concurrently if and only if following 3 condⁿ are satisfied simultaneously.

(i) $R(S_i) \cap W(S_j) = \phi$

(ii) $R(S_j) \cap W(S_i) = \phi$

(iii) $W(S_i) \cap W(S_j) = \phi$

⇒ $\{b, c) \cap \{d\} = \phi$

$\{e, f) \cap \{\} .$

S_1 S_2 S_3

S_2 ↓

```
Par begin
S1;
S2;
S3;
Parend
```

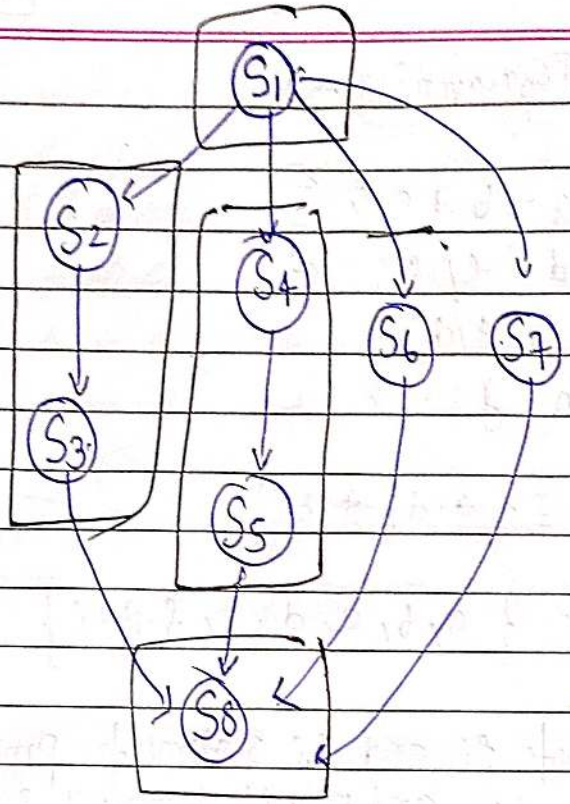
```
begin
S1;
S2;
S3;
end
```

← S_1 ↓ S_2 ↓ S_3

Par begin → parallel beginning

Date / /
Page

Q.



A.

~~begin~~

~~S1~~	~~S1~~
~~S2~~	~~S6~~
~~S3~~	~~S4~~
~~S4~~	~~S6~~
~~S5~~	~~S7~~
~~S6~~	~~S3~~
~~S7~~	~~S5~~
~~S8~~	~~S8~~

```
begin
  S1;
  Parbegin
    begin
      S2;
      S3;
    end
    begin
      S4;
      S5;
    end;
    S6;
    S7;
  Parend
  S8;
end;
```

Date / /
Page

Q.
```
int count = 0;

void dolly()
{ int i;
  for(i=0; i<5; i++)
  } count = count + 1;

int main()
{ Parbegin
    dolly(); —
    dolly(); —
  Parend
  return 0;
}
```

count = count + 1 is executed as:

(a) Load R, M[count];
(b) R ← R + 1;
(c) Store M[count], R;

→ what could be the minimum and maximum value of count after the execution of the above concurrent Program -

(a) 1, 10 (b) 2, 10 (c) 3, 10 (d) 4, 10
(e) 5, 10

P_1		P_2	
(1) i = 0	preempt R	(1) i = 0 to 3	R
a, b — = a	1	a, b, c =	(4) — (a) (6)
(2) i = 1 to		(2) Store — (4)	except
9, b —			

1st Dolly	R_1	2nd Dolly	R_2
i=0 a (b) (c)	1	i=0 a b c	1
i=1 a b c	2	i=1 a b c	2
i=2 a b c	3	i=2 a b c	3
i=3 a b c	4	i=3 a b c	(4)
i=4 a b c	5	i=4 a b (c)	2

Count = 0, 1, 2, 3, (4), 1, 2, 3, 4, 5, (2)

import process - header file

Date 21/11/22
Page

fork() system call →

→ fork() is a system call, with the help of which, a new process is created.
→ It returns, positive integer (PID of the child process) to the parent process.
→ It returns 0 to the child process.
→ It returns -ve integer if, child, creation is unsuccessful.

e.g. →

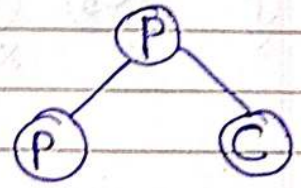
fork implementation -

```
fork() int main()
{ int pid;
  pid = fork(); child creation
  if (pid > 0)
  { printf ("I am parent process");
  }
  else if (pid == 0)
  { printf ("I am a child process");
  }
  else
  { printf ("child creation is unsuccessful");
  }
  return 0;
}
```

output -

(2)(1) → I am parent process
(1)(2) → I am child process

vice versa as per CPU allocation

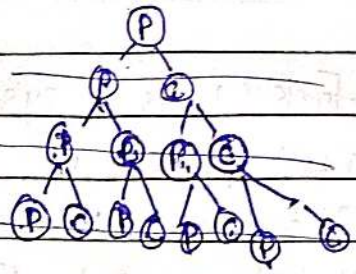


→ Both the parent and the child will have the same code.
→ The child and the parent will have different physical addresses, but they will have same virtual addresses.

Date / /
Page

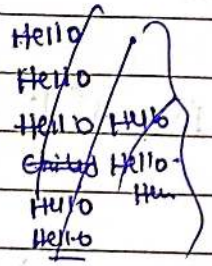
RANKA

Q.
```
int main()
{
    fork();
    printf("Hello");
    fork();
    printf("Hello");
    fork();
    printf("Hello");
    return 0;
}
```

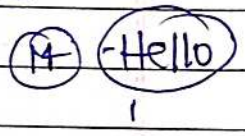


7 - child
7 - Parent

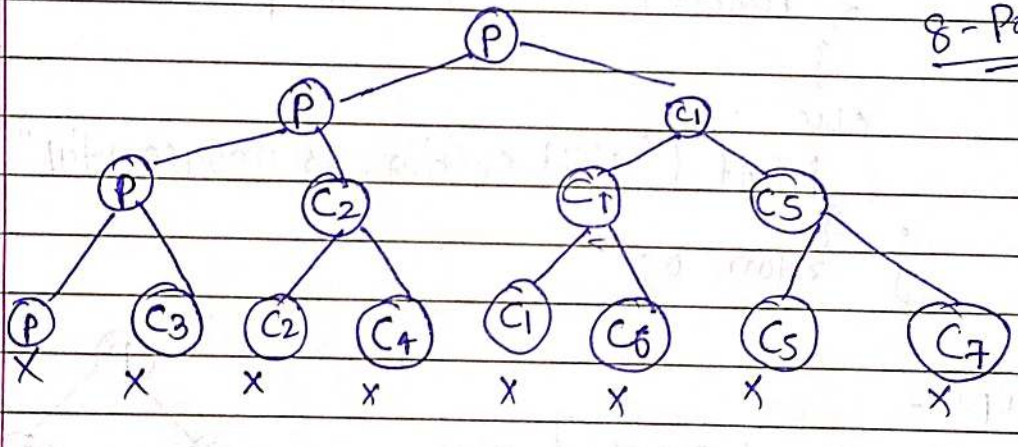
7 - child }
7 - Parent



$\frac{1+1+1+1+1+1+1}{7}$



7 - child
8 - Process



if - fork call is in serial order -

Total no. of child process = $2^n - 1$

n = no. of fork calls

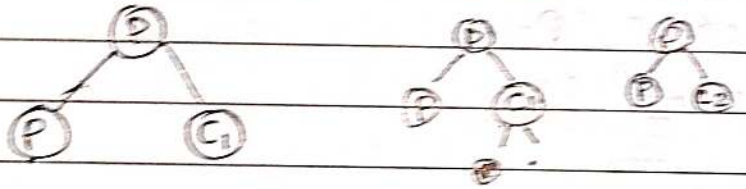
& Total no. of processes = 2^n

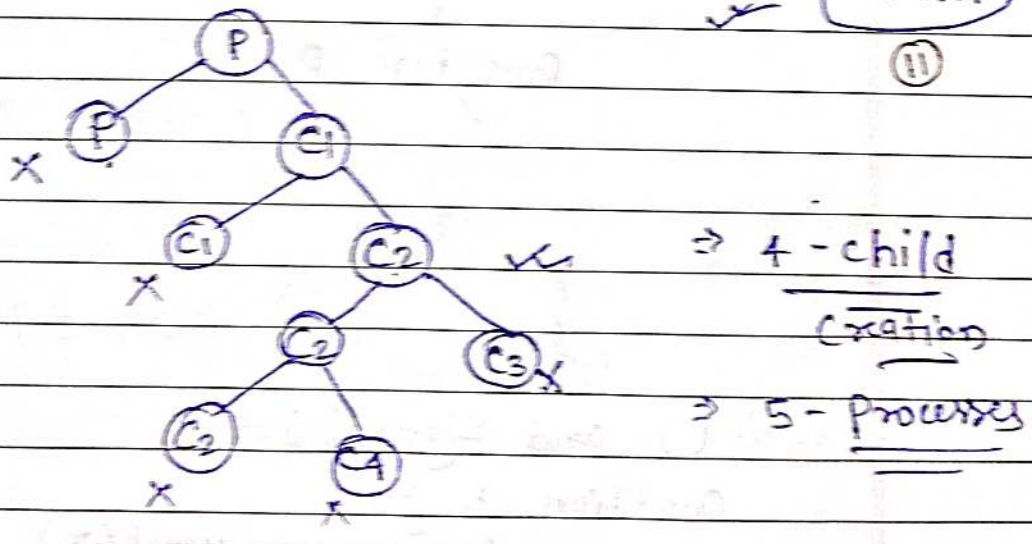
$2^3 - 1 = 7$ - child

$2^3 = 8$ processes - Total

Date / /
Page

Q. int main ()
{ printf ("Hello");
if (fork () || fork ())
printf ("Hello");
else if (fork () && fork ())
printf ("Hello");
else
printf ("Hello");
printf ("Hello");
return 0;
}





⇒ 4 - child creation

⇒ 5 - Processes

⇒ Note - All the fork are successful ✓

Monitor -

→ Monitor is a programming language that compiler supports to achieve Synchronisation.

→ It is collection of variables, condition variables & Procedures combined together in a special kind of Package known as Monitor.

→ The most important property of the Monitor is that only one process will be active inside the Monitor at any point of time.

monitor Example

Keyword

```
{
    Variables;
    Condition variables;
    Procedure P1
    {
    }
    Procedure P2
    {
    }
}
```

wait() and signal()

Condition x, y;

x.wait() or, wait(x)

y.wait() wait(y)

x.signal()

y.signal()

wait() – Condition varible –

↳ Suspend the ~~variable~~ process and put it in Block Queue of condition variable

Signal() – Condition variable –

↳ if the Block queue is Empty, signal will do nothing It will be lost.

↳ and if the Block queue is not Empty, signal() will wake up or resume any one of the process.

⇨ monitor Producer-Consumer.

```
{ Integer count;
  Condition Full, Empty;
  Procedure Enter_Item
  { if (count == N) wait (Full);
    Enter the item;
    Count = count + 1;
    if (count == 1) Signal (Empty);
  }

  Procedure Remove_Item
  { if (count == 0) wait (Empty);
    Remove the item;
    count = count - 1;
    if (count == N-1) Signal (Full);
    ~~Signal (Full);~~
  }
}
```

Date / /
Page

~~Procedure~~

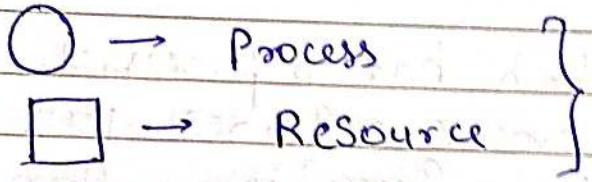
```
Procedure producer.
{
    while (true)
    { produce = item(itemp);
    producer : Consumer . Enter_Item";
    }
}

Procedure Consumer
{   while (true)
    {   Producer : Consumer . Remove_Item;
        Process_item (item());
    }
}
```

Date / /
Page

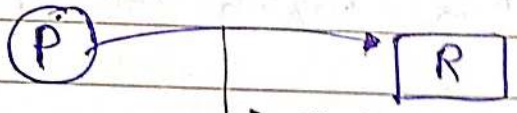
Deadlocks →

→ When two or more than two Processes are waiting for an event to happen, but that event never happens then, we say that these processes are involved in a Deadlocks.

- Conventions -



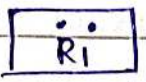
RAG



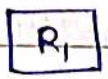
Requesting edge

Allocating edge

→ 3 instances of R_1 (Resource) is available currently to the system.



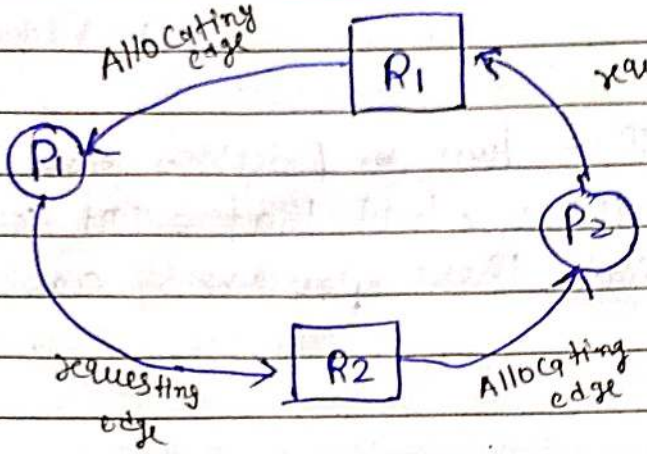
→ 2 instances of R_1 is available.



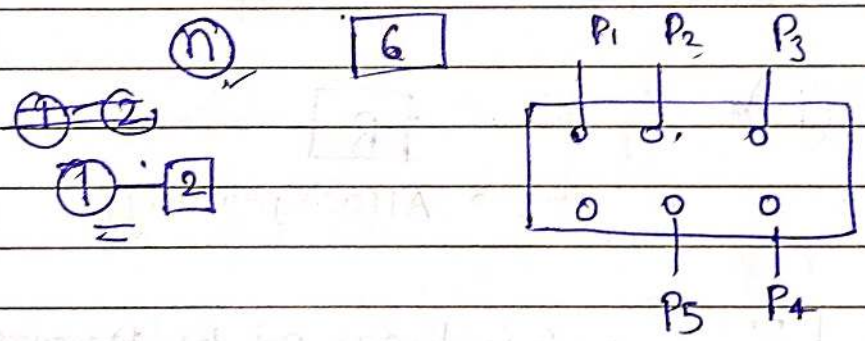
→ only one instance of Resource is available to the System.

- Requesting edge → Process is requesting to the resource and the Resource has not been Allocated.

- Allocating edge → Process has requested to the resource and the Resource has been Allocated.



Q. Consider a system which has n processes and six tape rights. each process requires two tape rights to complete their Execution, then what is the maximum value of n which ensure, deadlock free operation.



⇒ 5.

Q. Consider a system, which has three processes and each Process requires, two resources, to complete their execution what is the minimum no. of resource required ~~for~~ to ensure Deadlock free operation.

3-P.

P1 - (2) 2-1 - 1
P2 - (2) - 1
P3 - (2) - 1

(1) extra

Minimum = (4)

Q. Consider a System, which has three processes, P_1, P_2, & P_3. The peak demand of ~~each~~ process, P_1, P_2 & P_3 are, 5, 9, 13 respectively. What is the minimum no. of resources required to ensure deadlock free operation

P_1	P_2	P_3	
5	9	13	
4	8	12	+ 1

(25)

Q. (37) P_1 ---- P_n share m identical resource. reserved and released ~~at~~ one at a time.

Max = $P_i = S_p$ where $S_i > 0$.

$P_i max = S_p$

P.N. P_1, P_2, P_3 ---------- P_n.

Max^m demand S_1 S_2 S_3 ... S_n.

$m > (S_1-1)+(S_2-1)+(S_3-1)+ \ldots +(S_n-1)$

$m > (S_1+S_2+S_3+S_4 \ldots +S_n) - n$

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

(m+n) → min no. of resources

n → Max. no. of

$\sum_{i=1}^{n} S_i$ → Max^m demand

Deadlock — Resource oriented
Synchronization — variable oriented

Date / /
Page

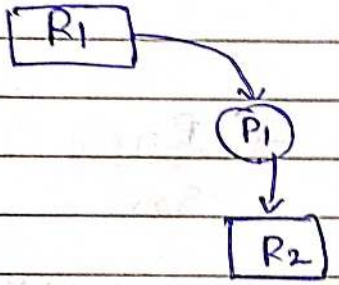
Deadlock characteristics :-

1. Mutual Exclusion
2. Hold & wait
3. Non-preemption
4. Circular-weight.

Mutual Exclusion

→ one resource at a time.

Hold & wait-



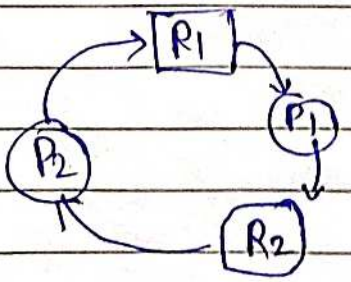
→ P_1 is holding the resource R_1 & requested for the Resource R_2 to be allocated.

Non-premption

→ No Premption.

Circular weight -

→ It doesn't guarantee about Deadlock.



→ Each of the above characteristics is a necessary condition for a Deadlock to occur. but not a Sufficient condition ~~to occur~~. for a deadlocks to occur.

→ Deadlocks है तो → Mutual Exclusion होगा ही. लेकिन कहीं Mutual Exclusion है तो Deadlock होगा है ये कहना ❌ गलत है।

→ The necessary and sufficient conditions for a deadlocks to occur is that the following four situations should be satisfied Simultaneously.
(a) ME (b) H & w (c) Non-Prem. (d) C.W.

→ चारों का एक साथ satisfy होना - Deadlock की गारंटी है। अगर कोई एक भी नहीं है तो हो भी सकता है, नहीं भी।

(1) Deadlocks prevention
(2) Deadlocks Avoidance
(3) Deadlocks Detection
(4) Deadlocks Recovery

(1) Deadlocks prevention ⇒

Disatisfy - any one of 4 characteristics

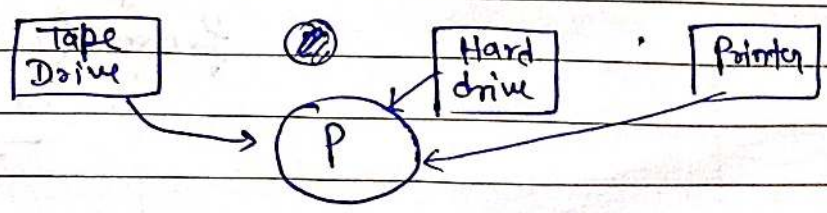
(a) M.E → It can't be dissatisfied

Resources → Non-Sharable to Sharable - then dissatisfied
→ Non-Sharable - Strictly then can't be dissatisfied.

(b) Hold & wait.

2- scenarios to disatisfy

(1)



In this scenario, we will provide all the Resources required by the process at the time of Process Execution. So, there will be no Hold & wait.

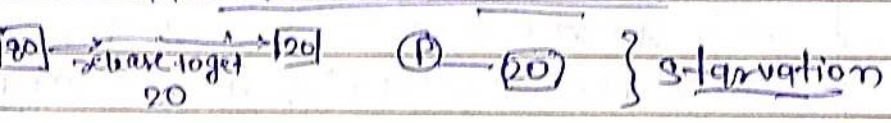
Disadvantage

=> Poor Resource Utilization
(Low)

- Scenario - 2

-> In this scenario, ~~we will~~ if any process has requested for the resource, then it has to release all the resources it has been allocated, so, there will be no Hold & wait.

Disadvantage - Suffer from Starvation



(c) Non-preemption ->

Suppose, P_1 has made a request for the Resource R_1

R_1 is available
↓
allocate it to P_1 immediately

R_1 is not available
↓
R_1 has been acquired by some other process say P_2.

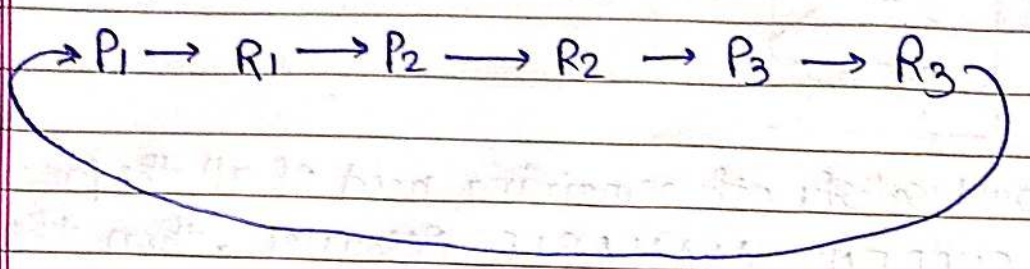
P_2 is in Execution
↓
Make P_1 to wait

P_2 is waiting for some other resource say R_2
↓
Preempt Resource R_1 from P_2 and allocate it to P_1

Date / /
Page

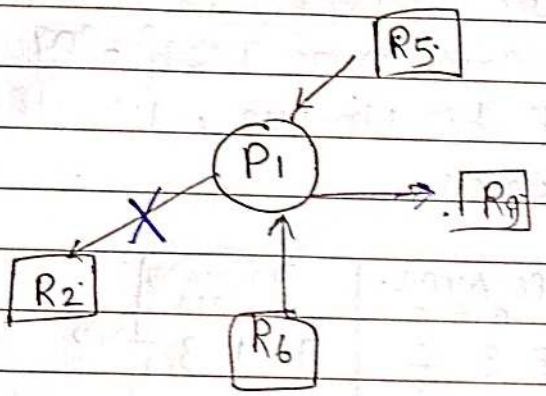
RANKA

(d) Circular wait -



→ Number all the Resources in System.

→ Process can make a request for the resources only in the increasing order of Enumeration. — Restriction



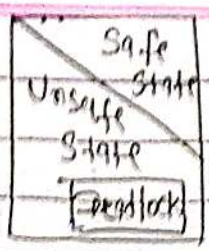
✡ Deadlock Avoidance - (Banker's Algorithm)

	Max NEED			Current ALLOCation			CURRENT AVAILABLE			Remaining Need
	A	B	C	A	B	C	A	B	C	
P0	7	5	3	0	1	0	3	3	2	743
P1	3	2	2	2	0	0				122
P2	9	0	2	3	0	2				600
P3	2	2	2	2	1	1				011
P4	4	3	3	0	0	2				431
				(7)	(2)	(5)				

Date / /
Page

RANKA

Safe / Unsafe State



→ If we could satisfy the remaining need of all the processes with the CURRENT AVAILABLE Resources, then the System is said to be in the Safe state, otherwise the System will be in the UnSafe state.

→ If the System is in UnSafe state, then there is a possibility of Deadlock.

→ The order in which we satisfy the remaining need of all the processes, is said to be a Safe-Sequence.

→ Safe Sequence may not be Unique, i.e. there could be multiple Safe Sequences.

	MAX NEED A B C	CURR. ALLOC. A B C	CURR. AVAL. A B C	Remaining Need
P_0	7 5 3	0 1 0	3 3 2	7 4 3 → P_0
			− 1 2 2	.
			CA = 2 1 0	
			after completion of exp of P_1 =	
			2 1 0	
			+ 3 2 2	
			CA = 5 3 2	
			− 0 1 1	
P_1	3 2 2	2 0 0	CA 5 2 1	1 2 2 → P_1
			+ 1 2 2	6 0 0 → P_2
			CA 7 4 3	0 1 1 → P_3
			CA 7 4 3	4 3 1 → P_4
			= 0 0 0	
			+ 7 5 3	
			CA = 7 5 3	
			− 6 0 0	
			CA = 1 5 3	
			+ 9 0 2	
			CA = 1055	
			− 1 3 1	
			6 2 4	
			+ 1 3 3	
			CA 1057	

753 3 2 2 ~~4 2 4~~ 4 2 1

0 1 0

1 2 2

Date / /

Page

RANKA

⇒ P0, request for 3 instance of A, 2 instance of B and 2 instance of C.

→ Invalid request

Q (3.10)

	X	Y	Z	X	Y	Z
P0	1	2	1	1	0	3
P1	2	0	1	0	1	2
P2	2	2	1	1	2	0

5 5 5

	alloc		request	Aval	~~Need~~
	X Y Z		X Y Z	~~5 5 5~~	
P0	1 2 1	~~4 3 4~~	1 0 3	~~5 5 5~~ 0 1 2	~~4 3 4~~
P1	2 0 1		0 1 2		~~3 5 4~~
P2	2 2 1		1 2 0		~~3 5 4~~
	5 4 3				

Ans- P2 at last

~~+2 1 4 3 4~~ ~~1 0 3~~ ~~2 3 1~~

	alloc		aval
P0	1 2 1	+1 0 3	P1 0 1 2
	2 2 4		0 1 2
P1	2 0 1	+0 1 2	6 0 0 — P1 comple
	2 1 3		2 0 1
P2	2 2 1	+1 2 0	0 1 2
	3 4 1		CA' 2 1 3 — 1
	2 1 4		1 0 3
	2 2 1		1 1 0 — after P0 compute
	1 2 0		
	5 5 5 — Total		

CA' = 1 1 0
1 2 1
1 0 3
3 3 4 —

CA' 1 2 0
2 1 4 — P2 compu

Q. 3.16

	Alloc	Need	Max	AVAL	Req
	X Y Z		X Y Z	X Y Z	X Y Z
P0	0 0 1		8 4 3	3 2 2	0 0 2
				0 0 2	
P1	3 2 0		6 2 0	3 2 0	2 0 0
P2	2 1 1		3 3 3	~~5 3 0~~	
				~~1 2 0~~	

req1

→ Deadlock avoidance is less restrictive than Deadlock Prevention.

→ The Unsafe State purely Depends on the Behaviour of the processes.

→ Each and Every time, when the process, is requesting for any resource, the Banker's algorithm will be applied to identify whether the system is in safe state or Unsafe state.

→ If the System is in the Safe State, then the request of the process will be granted. and if the system is in the Unsafe state, than the request of the process will be denied and the Deadlock will be avoided.

⇨ Deadlock Detection -

(i) Cycle Completion for Single instance
(ii) Banker's Algorithm for Multiple instance.

Memory Manggement

$\log_2 N$

Date / /
Page

RANKA

2^4

⇒ RAM or Physical memory
or, Main memory

Goal → Efficient utilization of main memory by minimizing Internal and External fragmentation.

Memory can be represented in two ways:

Byte Addressable

→ No. of bits in one byte is fixed

Word Addressable

→ No. of bits in one word is not fixed; it varies from machine to machine.

1 w = 16 bits = or, 2 B
or, 1 w = 32 bits = or, 4 B
or, 1 w = 64 bits = or, 8 B

16w → 2^4 w
4-B.

16 × 4 B
= 64 B

→ Convert into basic units Bytes, & words

$\lceil \log_2 N \rceil$

1 K = 2^{10}
1 M = 2^{20}
1 G = 2^{30}
1 T = 2^{40}

512 K w
$2^9 \times 2^{10}$ w
2^{19} w

$\lceil \log_2 2^{19} \rceil$
19 bits

Date / /
Page

Q. Consider a system which has 256 M-words and each word is having the size of 64 bits. What is the capacity of main memory in Bytes.

256 M w
$2^8 \times 2^{20}$ words
2^{28} words

1 word = 64 bits

$2^{28} \times 64$ bits
$2^{28} \times 2^6$
$\Rightarrow 2^{34}$ bits

1 B = 8 bits
64 bits
8 B

$2^{28} \times 8$ B
$2^{28} \times 2^3$ B
$= 2^{31}$ B
$= 2 \times 2^{30}$ B
$\Rightarrow$ 2 GB

Q Consider a system which has 64 K words, and each word is having the size of 16 Bytes, what is the capacity of main mem- in Bytes

64 K w
$2^6 \times 2^{10}$ w
2^{16} w
$2^{16} \times 16$ B
$2^{16} \times 2^4$ B
$= 2^{20}$ B
$\Rightarrow$ 1 MB

1 w = 16 B

Q. Consider a system, where 32 binary bits are used to represents all the words of M.M., each word is having the size of 32 bits what is capacity of M.M. in Bytes.

32 bits W
$2^5 \times 2^5 = 2^{10}$
4 B x

1 w = 32 bits
4 B

1 B = 8 bit

No. of words = 2^{32} w 32 bits - all words

2^{32} words 2^{32} - total words

$\Rightarrow 2^{32} \times 4\,B$

$\Rightarrow 2^{32} \times 2^2\,B$

$\Rightarrow 2^{34}\,B$

$\Rightarrow 2^{30} \times 2^4\,B$

$\Rightarrow$ 16 GB

✡ RAM-Chip Implementation –

Q. Ram chip size = 128 B
organise the memory of capacity 16 KB.

(1) Calculate the RAM chips required ?
(2) organise the memory
(3) Size of Decoder = ?

→ = Size of the Memory required / RAM Chip size

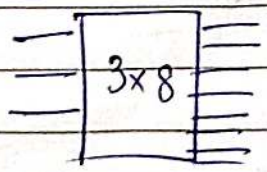
$= \frac{16\,KB}{128\,B} = \frac{16\,K \times B}{128 \times B} = \frac{2^{14} \cdot B}{2^7 \cdot B}$

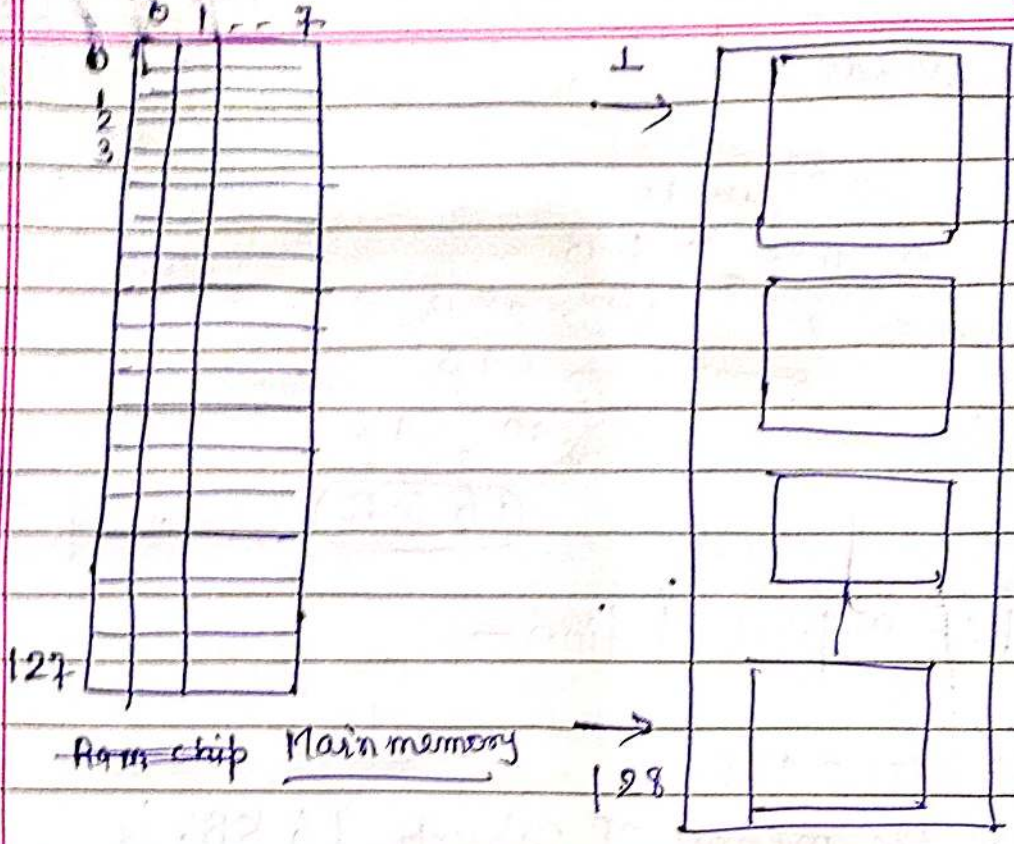
$= 2^7 \times 1$ (2^7 = 128)

Row Column

→ Decoder is used to select a row.

7×2^7
row
$n - 2^n$





Q. Ram chip size = 256 × 1 bit
Organise the memory of capacity 32 MB.

Q. Ram chip size = 512 × 2 bit
Organise the memory of capacity 64 GB

(1) Ram chip size = 256 × 1 bit
32 MB – organize

no. of Ram chips required $= \frac{32\,MB}{256 \times 1\,bit} = \frac{32M \times 1B}{256 \times 1\,bit}$

$= \frac{32M \times 8\,bit}{256 \times 1\,bit}$

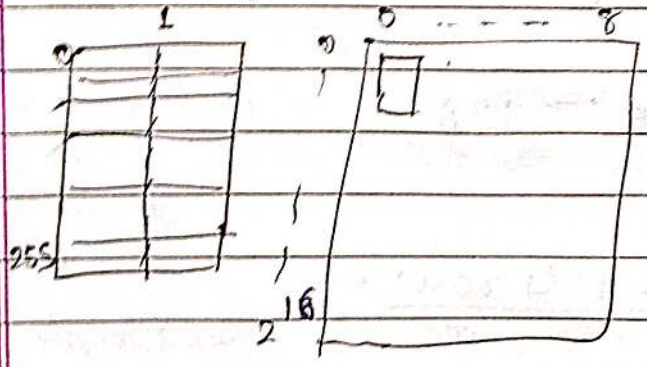
$\Rightarrow \frac{2^5 M \times 8}{2^8 \times 1}$

$\Rightarrow \frac{1}{2^3} M \times 8$

$\Rightarrow 2^{20} \times 2^{-3} \times 8$

$\Rightarrow 2^{17} \times 8$

$\Rightarrow 2^{20} =$



20×2^{20}

Date / /
Page

RANKA

(2) Ram chip size = 512 × 2 bit

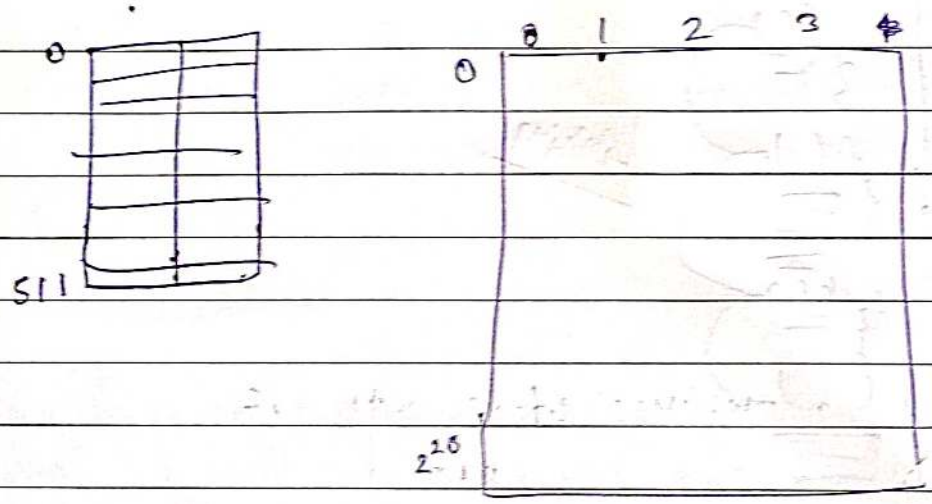
organise = 64 GB

No. of RAM chips required = $\frac{64\text{ GB}}{512 \times 2\text{ bit}} = \frac{64\text{ G} \times 1\text{ B}}{512 \times 2\text{ bit}} = \frac{64\text{ G} \times 8\text{ bit}}{512 \times 2\text{ bit}}$

$= \frac{2^6 \times 2^{30} \times 4}{2^9}$

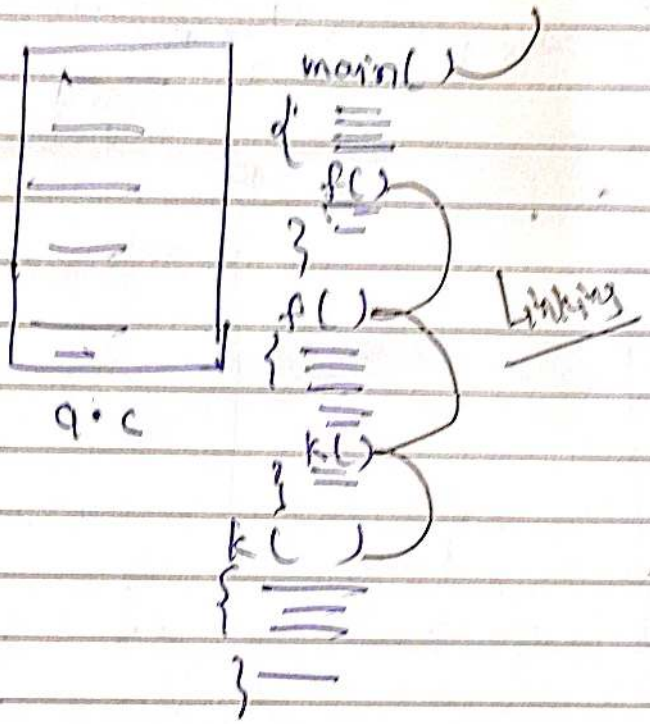
$\Rightarrow 2^{36-9} \times 4 = \frac{2^{27}}{} \times 4$

$= 2^{29} \Rightarrow$ (29 × 2^{29})



Loading, Linking and Address Binding

Loading - Loading of the modules & programmes from Secondary memory to the Main memory.
→ Loading is done by Loader. (part of OS)



Static Loading

→ Program Execution will be faster, due to -
→ copy of all the programmes to the Main memory before its Execution.

Advantage - (Execution time = Less)
→ Execution time ~~= Slow~~ ↳ fast Execution

Disadvantage -
→ not the efficient utilization of the Main memory.

Dynamic Loading

→ Program Execution will be slower due to -
→ Copy of only the necessary part of programmes, to the Main memory, and if required then Access the secondary Memory for requirement.

Advantage →
→ Efficient utilization of Main memory

Disadvantage -
↳ Execution time - ~~slow~~ High
↳ Time taking ~~(~~)

Date / /
Page

RANKA

☆ Linking -

→ Establishing the link b/w all the modules of a programs is known as Linking.

→ Linking is done by the ~~the~~ Linker. (part of OS)

☆ Address Binding : -

→ Association of programmes, Instructions and data to the actual physical memory. is known as Address Binding

→ There are ~~three~~ types of address Binding -

1. Compile time address Binding. — Compiler
2. Load-time address Binding — Loader
3. Execution time address Binding — Processor

→ Reallocation

↳ The address binding will be postponed even after Loading the program into the Main memory.

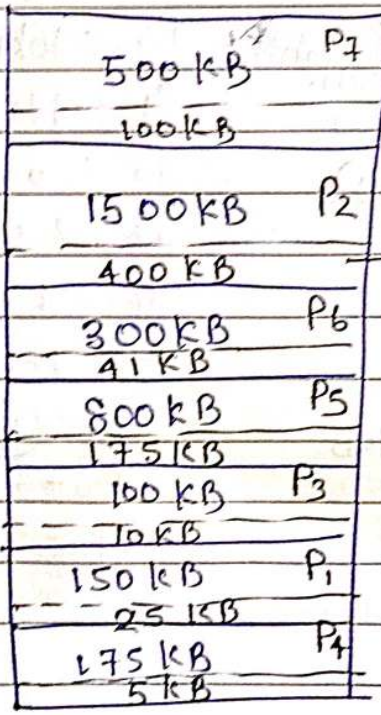
↳ The address binding will be done by the processor at the time of program Execution.

Date / /
Page

Memory management techniques →

- Contiguous Memory Allocation
  - fixed partition scheme
  - variable partition scheme
- Non-Contiguous Memory Alloc[n]
  - Paging
  - Multi-level paging
  - Inverted paging
  - Segmentation
  - Segmented paging

① fixed partition scheme →



- Main memory

→ lower limit register → starting address of that partition
→ higher limit register → ending address of that partition

internal fragmentation

$P_1 \rightarrow 125$ KB
$P_2 \rightarrow 1100$ KB
$P_3 \rightarrow 90$ KB
$P_4 = 170$ KB
$P_5 = 625$ KB
$P_6 = 259$ KB
$P_7 = 400$ KB
$P_8 = 110$ KB

→ No. of partitions is fixed.
→ The size of partition can vary.
→ In one partition, only one program can Execute.
→ Degree of Multiprogramming is restricted.

3525 KB

756

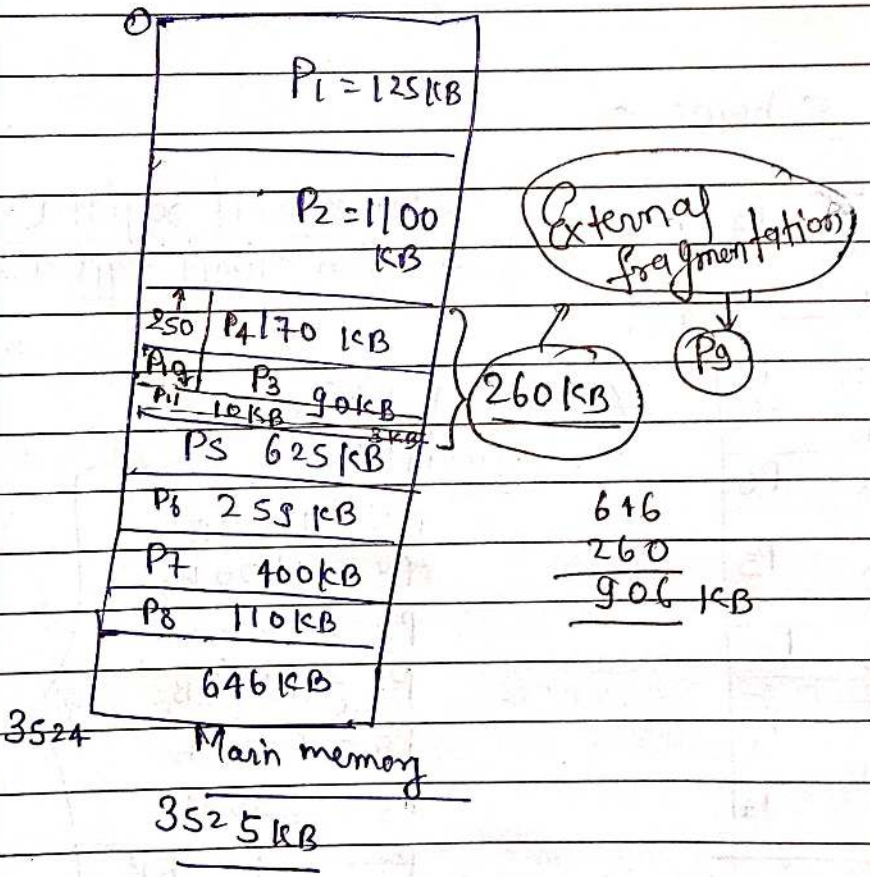
Date / /
Page

RANKA

→ Static loading
→ The size of program Executed is restricted due to size of partition

(2) Variable Partition Scheme :-

→ Initially, Main memory, will be one single contiguous Block of the fixed size of memory.



P1 → 125KB
P2 → 1100KB
P3 = 90kB
P4 = 170kB
P5 = 625KB
P6 = 259 KB
P7 = 400KB
P8 = 110SB
P9 = 700 KB
P10 = 250
P11 = 7KB

→ ~~No~~ Internal fragmentation Can't be ~~possi~~ implemented in variable partition scheme
→ External fragmentation Can't be implemented in fixed partition scheme.

Date / /
Page

RANKA

- Solution to Remove External fragmentation -

① Compaction → By Realloacting the memory.

→ Practically, it is very difficult to implement
→ Running Process gets interrupted here.

② Non-Contigous Memory Allocation -

✱ Partition Allocation methods :-

① first fit
② Best fit
③ worst fit
④ Next fit — circular

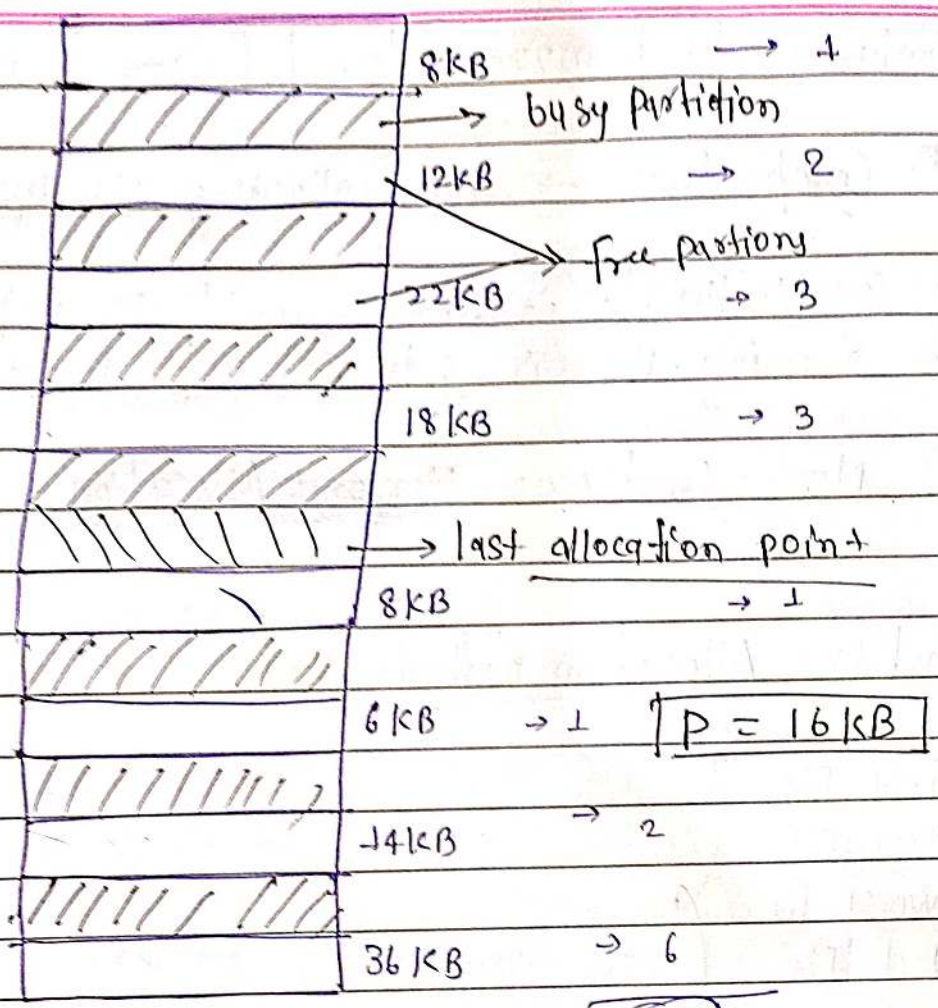
① first fit -
→ Allocate the process, in the first, sufficient partition, from the top of the memory.

② Best fit →
→ Allocate the process in the partition, which is the smallest sufficient partition, from all the available partition.

③ worst fit →
→ Allocate the Process in the partition, which is the largest sufficient Partition, Among all the available / free Partitions.

④ Next fit → Next fit also works like the first fit, but ~~next~~ the searching will start from last allocation point.

Q.



M·M

(19)

first = 22kb
best = 18kb
worst = 36kb
next = 36KB

Q. How many successive request of 6KB could be satisfied assuming variable partition scheme.

P = (6KB) ⟶ (19) request could be satisfied

By default it → Byte addressed
Memory

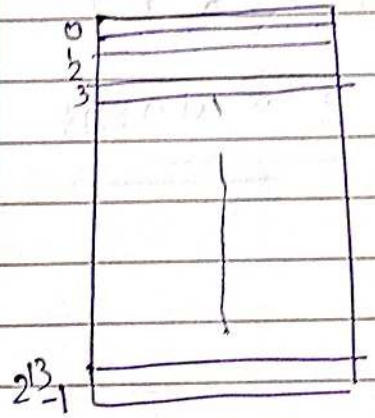
Date / /
Page

✡ Paging →

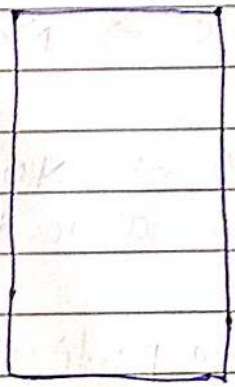
① LAS → Logical address space
② PAS → physical address space
③ LA → Logical address
④ PA → Physical address

VAS → Virtual address space

LAS → always Byte/word
↳ Size
↳ Program Size = LAS



Program = 8KB



PAS = 4KB
M.M

LA → No. of bits required to represent a byte or a word of the Program

LA → Number

Q. → LA = 34 bits
Memory is word addressed
1w = 4B
∴ LAS = 2^{34} w
→ 34 bits required to represent a word of Program.
= $2^{34} \times 4B$
= 2^{36} B
⇒ 64 GB

2^{34} words

Date / /
Page

RANKA

Q LAS = 512 MB

Memory is word addressable

1W = 2B

Calculate LA?

$\Rightarrow = 2^9 \times 2^{20}$ B

$= 2^{29}$ B

$\Rightarrow$ 1W = 2B

$\Rightarrow 2^{29} \times 2^{-1}$

$\Rightarrow 2^{28}$ words

$\Rightarrow$ LA = 28 bits

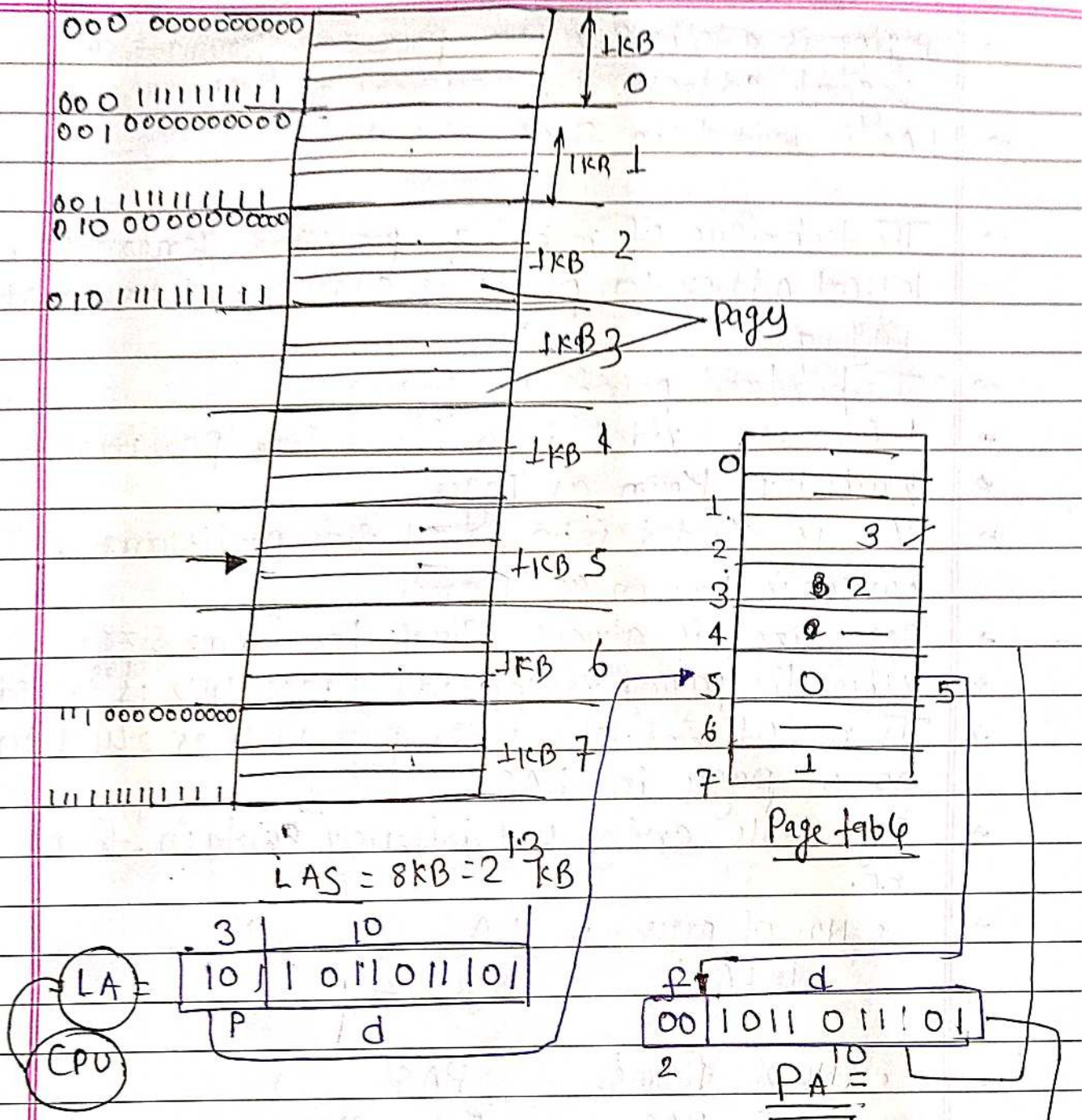
- PAS → Main memory size
- PA → No. of bits required to represent a byte or a word of the Main memory.

→ In a particular system, PA is always fixed, LA can vary.

⇒ Program size vary → LA vary

⇒ In OS, CPU always generates the Logical address

→ In CO " " " " Physical address


000 0000000000
000 1111111111
001 0000000000
001 1111111111
010 0000000000
010 1111111111
111 0000000000
111 1111111111
1KB 0
1KB 1
1KB 2
1KB 3
1KB 4
1KB 5
1KB 6
1KB 7
Page
0
1
2
3
4
5
6
7
3
2
—
0
—
1
5
Page table
LAS = 8KB = 2^13 KB
3
10
10
1011011101
P
d
LA
CPU
P
d
00
1011011101
2
10
PA =
f = no. of bits required to represent a frame of PAS
0
1
2
3
5
7
3
2
1KB
1KB
1KB
1KB
frame
PAS = 4KB = 2^12 B
RAM
P = no. of bits required to represent a page of LAS.
d = no. of bits required to represent a word/byte of a page
Or, no. of bits required to represent a page size or, frame size

Date / /
Page

→ Paging is a technique of processor generated Logical address to physical address.
→ LAS is divided in fixed sized part.

→ The technique of mapping processor generated, logical address to physical address is known as Paging.
→ Important points on paging :-
→ LAS is divided into fixed size partitions ; which is known as "Pages".
→ PAS is divided into fixed size partitions , which is known as "frames"
→ Page size is always equal to frame size.
→ When the paging is applied, a page table will be created
→ The no. of Entries, in the page table is equal to no. of pages in LAS.
→ Page table entry will definitely contain a frame no.
→ • No. of pages in LAS = $\frac{\text{LAS}}{\text{page size}}$

• No. of frames in PAS = $\frac{\text{PAS}}{\text{frame size}}$

• Page Table Size = No. of Pages in LAS * Page Table Entry Size

⇓ to store frame no.
bits required

⇒ no. of Pagetable Entry = no. of pages

→ whenever the process is created, the paging will be applied on the process, and then the page table will be created.

→ The Base address of the page table will be stored in the "PCB".

→ The paging is with respect to, Every process, and Every process, will have its own page table.

→ The page table of the processes will be stored in the Main memory.

Q. Consider a system, which has Logical address of 27 bits and physical address of 21 bits, Page size = 4K word. Calculate no. of pages & no. of frames.

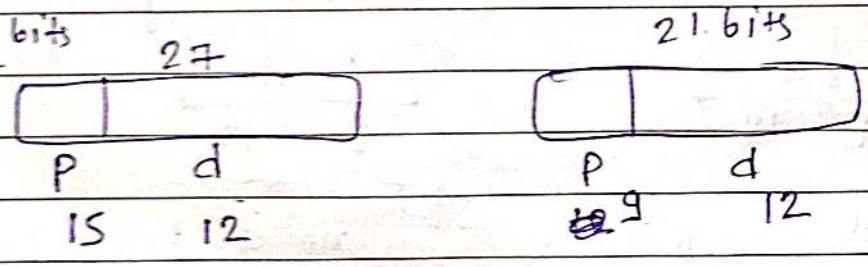
LA = 27 bits = 2^{27} word

PA = 21 bits = 2^{21}

Page size = 4K word.

No. of Pages = $\frac{2^{27} \text{ word}}{4 \times 2^{10} \text{ w}} \Rightarrow 2^{27-12} \Rightarrow 2^{15}$

No. of frames = $\frac{2^{21} \text{ word}}{4 \times 2^{10} \text{ w}} \rightarrow 2^{21-12} \Rightarrow 2^{9}$



Date / /
Page

8. Consider a system, where, no. of pages = 2K. & Page size = 4KW. Physical Address = 18 bits Calculate, LA & No. of frames in PAS.

~~no of pages~~ = 2K

no. of pagy = $2 \times 2^{10} = 2^{11}$

Page size = 4KW.

$= 2^{12}$ w

d = 12

P = 11

$\Rightarrow$ d+P = 23

PA = 18 bits

$= 2^{23}$

PAS = 2^{18}

LAS = no. of pagy x pages

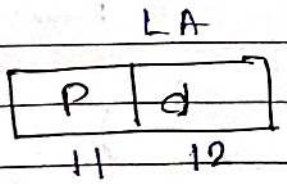
$\Rightarrow 2^{11} \times 2^{12}$

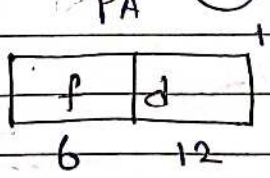
$= 2^{23}$ w = 23.

LA = 23 bits

~~Page table~~ Size = No. of frames = $\frac{2^{18} w}{2^{12} w}$

$\Rightarrow 2^6$





2K = 2^{11}

LA = 23

no. of frames = 2^6

Date / /
Page

Q. Consider a system, which has LAS, of 128 MW and PA = 24 bits. PAS is divided into 8 K frames what is page size and howe many pages are there in LAS.

PAS = 8 K

no. of frams $= 2^3 \times 2^{10} = 2^{13}$ frams

~~213 pages~~

Page size = ?

PA = 24 bits $PAS = 2^{24} w$ $\Rightarrow 2^{13} = \frac{2^{24}}{\text{frame size}}$

LAS = $128 \times 2^{20} w$ $\Rightarrow$ page size $= 2^{24-13}$

$\Rightarrow 2^{27} w$. frame size $= 2^{11} w$

no. of pages $= \frac{2^{27} w}{2^{11}} \Rightarrow 2^{16}$.

$\Rightarrow$ page size $= \frac{2^{27}}{2^{13}} = 2^{14}$

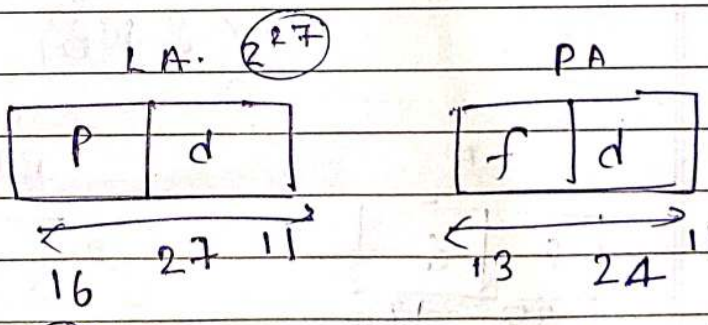
LAS = 128 MW

$= 2^{27} w$

PA = 24 bits

PAS = $2^{24} w$

No. of



2^{16} – Pags

8K = 2^{13} frame

Q. Consider a system which has LA = 32 bits, and PAS = 64 MB, Page size = 4 KB, Memory is Byte addressable. What is approximate size of page table in Bytes.

LA = 32 bits

LAS = 2^{32} Byte

no. of pages = $\frac{2^{32}}{\text{Page size}} = \frac{2^{32}}{2^{12}} = 2^{20}$

~~Page table = 2^{20} x~~

$\Rightarrow 2^{32}$

no. of Frame = $\frac{\text{PAS}}{\text{Page size}} = \frac{2^6 \times 2^{20}\ B}{2^{12}}$

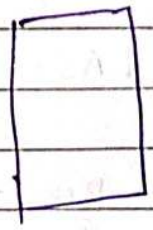
$\Rightarrow 2^{26-12} = 2^{14}\ B$

$= 14$ bit

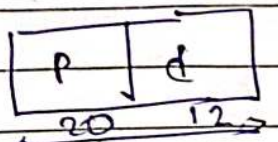
Page Table = $2^{20} \times 14$-bit

$\Rightarrow 2^{20} \times 2B$

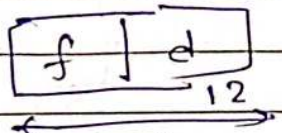
$\Rightarrow$ 2 MB



LA



PA



PAS = 64 MB

$= 2^{26} B$

$\Rightarrow$

Page size = 4 KB

$= 2^{12}$

no. of pages = 2^{20}

$\Rightarrow$ PTE = 14 bits

$= 2B$

Page Table Size = $2^{20} \times 2B$

$= 2$ MB

Date / /
Page

Q. Consider a system, which has, page table with 4K entries, and LA = 29 bits, what is PA if the system has 512 frames.

Page table Entry size = 2^{12} = no. of pages

LA = 29 bits, LAS = 2^{29} B.

no. of frames = 512

no. of pages = 512.

PA = ?

Page table size = ~~512 x ...~~

$\Rightarrow$ ~~$2^9 \times 2^{17}$~~

$\Rightarrow$ ~~2^{17}~~

~~2^{17}~~ ∴ no. of pages = $\frac{2^{29}}{2^{12}}$

$\therefore 2^{17}$

no. of pages = $\frac{2^{29}}{\text{page size}}$ ~~$2^{17} \times$~~

$\Rightarrow 2^{12} = \frac{2^{29}}{\text{page size}}$

$\Rightarrow$ page size = 2^{17}

$\Rightarrow$ ~~page table size~~ = $2^{17} \times 2^9$

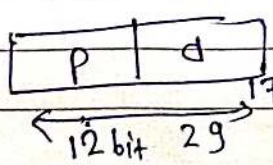
PAS = 2^{26} PA = 26.

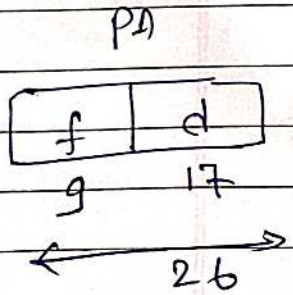
No. of frames = $\frac{\text{PAS}}{\text{frame size}}$

$\Rightarrow 2^9 \times 2^{17}$ PAS

$\Rightarrow 2^{26}$ = PAS

$\Rightarrow$ PA = 26



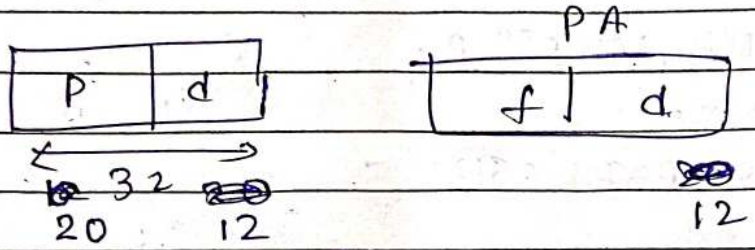


Q. LA = 32 bits

Page size = 4 KB

Page table Entry size = 4 B

Calculate Page table size.

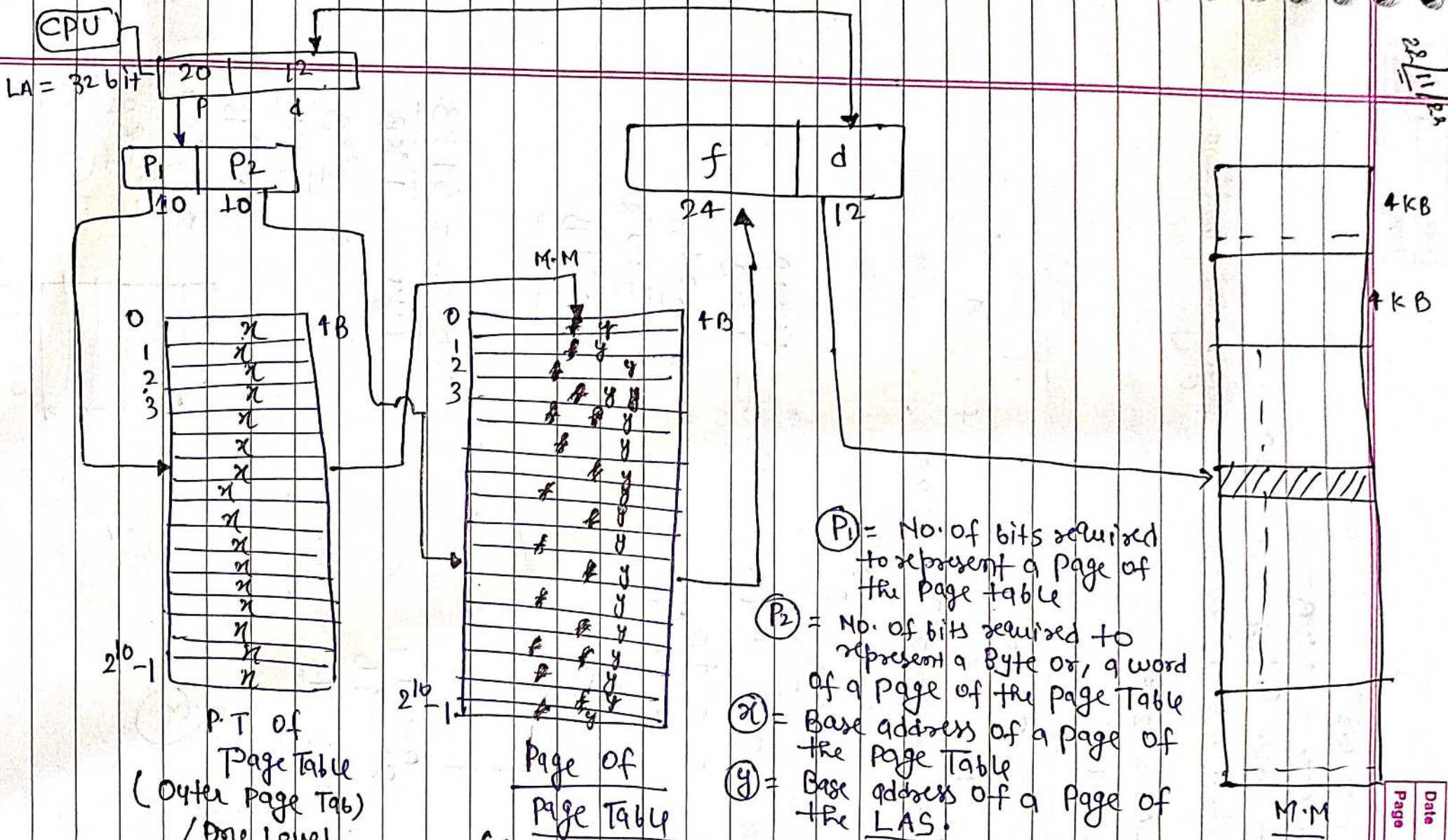


Page size = 2^{12} 4 B

$\Rightarrow$ 2^{20} X 4 B

$\Rightarrow$ (4 MB) $\Rightarrow$ Page table Size

→ Paging completely removes, External fragmentation.. but Paging does not completely removes ~~Exter~~ internal fragmentation, because, the last page of the program may not be equal to the size of page.



(P1) = No. of bits required to represent a Page of the Page table

(P2) = No. of bits required to represent a Byte or, a word of a Page of the Page Table

(x) = Base address of a Page of the Page Table

(y) = Base address of a Page of the LAS.

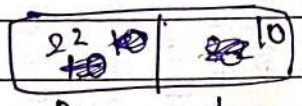
Two Level paging (Multilevel paging)

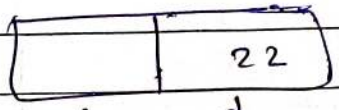
Date / /
Page
RANKA

LA = 32 bit
Page size = 1KB
Page table Entry size = 4B
How many levels of paging will be there.

LAS = 2^{32}

Page size 2^{10}



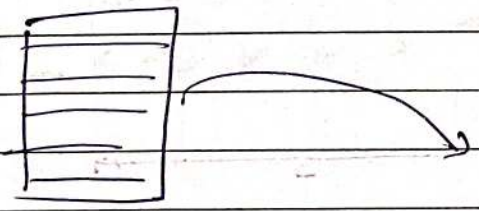


frame size = 2^{10} = 1KB

$\Rightarrow \frac{2^{32}B}{2^{10}B} = 2^{22}$

P. Table Size = $2^{22} * 4B$
$= 2^{24}B$
$= 16MB$

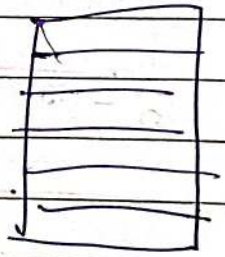
No. of Pages in PT. = $\frac{16MB}{1KB}$
$= \frac{2^{24}B}{2^{10}B}$
$\Rightarrow 2^{14}B$



Outer P.T
$2^{14} \times 4B$
$\Rightarrow 2^{16}B \Rightarrow 64KB$

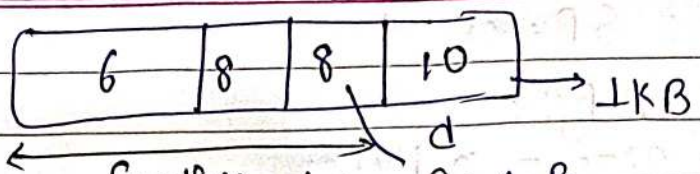
No. of Pages = $\frac{64KB}{1KB}$
$= \frac{2^{16}B}{2^{10}B}$
$= 2^6 B$
$\Rightarrow 2^6 \times 4B$
$= 2^8 B$
$\Rightarrow 256B$

No. Of Levels = 3



Outer P.T

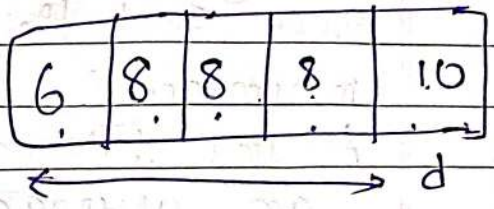
LA = 32B



Trick

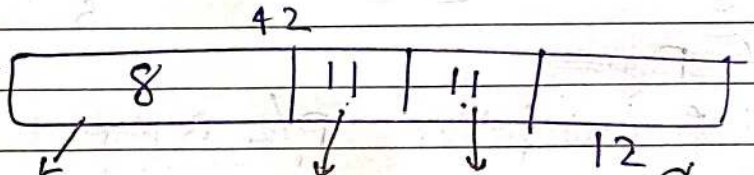
further divided (P) $2^8 \times 4B$

LA = ~~64~~ 40



Q. LA = 42 bits
Page size = 4KB
Page table Entry size = 2B

$2^{12} \times (2)$



One Level | Two Level | Three Level

24/11/22

Date / /
Page

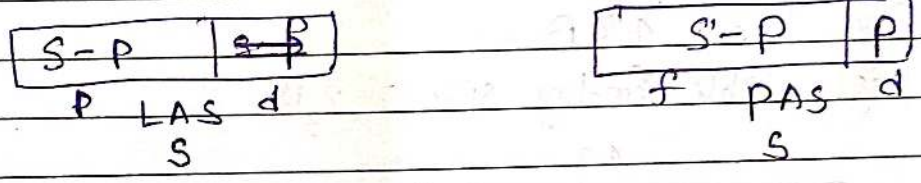
Q: Let LAS = PAS = S Byte

Page size = P Byte

Page Table entry Size = e bytes

what would be the optimal page size so that the memory overhead of maintaining the page table size and the internal fragmentation is minimized?

Assume internal fragmentation = P/2 Byte on an average.



~~No. of Pages~~

internal fragmentation = P/2 Bytes.

$\frac{S}{P}$ (e)

y = PTSize + If.

memory overhead $= \frac{S}{P} * e + \frac{P}{2}$

PTSize, internal fragmentation

$\frac{dy}{dP} = \frac{S}{P} * e + \frac{P}{2}$.

$\Rightarrow S \cdot e \cdot \frac{-1}{P^2} + \frac{1}{2} = 0$

$\Rightarrow \frac{S \cdot e}{P^2} = \frac{1}{2}$

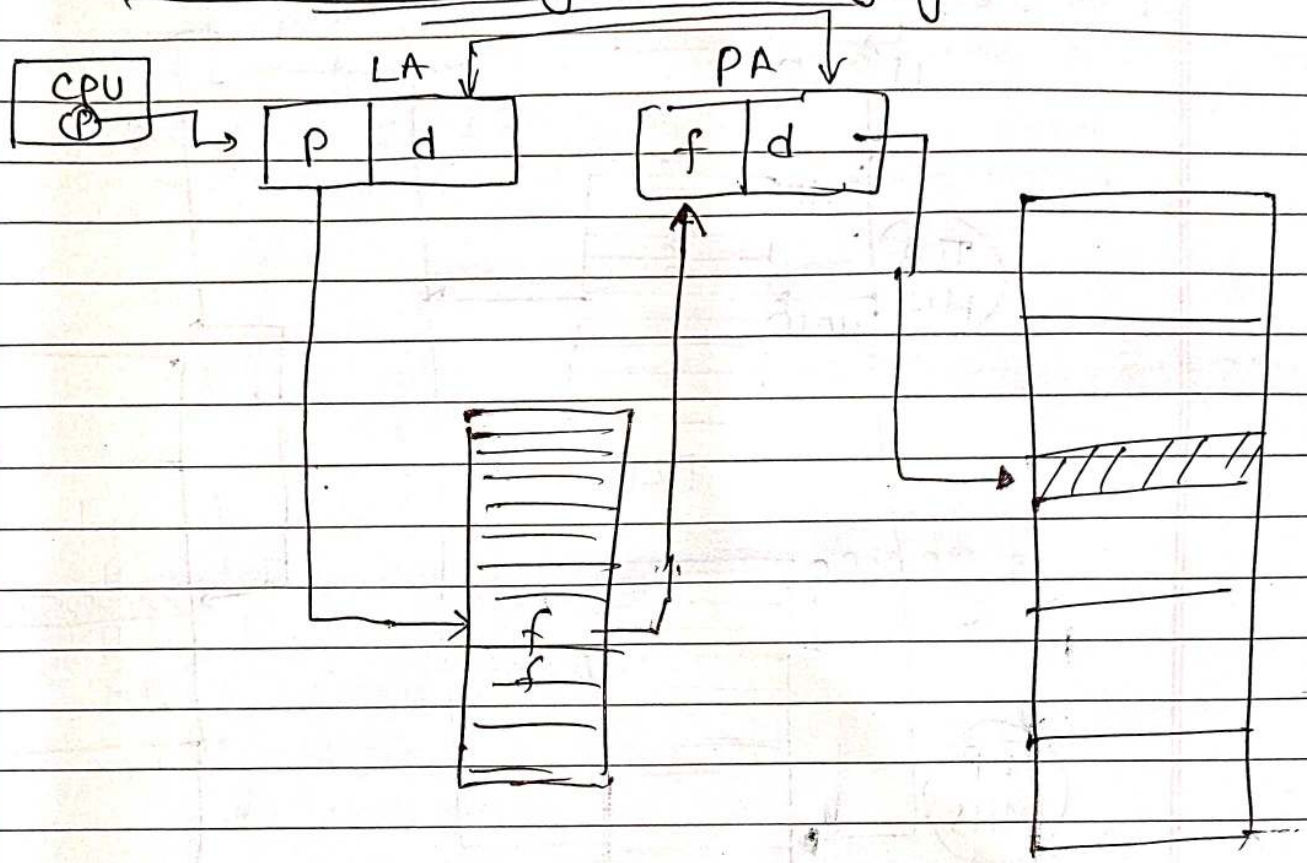
$\Rightarrow P^2 = 2SE$

$\Rightarrow \boxed{P = \sqrt{2SE}}$

Date / /
Page

RANKA

✡ Performance of Single Level paging →



Let m = main memory access time

Effective Memory Access time = 2 * m
(EMAT)

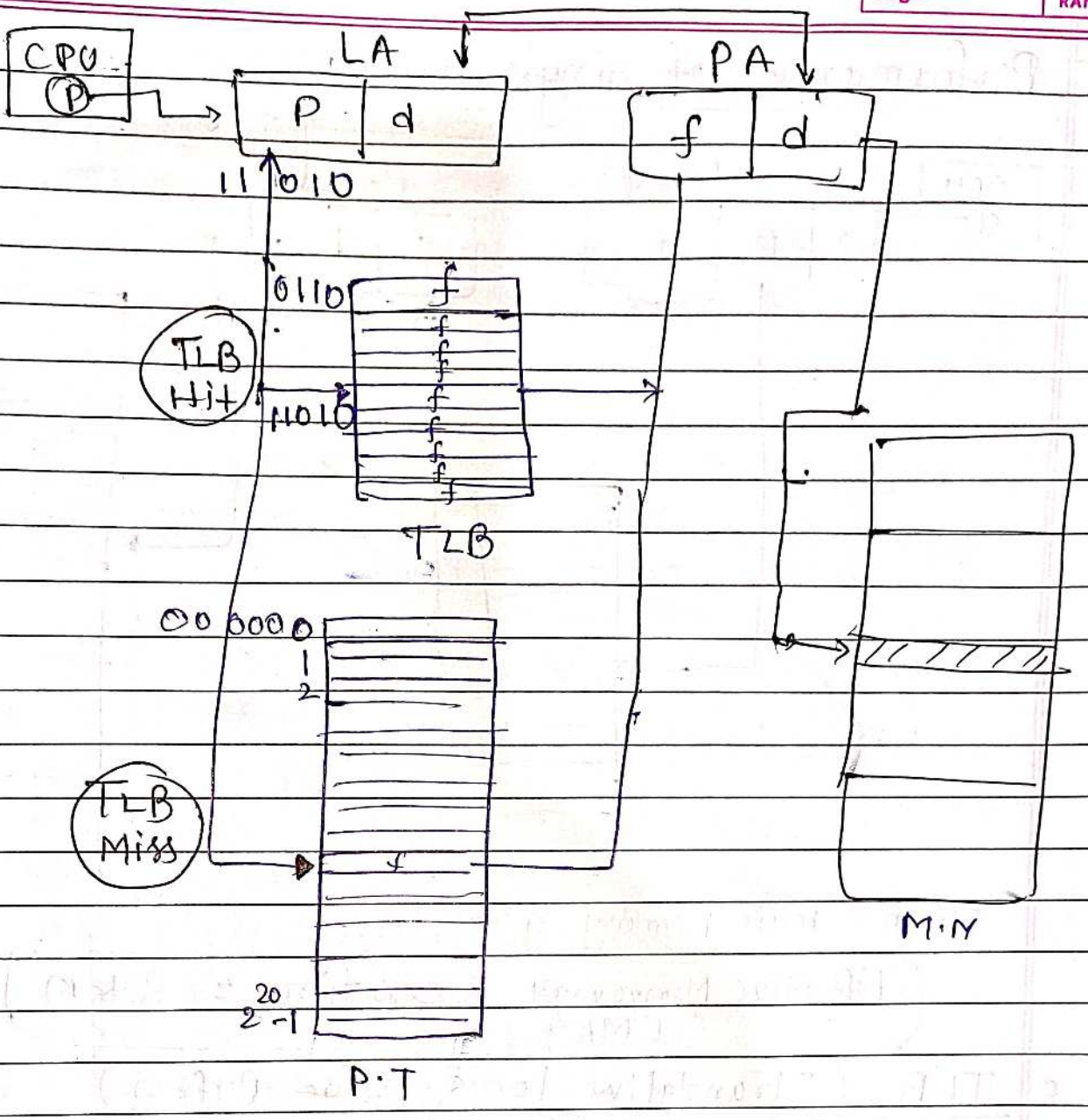
- **TLB** (Translative look-aside Buffer)

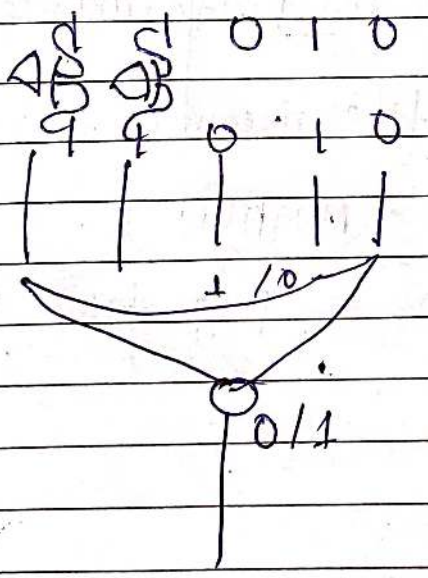
→ TLB is added to improve the performance of paging

→ It is one type of Associative memory.

Cache ⇌ Main memory
TLB ⇌ Page table } Relation

→ Most frequently used pages of the page table are kept in TLB.





Let m = MMAT

C = TLB accesstime

α = TLB hit ratio

EMAT =

$\Rightarrow \alpha(C+m) + (1-\alpha)(C+m+m)$

$\Rightarrow \alpha(C+m) + (1-\alpha)(C+2m)$

Date / /
Page

Q. Consider a system, where MMAT = 100 ns = m
TLB access Time = 20 ns = c
TLB Hit rate = 95 % = x
what is EMAT with TLB, and without TLB.

with TLB

$$EMAT = \frac{95}{100}(0.20*100) + (1-0.95)(20+200)$$

$$\Rightarrow 0.95(120) + (0.05)(220)$$

$$\Rightarrow \frac{0.95 \times 120}{100} + \frac{5}{100} \times 220 \Rightarrow 114 + 11$$

$$\Rightarrow 125 \text{ ns}$$

EMAT without TLB
= 2m
= 200 ns

Q. what hit ratio is required, to reduce EMAT, from 300 ns, without TLB, to 250 ns with TLB.
TLB access Time = 60 ns.

$300 = 2m$
$\Rightarrow m = 150$

$$250 = x(60+150) + (1-x)(60+300)$$

$$\Rightarrow 210x + 210 = 210x$$

$$\Rightarrow 210x + 360 - 360x$$

$$\Rightarrow -150x + 360$$

$$\Rightarrow 250 - 360 = -150x$$

$$\Rightarrow -110 = -150x$$

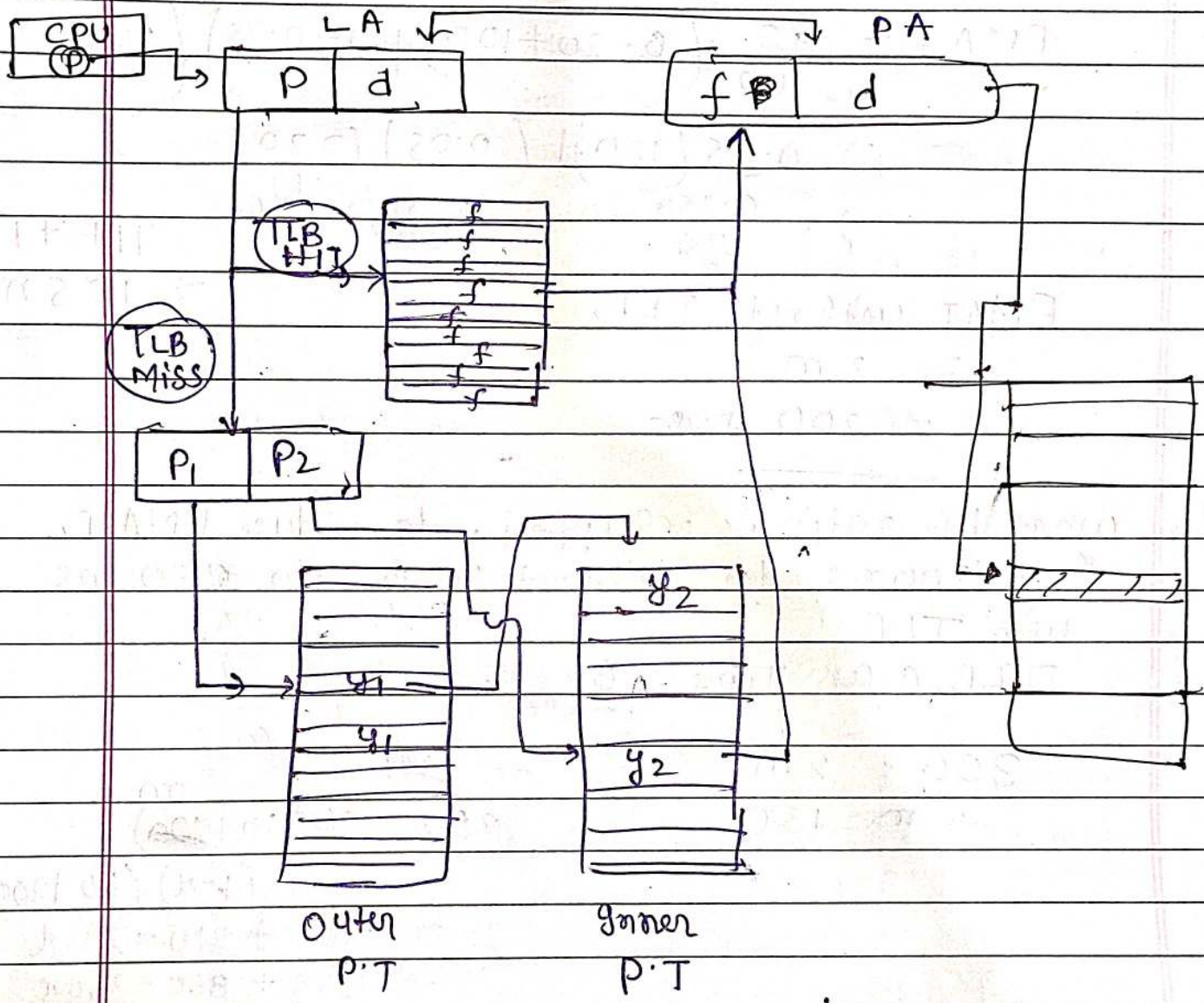
$$\Rightarrow x = \frac{11}{15}$$

Performance of two level paging

Let m = MMAT

EMAT = (3m) ⇒ ~~m m m~~ m + m + m

PT of PT · P·T · M·M



EMAT with TLB

= x (C + m) + (1 − x) (C + m + m + m)

~~m~~ PT_1 PT_2 M·M

(C + (n+1) m)

n-Level

Logical address → physical address mapping

Date / /
Page

$$m = t * c_1 + (1-t)(c_1 + m)$$

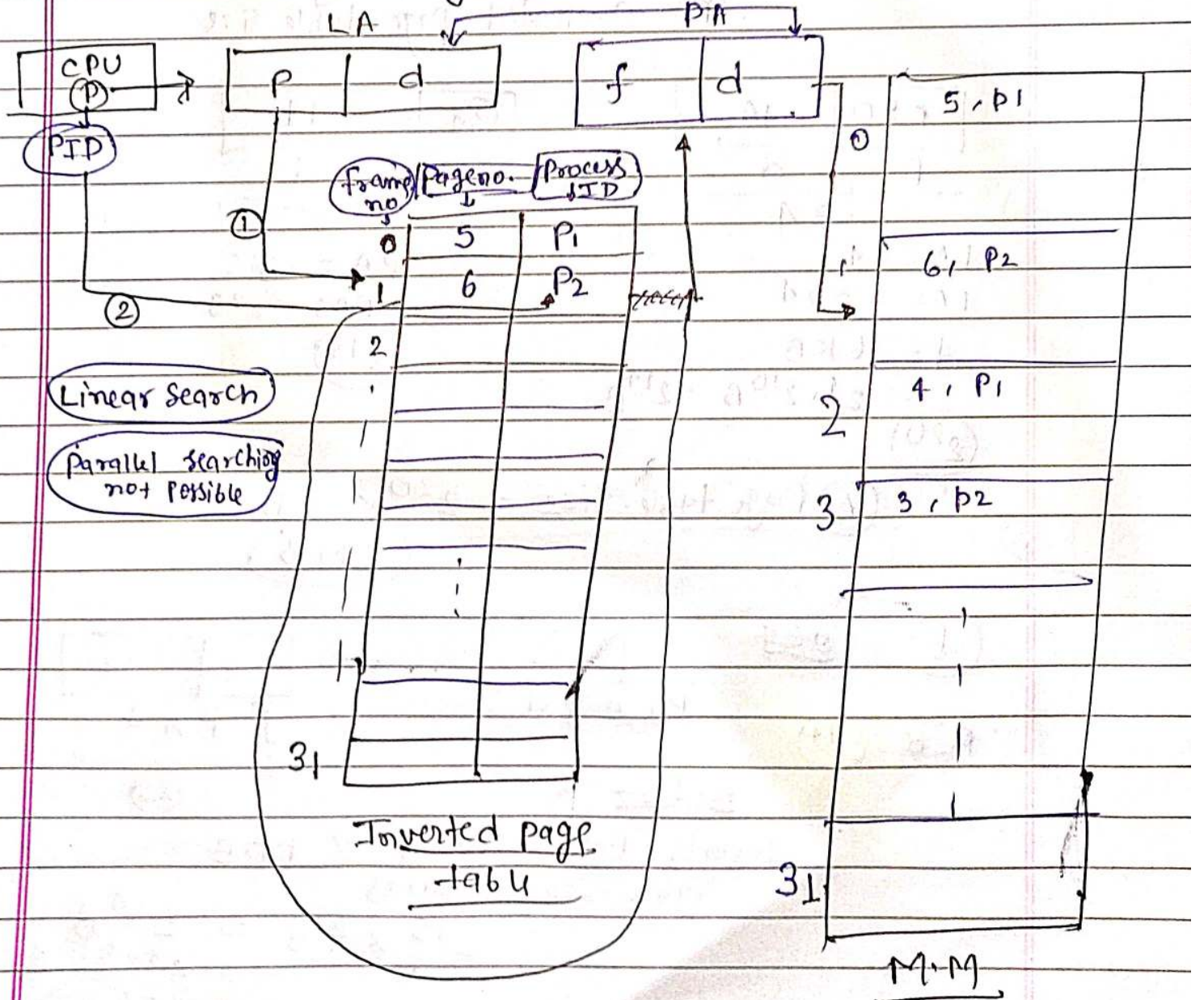
↓

cache ✓

Inverted Paging →

→ Disadvantage of Multilevel
↳ Many page tables are required

→ There will be only one page table, and this page table will be shared by all the processes.

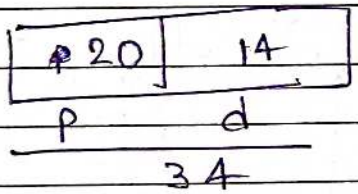


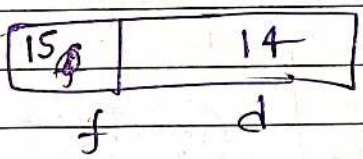
→ The no. of Entries in the Inverted page table will be equal to no. of frames in M.M.

→ **Linear Search** is the disadvantage of Inverted Paging

Q: Consider a System
LA = 34 bit
PA = 29 bit
Page size = 16 KB
Page table Entry Size = 8 B

Calculate no. –
(A) – Conventional page table size
(B) Inverted page table size



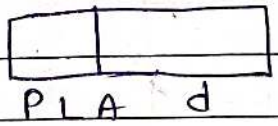


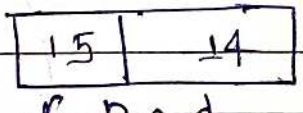
LA = 34
LAS = 2^{34}
d = 16 KB
⇒ $2^4 \times 2^{10}$ B = 2^{14} B
2^{20}

PA = 29
PAS = 2^{29}
2^{15}

(A) Page table size = $2^{20} \times 8$ B
⇒ 8 MB

(B)





d = 2^{14}

29

Inverted Page Table Size = No. of frames × PT.E.
= $2^{15} \times 8$ B ⇒ 2^{18} B
⇒ $2^8 \times 2^{10}$ B
= 256 KB

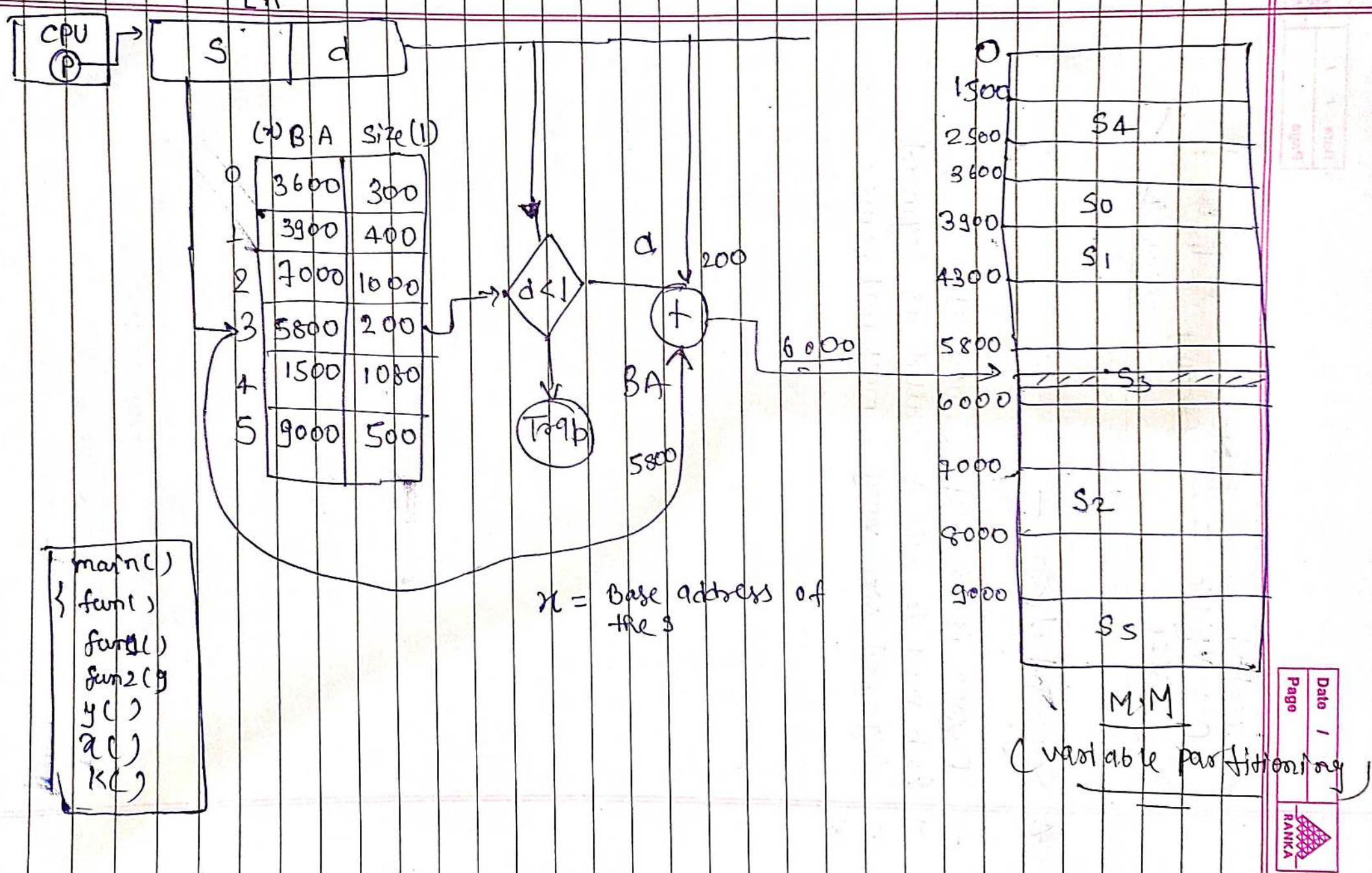
Date / /
Page

RANKA

Inverted page table size = No. of frames x PTE

✡ Segmentation –

→ Program is made up of modules.
→ Each module will be called as segment
→ Segment size vary, may not same.
→


LA
CPU
P
S
d
(x) B A Size(l)
0 3600 300
1 3900 400
2 7000 1000
3 5800 200
4 1500 1080
5 9000 500
d<l
Trap
d
200
+
6000
BA
5800
x = Base address of the S
0
1500
2500
3600
3900
4300
5800
6000
7000
8000
9000
S4
S0
S1
S3
S2
S5
M.M
(variable partitioning)
main()
{ fun1()
fun2()
fun3()
y()
a()
k()

Date / /
Page

→ ~~Facts~~ Disadvantage

↳ Segment size is greater than the size of the M.M. Blocks

↳ External fragmentation is the main cause so, for removing the External fragmentation we will apply paging on Segment (d)

→ d → will be broken down.

Date / /
Page

RANKA

✡ Segmented Paging →

→ Segment
↳ Page table

→ Every segment has its own page table.

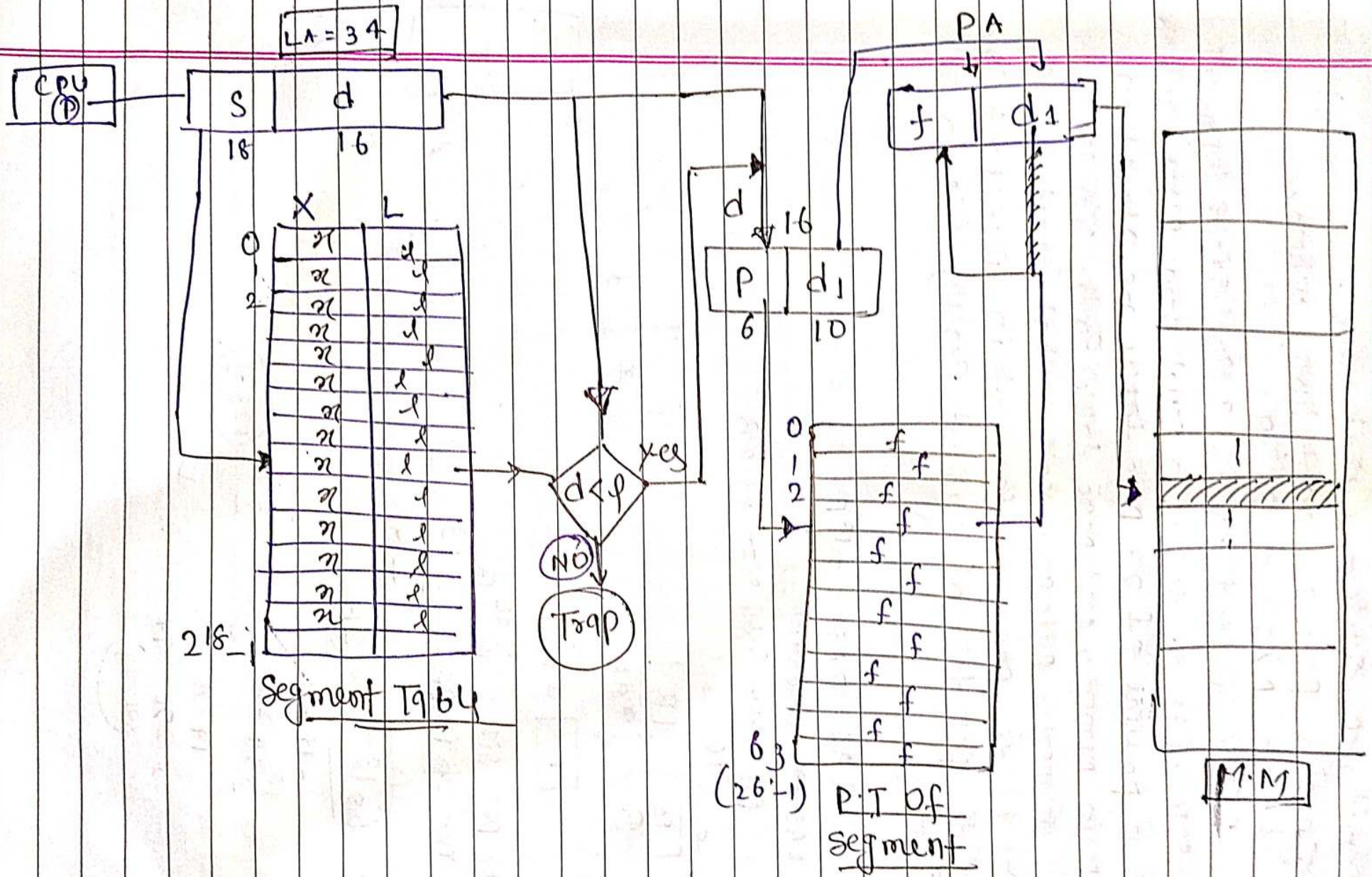


Fig. Segmented paging on segment

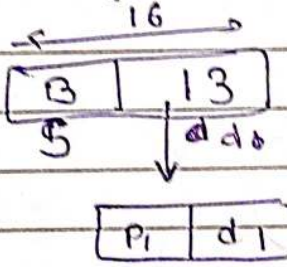
Q. Consider a system that uses Segmented paging architecture, where LAS = PAS = 2^{16} B

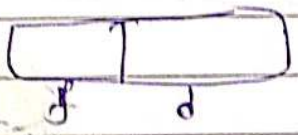
LAS is divided into 8 equal-sized segments. The segment is divided into equal sized pages, which are in powers of 2. Page tables are stored in main memory, and page table Entry size is 2 B. The memory is Byte addressable. what must be the size of the page of segment in Bytes so, that the page table of segment exactly fits in one frame.

LAS = PAS = 2^{16} B

LAS ⇒ 8 equal

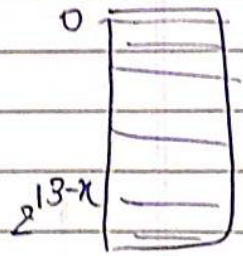
Page table Entry size is 2 B.





Let Page size = 2^x B

no. of Pages = $\frac{2^{13}}{2^x} = 2^{13-x}$



⇒ $2^{13-x} \times 2B$

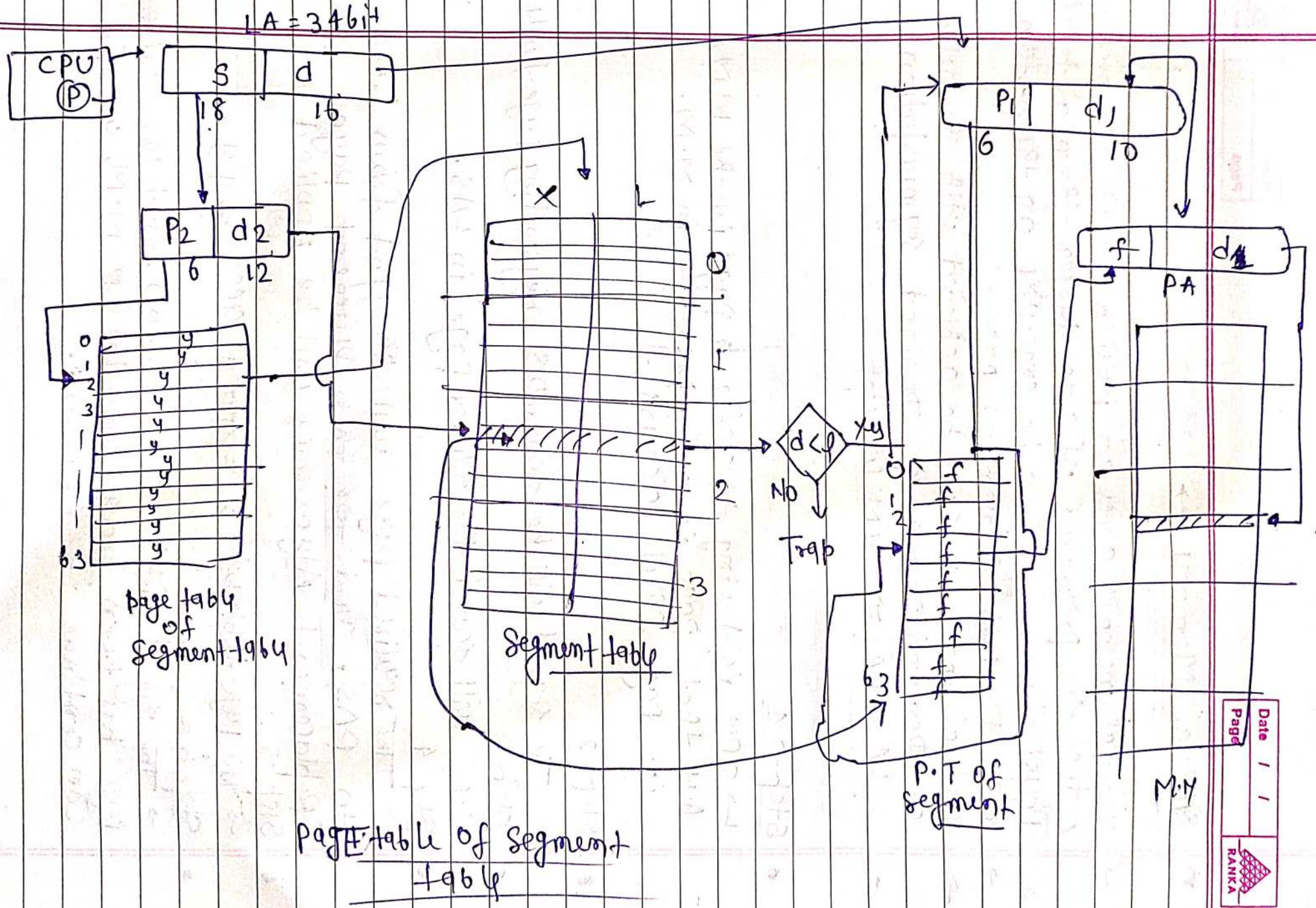
⇒ 2^{14-x}

⇒ $2^{14-x} = 2^x$ $\quad 2^7 = 128$ B

⇒ $14 - x = x$

⇒ $2x = 14$

$x = 7$



Page table of Segment table

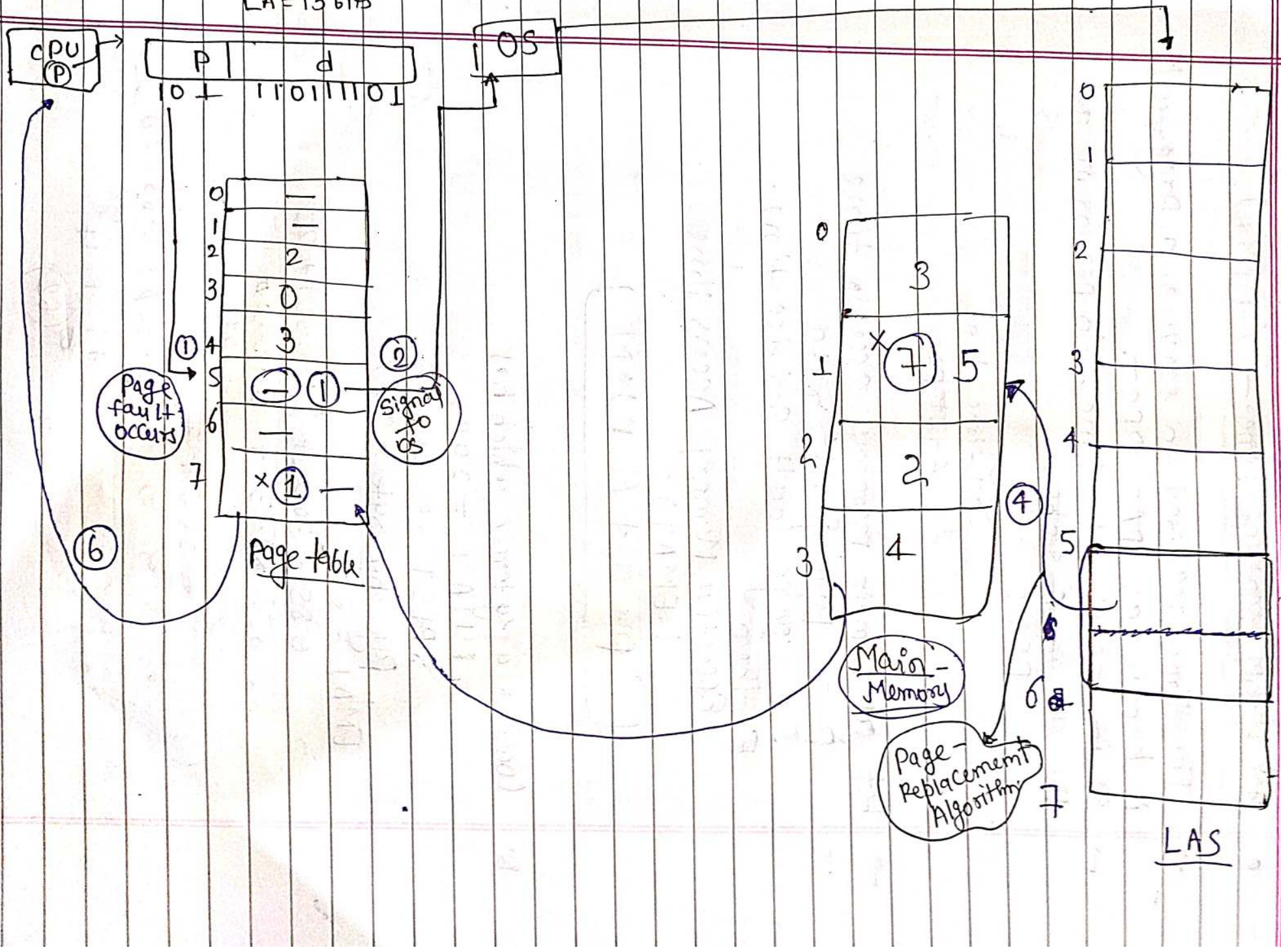
Date / /
Page

RANKA

Virtual Memory →

→ Virtual memory gives an illusion to the programmer, that the programs of size, larger than actual Physical memory, can be executed on the system.
→ with the help of "Demand paging"
→ Virtual memory can be achieved with the help of Demand paging or Demand segmentation.
→ [LAZ PA] ⇒ VM concept

- **Step-> 1**
  ↳ CPU is trying to refer a page in the MM, but that page is not present in M.M.
  i.e. Page fault occurs.
- **Step-2**
  → The signal will be sent to OS, Regarding Page-fault.
- **Step-3**
  → OS will search the required page in LAS.
- **Step-4**
  → The required page will 'Brought from LAS to PAS, and for this Different Page "Replacement Algorithms will be Applied.
- **Step:-5**
  → The page table will be updated accordingly.
- **Step-6:-**
  → The signal will be sent to CPU, that the required page has been brough into the M.M, and it can continue its execution.


LA = 13 bits
CPU
P
p
d
1 0 1 1 1 0 1 1 1 0 1
OS
Page fault occurs
Signal to OS
Page table
Main Memory
Page Replacement Algorithm
LAS
1
2
3
4
5
6
7

Date / /
Page

RANKA

- **Page fault service time – (PFST).**

→ The time required to service a page fault is known as ~~PAS~~ PFST.

→ It ~~is~~ includes the time, to perform all the above 6 steps.

PFST → S

all time doing 6-steps

→ Let M = main memory access time
P = Page fault rate
S = page fault service time.
E. then
Effective Memory Access time (EMAT) =

$$= P * S + (1-P) * M$$

Q. Consider a system, which has
MMAT = 20ns
PFST = 300 ns.
Page hit rate = 85%.
EMAT = ?

⇒ $0.85 * 300 + (1-0.85) * 20$

⇒ $85 * 3 + 0.15 * 20$

$\frac{100}{5}$

~~⇒ 255 + 3~~

⇒ $\frac{0.15}{100} \times 300 + \frac{0.85}{100} \times 20$

⇒ $45 + 17$

⇒ 62

Date / /
Page

Q Consider a system, using demand paging Environment, where it takes, 8 ms, to service a page fault, if either an empty frame is available, or, the page to be replaced is not modified. it takes 20 ms, if the page to be Replaced is modified.

MMAT = 1 ms

Assume the page to be replaced, is modified 70% of the time. what is the maximum acceptable Page fault rate to get an EMAT, not more than 2 ms.

PfSt = 8 ms. } no updation

PfSt = 20 ms — ~~write~~ updation.

MMAT = 1 ms

EMAT = P * S + (1-P) * M

~~2 =~~ ~~70/100 x 8 + (1-0.7) * 1~~ + ~~70/100 x 20 + (1-0.7) * 1~~

PfST (S) = 0.3 * 8 + 0.7 * 20

⇒ 2.4 + 14

S ⇒ 16.4

EMAT ≤ 2

⇒ P * S + (1-P) * M ≤ 2

⇒ P * 16.4 + (1-P) * 1 ≤ 2

⇒ 16.4P + 1 - P ≤ 2

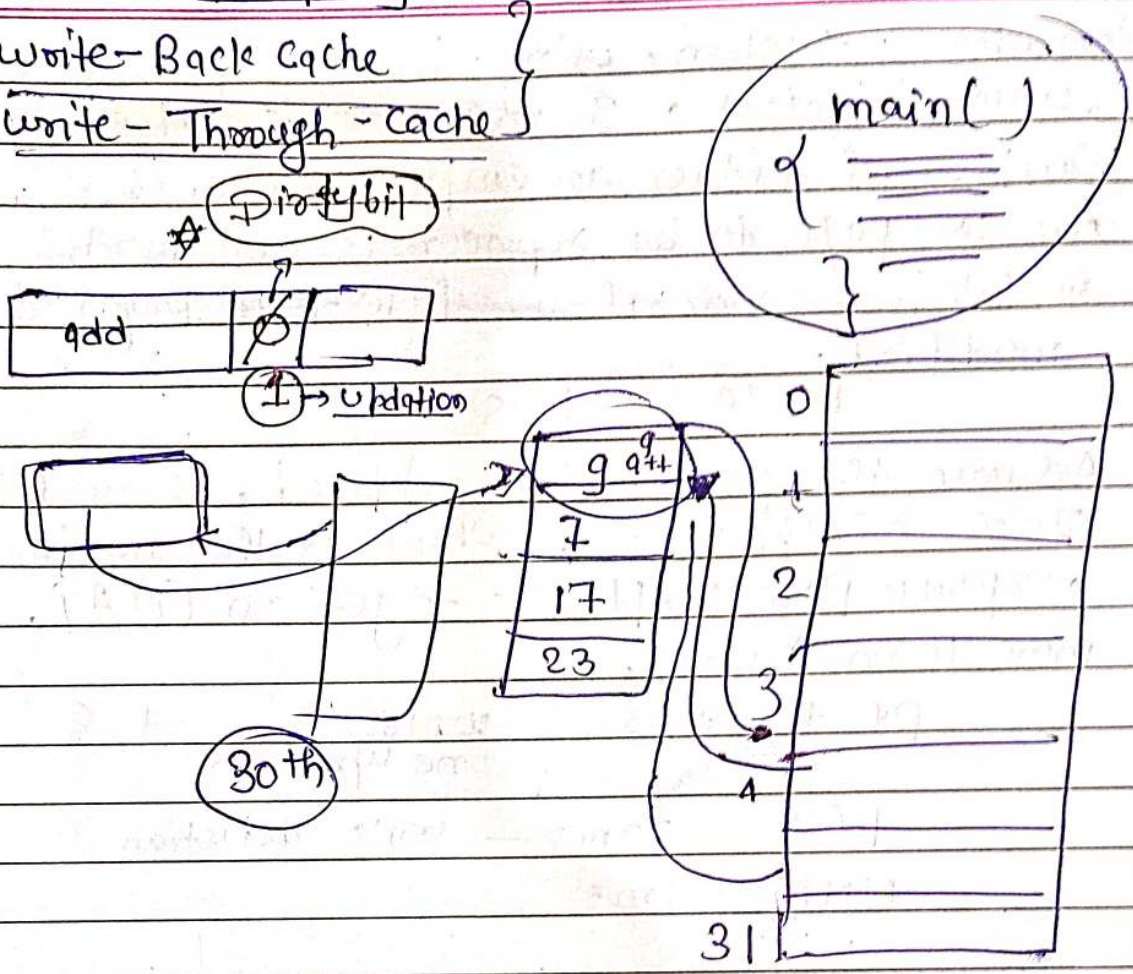
⇒ 15.4P + 1 ≤ 2

⇒ $P \leq \frac{1}{15.4}$

Date / /
Page

Updation policy

Write-Back Cache
Write-Through-Cache



- One way Copy
- Two way Copy

Dirty bit — 0 / 1
0 → not updated
1 → updation done

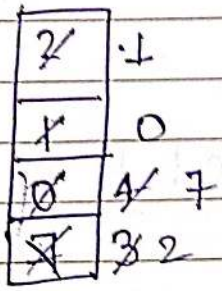
Write Back
↳ content of cache and M.M may not be same at any pt. of time

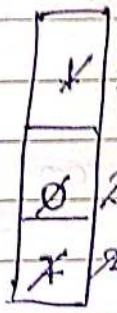
write through
↳ Content of cache and M.M., will be same at any pt. of time.

Page Replacement Algorithms →

Referenced Strings = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

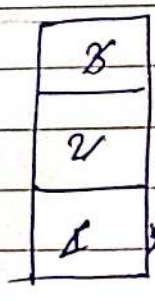
- FIFO



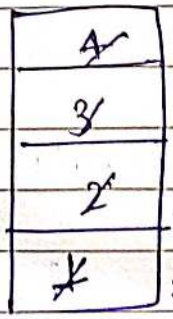


Referenced Strings -

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



(9)



(10)

✡ Belady's Anamoly →

→ Generally the page fault should decrease with the increase in no. of frames but Here in above example the no. of page fault increases with the increase in no. of frames.

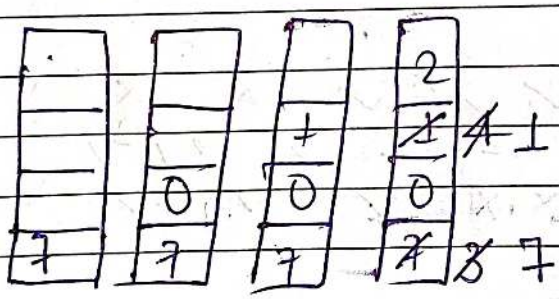
→ This is known as Belady's Anamoly, and it happens only in FIFO.

- **LRU (Least Recently Used)**

Reference strings

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

✓ ✓ ✓ ✗ ✓ ✗ ✓ ✗ ✗ ✗ ✗ ✗
✓ ✗ ✗ ✗ ✓ ✗ ✗



Left farthest past

= 8

Date / /
Page

RANKA

- Optimal Replacement Algorithm (ORA)

Referenced String.

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

Right - farthest future

one - not match
↳ ignore

Two - not match
↳ Apply FIFO

- **MRU (Most Recently used)**

  ↳ Shortest → removed

- 3 2 1 1 4 7 1 0

  300, 250, 120, 150, 450, 750, 100, 80

  Byte no. These are addresses reffered by CPU not Program

  ↓ Byte no.

where page size = 100 Byte

100) 300 (3
300

×

1024) 5192 (5
5120

0072