Book :- Galvin

OS (8 - 9) Marks

① Introduction & Basics
- → what is OS ?
- → Goals & function ?
- → Evolution
- → types of os

② Process management
- Basics
- → what is Process
- → PCB
- → Context switch

scheduler
- → CPU (1-2) scheduling.

Synchronisation (1-2)

Deadlock (1-2)

③ Memory management (3-4) mark.
- → Main memory
- → space allocation
- → address translation.

④ Extra
- → File management
- → I/O devices.
- → For Command
- → Kernal
- → CLI
- → System Call
- → Thread

① what is operating system?

→ ① operating system is a system software which is

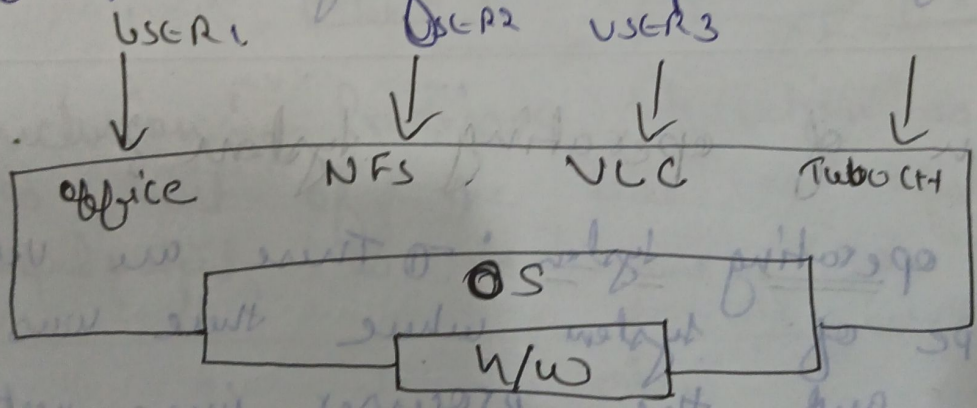
① A intermediatory between the hardware and the user.

② It works as a Resource manager and distributes Resource among process in a unbiased fashion.

③ It works as a platform on which other applications are installed.

Example :- Android, Linux, windows.

Abstract view of a Computer.



* Goals and function :-

Goals are the generalised thing for which operating systems are designed.

① Primary Goal is Convinience (User friendly)

② Secondary Goal Efficiency.

* Functions :- (PMIFNSP)

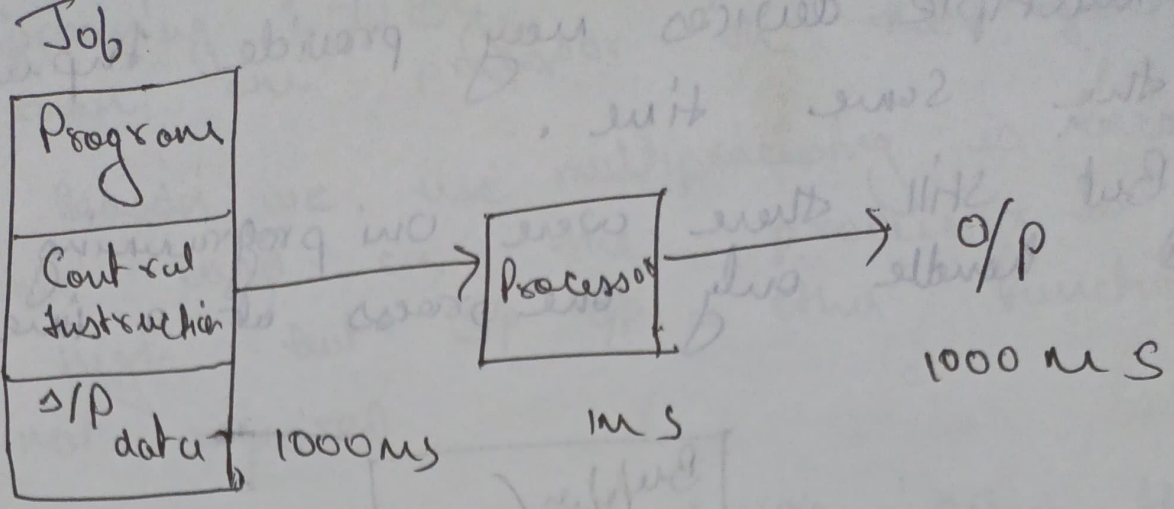
Functionalities are the function which operating system perform in order to accomplish Goals.

① Process management
② memory management
③ I/O management
④ File management
⑤ Network management
⑥ security & Protection management.

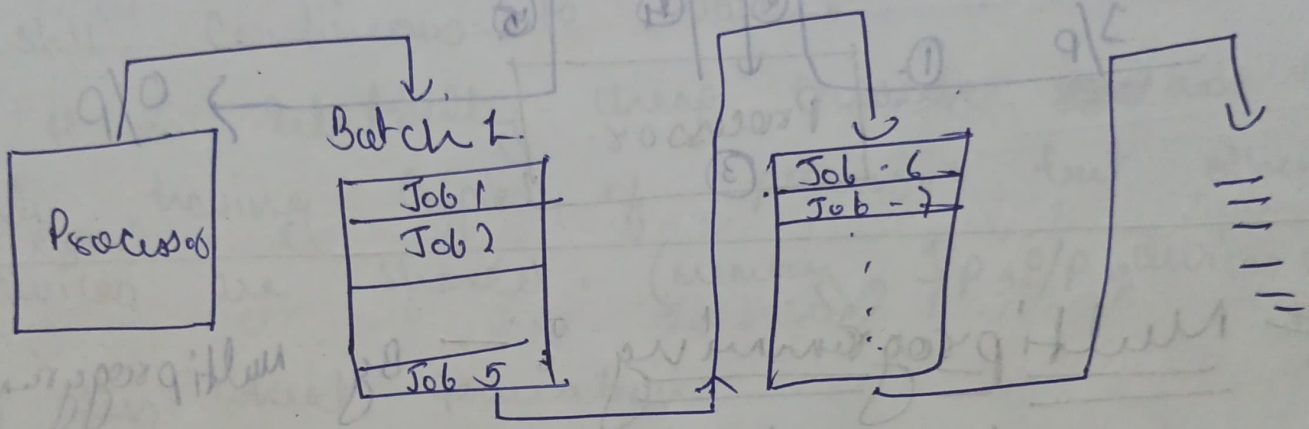
Evolution of operating system :

① Batch operating system :- These are very old type of system where there was no memory and the processor were not interactive therefore user use to prepare a Job which Contains program, Control instructions and input data.

② similar type of Jobs are Grouped Together To form a Batch and processor Executes Batch by Batch.



CPU usage :- $\frac{1\ ms}{2001\ ms} \times 100$



Conclusion :- Processor Utilization is very poor as most of the time is wasted waiting Input output as no memory is in the device

---

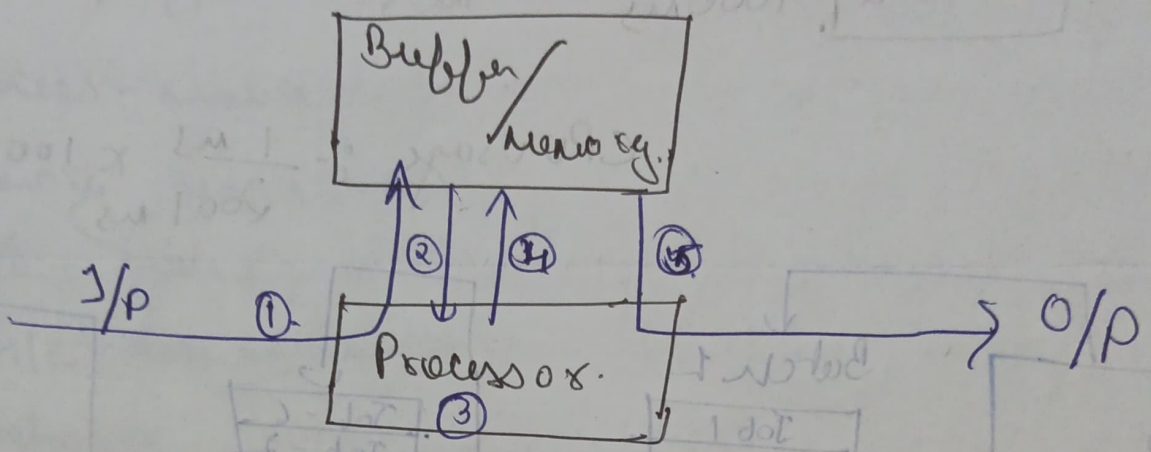
* Spooling :- Simultaneous pheripheral operation online

It is the first time when memory was introduced in the system. what ever input user give directly goes into memory and the processor interact only with the memory

* Conclusion :- 1) No interaction b/w the I/P O/P devices at the processor.

2) multiple devices may provide input at the same time.

3) But still there were uni programming machine and handle only one process at a time.



Multiprogramming :- By multiprogramming we means there will be more than 1 process in main memory and if the running process wants to wait for an event like Input output then instead of sitting ideal CPU will wait or Context switch and will pick other process. (the procedure will continue)

Advantage :- ① CPU Utilisation is very high.
② And CPU can be ideal only when there is no process in main memory or at the time of Context switch.

Multiprocessing :- ① By Multiprocessing we mean more than one processor in the system.

② the Reason we use multiprocessing is because sometime load on the processor is very high but I/P O/P or other functions are not required.

Advantage :- ① More Reliable as Even if 1 processor Goes down than the other Can still Continuous to work.

② It is Relativelly cheap because we are only having Copies of processor but other devices are shared. (Memory, I/P, O/P, devices are shared)

③ less heat Generation.

④ less battary Consumption

Disadvantage :-

① more Complex

② difficult to manage

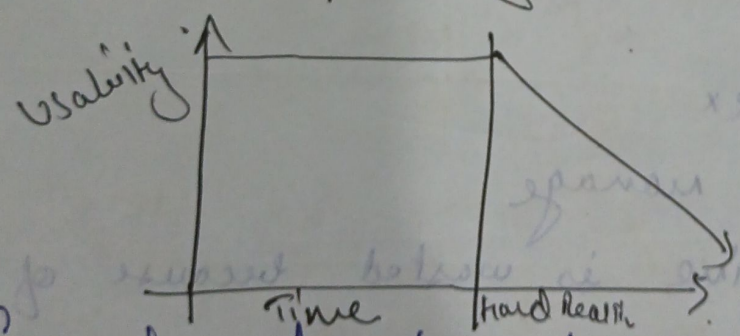
③ And some time is wasted because of extera Coupling.

Multitasking :- the Multitasking or timesharing OR Timesharing the processor is so powerful that it is shared by number of user and processor makes a context switch for each of them so that every user feels that entire system is dedicated for his or her use even though it is shared.

Real time system :-

Realtime system are those system where with every task deadline involved and the task must be completed on or before the deadline otherwise the system will fail.

Example :- weapon system, ATS, Railway signal controlling.



It can be of two types hard Realtime & Soft Real time.

Hard Realtime means usability will drop down to 0 immediatly after crossing the deadline.

Soft Realtime here usability drop down to zero Gradually. Ex :- Visualt Reality, gaming etc

What is process?

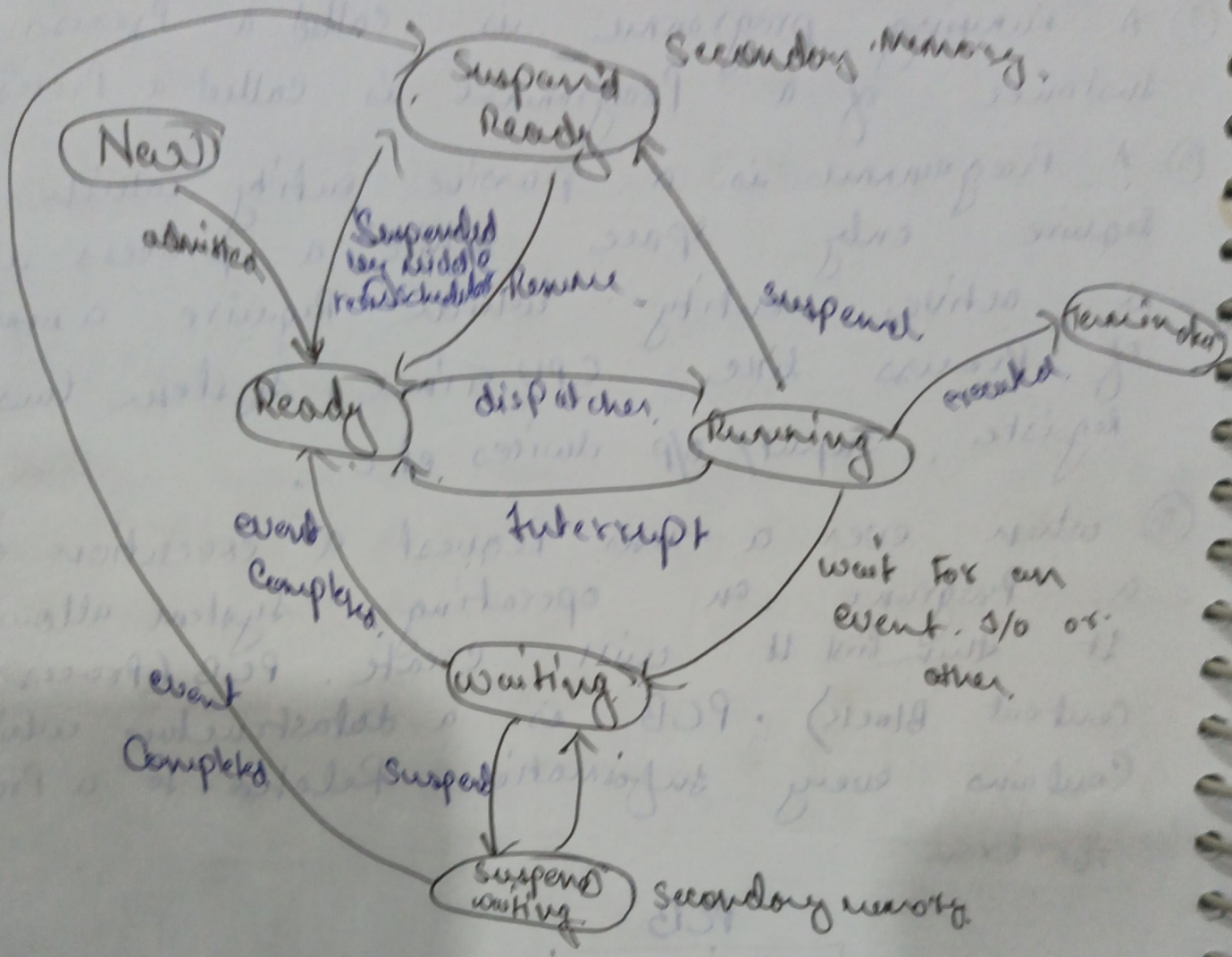
① A programme is a sequence of instruction.
② Every instruction is atomic in nature.
③ A Running programme is called a Process or instance of a Programme is called a Process.
④ A Programme is a passive entity which require only space while a process is a active entity which require a number of Process like CPU Time, system bus, register, input/o/p devices etc.
⑤ when ever a user Request a execution of a Programe an operating system allows it then first it will create PCB (Process control Block). PCB is a datastructure which Contains every information Related to a Proce

For Exam

PCB

P_id	states
Priority	owner
list of files	
list of register	
memory	

⑥ the moment the OS Creates a PCB we say a Process is Being Created.



New state :- A process is said to be new state if the process is Just being created

Ready state :- A process is in Ready state when it is Ready for execution.

Running state :- when A process is scheduled on CPU

waiting state :- when a process is waiting for some event.

Terminated :- If a process has Completed its execution.

Suspand ready :- when the process is Ready. but suspended to secondary memory by middle term scheduler

Suspand waiting :- when the process is in waiting state but suspended to secondary memory.

Scheduler :- Scheduler are the programmes which takes decision that which process will be switching From 1 state to other state.

① short term scheduler :- It picks a process From ready state and allows it to Get scheduled on the processor. [Sends it to running state]

② middle term scheduler :- ~~It temporarily manages the degree of multiprogramming~~ and if there is a process in main memory then it will suspend some of them to secondary memory which can be Resumed later.

③ long term scheduler :- It actually decides the degree of multiprogramming It takes a decision that how many new processes can be Created.

NOTE :- Frequency of short term scheduler is very high compare to long term scheduler.

Frequency of long term scheduler must be equal to through of the system.

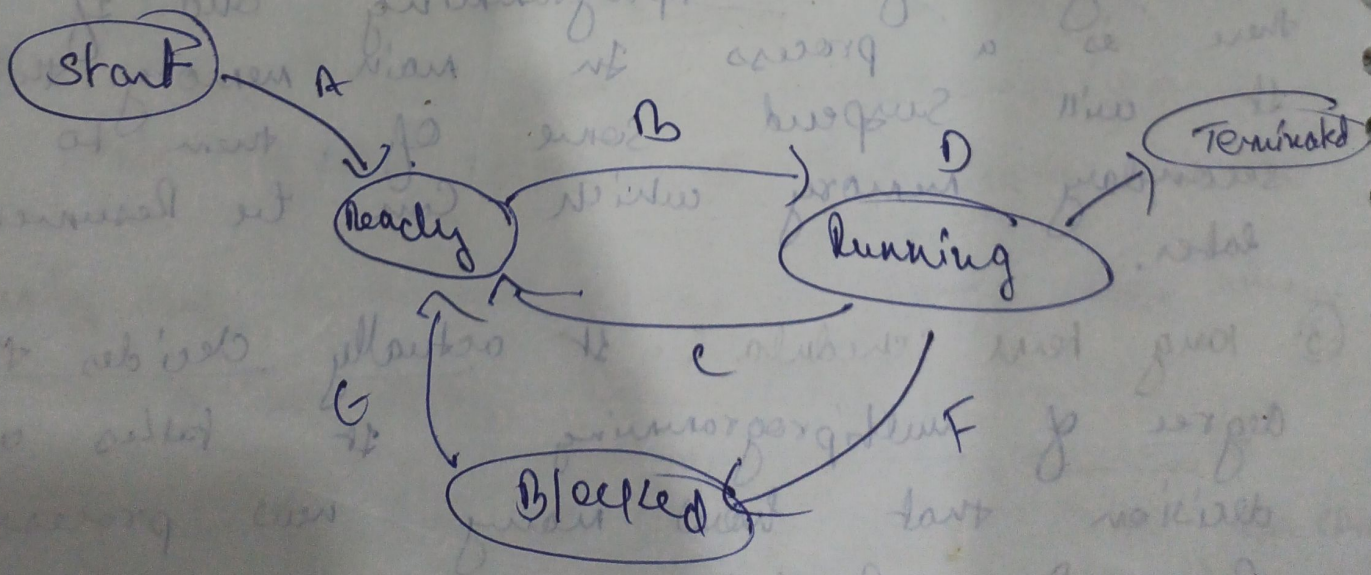
Dispatcher :- dispatcher is a Programme which implies the decision taken by short term scheduler and the time taken in the entire process is called dispatcher latency.

Q If a system has n numb of process and n number of processor.

Process n

Processor n

	Ready	Running	wait
min	0	0	0
max.	n-n	n	n



CPU Scheduling :-

① In General some process are known as Input output bound process which means they require most of the I/O time while other processes are CPU Bound process. which means they require most of the CPU time.

Conclusion :-

there for a good scheduling algorithm should choose combination of both so that both IP/output and CPU are fully Utilized.

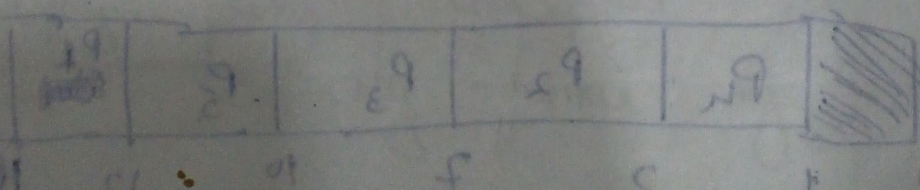
Coiteria For Judging a CPU schedulingalgo.

① Throughput :- The no. of process executed per Unit time.

② CPU Utilization :- For what percentage of time CPU was active.

③ waiting time :- Average waiting time should be low.

④ Average Response time :- Avg. RT should be low.



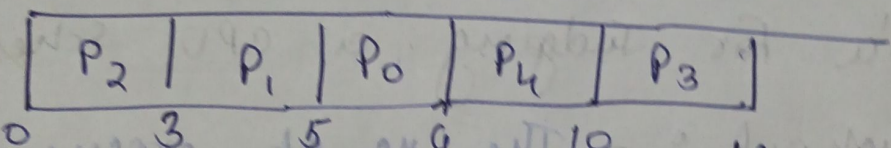
First Come First Serve :-

Q1 Consider the following table and Schedule if using FCFS algorithm?

⇒

	A.T.	B.T.	T.A.T = C.T − A.T	W.T = T.A.T − B.T
P_0	2	4	9 − 2 = 7	7 − 4 = 3
P_1	1	2	5 − 1 = 4	4 − 2 = 2
P_2	0	3	3 − 0 = 3	3 − 3 = 0
P_3	4	2	12 − 4 = 8	8 − 2 = 6
P_4	3	1	10 − 3 = 7	7 − 1 = 6

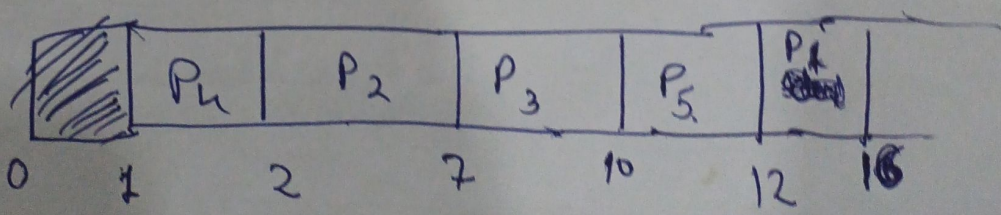
Gantt



Q2

	A.T.	B.T.	T.A.T = C.T − A.T	W.T = TAT − BT
P_1	6	4		
P_2	2	5		
P_3	3	3		
P_4	1	1		
P_5	4	2		

Gant chart.



FCFS :-

① In FCFS algo. at any instance of time out of all available process the process which arrives first will Get the CPU.

Advantage :- ① It is very simple and Easy to understand and implinent.

Disadvantage: FCFS suffers from Convoy effect

Convoy effect.

	A.T.	B.T
P_0	0	100
P_1	1	1

	A.T.	B.T.
P_0	0	100
P_1	1	1

Convoy effect :- when smaller process have to wait For larger process then it is called Convoy effect.

we must Run smaller process before the larger process there fore the turnaround of the waiting of larger process will be very less compare to the degrees the waiting time of smaller process.

Important Points ① FCFS is always non primitive

② Queue is the most appropriate data structure to implement FCFS.

③ FCFS suffers from Convoy effect but no starvation because the processor is unbiased

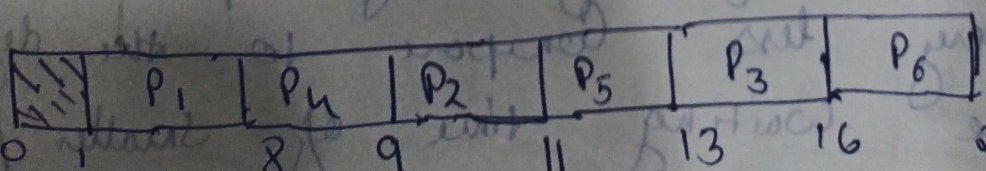
---

Shortest Job first

Q Consider a Table And schedule it shortest Job first scheduling primitive as well as non-primitive.

	A.T.	B.T	T.A.T = Exit time − Arrival time	W.T = TAT − B.T
P_1	1	7	8 − 1 = 7	7 − 7 = 0
P_2	2	2	11 − 2 = 9	9 − 2 = 7
P_3	3	3	16 − 3 = 13	13 − 3 = 10
P_4	5	1	9 − 5 = 4	4 − 1 = 3
P_5	4	2	13 − 4 = 9	9 − 2 = 7
P_6	3	4	20 − 3 = 17	17 − 4 = 13

Gantchart

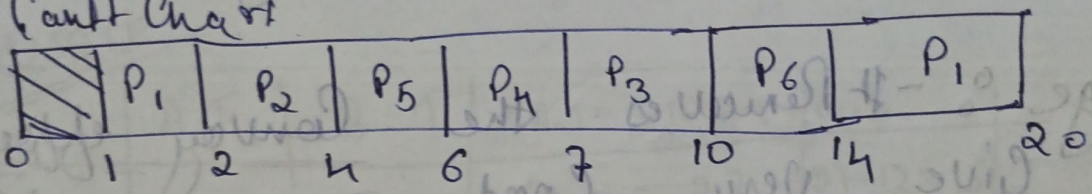


↑ For NON-primitive

For primitive :-

	A.T.	B.T.	T.A.T = C.T - A.T.	W.T = TAT - B.T.
P_1	1	7	20 - 1 = 19	19 - 7 = 12
P_2	2	2	4 - 2 = 2	2 - 2 = 0
P_3	3	3	10 - 3 = 7	7 - 3 = 4
P_4	5	1	7 - 5 = 2	2 - 1 = 1
P_5	4	2	6 - 4 = 2	2 - 2 = 0
P_6	3	4	14 - 3 = 11	11 - 4 = 7

Gantt Chart



Shortest Job first :- ① In this algorithm at any instance of time out of all available process we will schedule a process with the minimum CPU Burst Requirement.

② It is used in both Version.

Non Primitive :- Here Once we take a decision the process is Completed always then we schedule the other process.

[redacted] :-

#Primitive :- Here at any instance of time when a new process arrives. we compare the time requirement of a new process with the Remaining time requirement of ~~process~~ current Running process and if it is less, then the new process Get scheduled.

#SRTF :- It is also known as optimal algorithm It also Guarantees minimum average waiting time.

Advantage :- It Reduces the Convoy effect and Give very Good average waiting time.

DisadVantage :- ① Relatively Complex and FCFS.

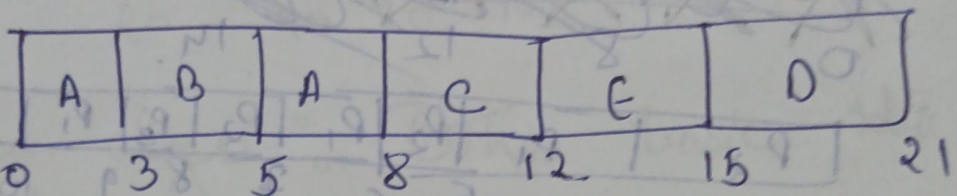
② Here ~~the~~ process with largest CPU Burst requirement will Starve for the CPU.

③ This algorithm Cannot be implimented because Knowledge Knows the burst time of a new upcoming process.

Q SRTF

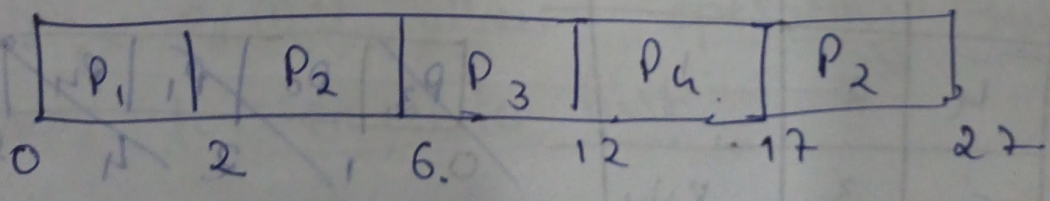
	A.T.	Execution Time	
A	0	3	8-0=8
B	3	2	5-3=2
C	5	4	12-5=7
D	7	6	21-7=14
E	10	3	15-10=5

A(TAT) = 7.2



Q SRTF

	A.T.	B.T.	T.A.T = C.T - A.T	W.T.
P_1	0	12	27-0 = 27	27-12 = 15
P_2	2	4	6-2 = 4	4-4 = 0
P_3	3	6	12-3 = 9	9-6 = 3
P_4	8	5	17-8 = 9	9-5 = 4



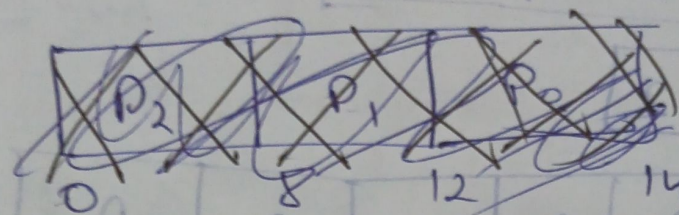
Average w.t = 5.5

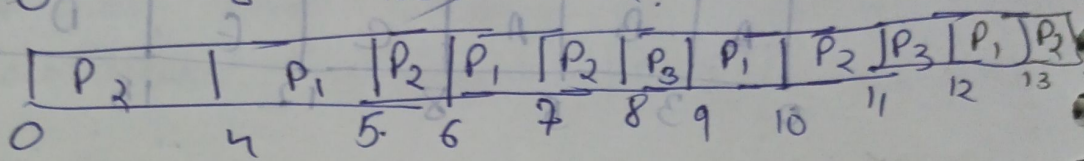
Q. Consider Three processes

	A.T	B.T	T.A.T	T.A.T
P_0	0	2	14-2=12	
P_1	0	4	12-4=8	
P_2	0	8	8-0=8	

Condition
① Longest Remaining time first

② with priority to who come first



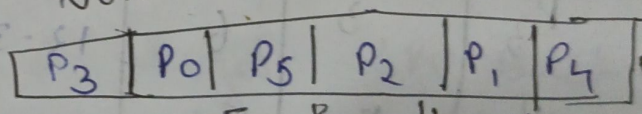


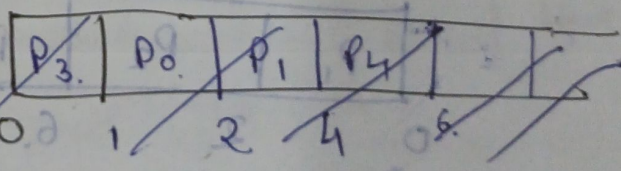
Priority algorithm :-

Q. Consider a table:

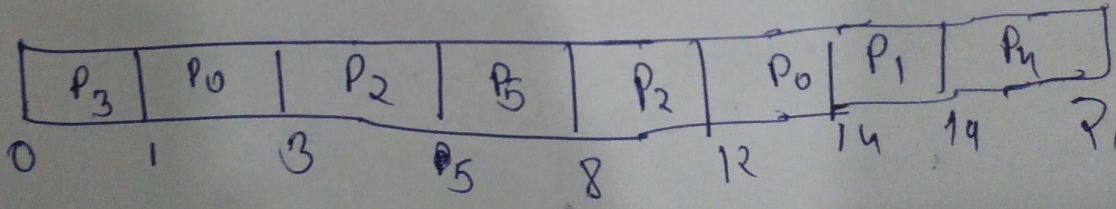
	A.T	B.T	Priority
P_0	1	4	5
P_1	2	5	2
P_2	3	6	6
P_3	0	1	4
P_4	4	2	1
P_5	5	3	8 high

NON - Primitive





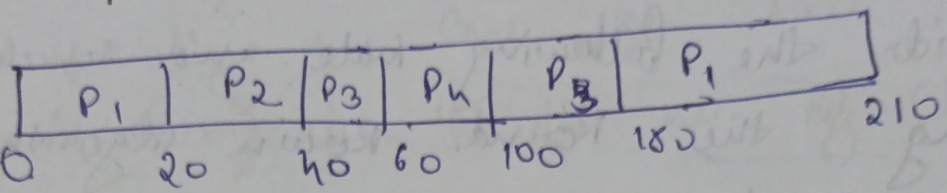
Primitive



Q

	A.T.	B.T.	
P_1	0	50 (30)	4
P_2	20	20	1 (1)
P_3	40	100 (80)	3
P_4	60	40	2

A(W.T) = 50



Priority algo :- ① The basic idea behind Priority algo is to support operating system process as sometime neither the burst nor the waiting time is important we want to execute the system time process.

② Priority can be used in both version Primitive as well as non-primitive.

③ Conclusion :-

Advantage :- ① It allows operating system Processes to execute first.

disadvantage :- ① Here the process with less priority will starve for the CPU.

Solution :- Aging Here after waiting for the fixed time quantum if the Process don't get the CPU then

its priority is increased by 1, and the Procedure is repeated until the Process gets the CPU.

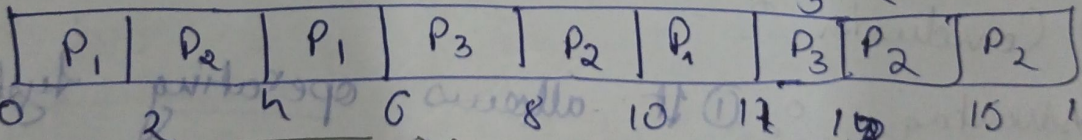
Round Robin :-

Q. Consider the following table and schedule it using the Round Robin algorithm Time Quanta to be fixed as 2 unit.

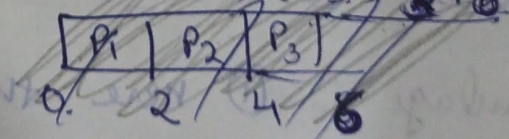
	A.T.	B.T
P_1	0	5
P_2	1	7
P_3	3	4

Gantt chart

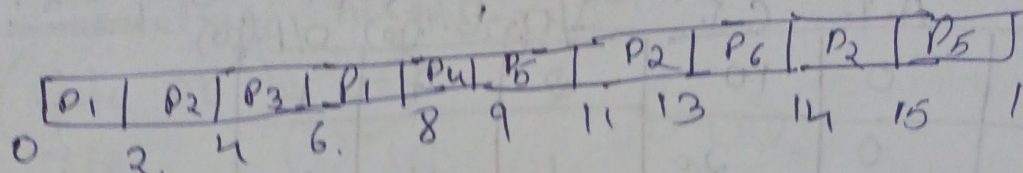
$P_1\ P_2\ P_1\ P_3\ P_2\ P_1\ P_3\ P_2$



	A.T.	B.T	
P_1	0	4	2
P_2	1	5	
P_3	2	2	
P_4	3	1	
P_5	4	6	
P_6	6	1	



$P_2 P_3 P_1 P_4 P_5$ P_2, P_6, P_5, P_2, P_5



Round Robin :- ① the Round Robin algorithm the idea is to provide a better response time specially in case of time sharing system (server which handles no. of clients)

② here we set a fixed time quantum and if a process require more then that then it is forcefully primited and must re-entred into the Queue

Advantage :- ① works well to give better Response time.

disadvantage :- ① It should not be used for background process where response time is not an issue.

NOTE :- ① the performan of RR depends on time quantum

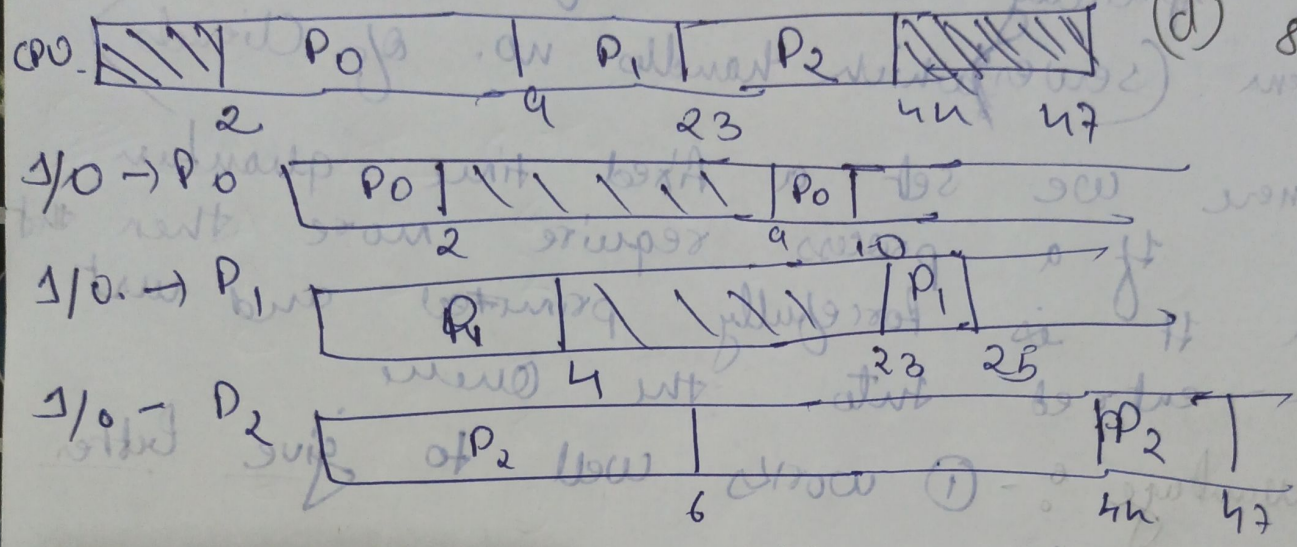
If it is very large then Response time will be poor If it is very less then a good amount of time is wasted in Context.

② Round Robin is always primitive

Q Consider 3 process AT = 0 | S.R.T.F

	A.T.	C.T.	I/O (2)	B.T (10)	O/P (10)
P_0	0	10	2	7	1
P_1	0	20	4	14	2
P_2	0	30	6	21	3

(a) 0%
(b) 10-6 ✓
(c) 30%
(d) 84.4%

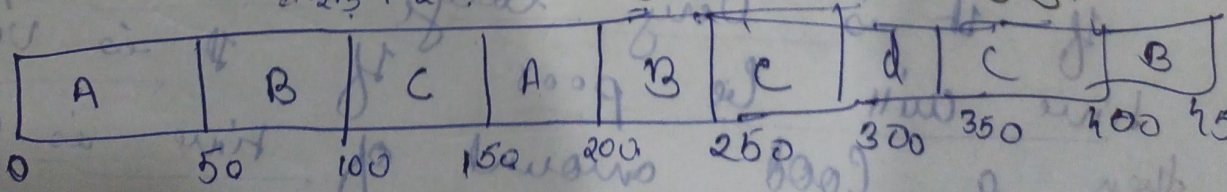


Q

	B.T.	I/O	A.T
A	100	500	0 ms
B	350	500	5 ms
C	200	500	10 ms

Round Robin = 50

$P_1 P_2 P_3 P_1 P_2 P_3$ — A B C A B C B C B C B.



C

450 500

Q. Consider a uni-processor scheduler, which is executing three task t_1, t_2, t_3, each (Gate PYQ-56)

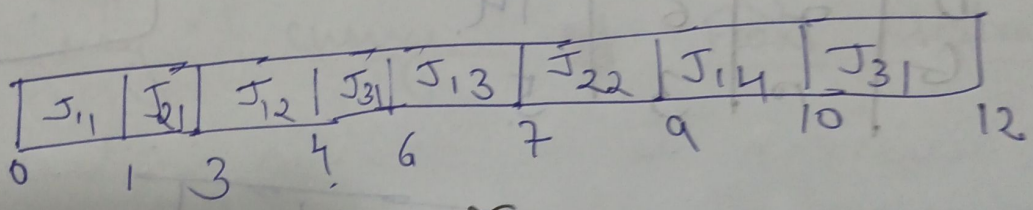
(3) (7) (2) 20 (4)

T_1 (1) 7 $J_{21}/0$ $J_{31}/0$

(1) $J_{11}/0$ $J_{22}/7$ (4) $J_{32}/20$

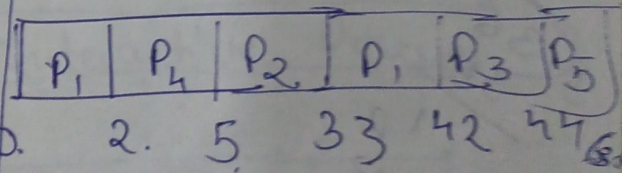
(1) J_{12} $J_{23}/14$

(1) J_{13} $J_{33}/21$

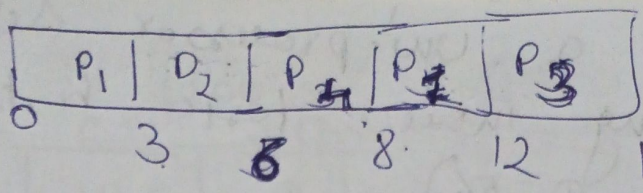


Q64 Gate Previous year

	A.T.	B.T.	Pr	C.T.	T.A.T.
P_1	0	11	2	42	31
P_2	5	28	0	28	0
P_3	12	2	3	32	30
P_4	2	10	1	3	
P_5	9	16	4	51	



Q3.



$P_1 = 4$.

$P_2 = 4$.

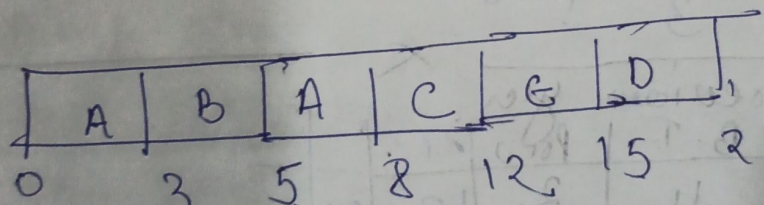
12 5 3
3 0 4) 12
12 7 12
2 0

Co. T. = 12

Q.

Pn.	AT.	C.T.	TAT.
A	0	6	8
B	3	2	2
C	5	4	7
D	7	6	14
G	10	3	5

7.2



$0 < \beta < \alpha$ → FCFS

$0 < \alpha < \beta$ → RR

Multilevel queue scheduling :- 1) Every algo. supports a different class of Process But in a Generalised system some Process want to be scheduled using priority algorithm. while some process want to Remain in system. (Interactive) process. while some one background process where execution can be delayed.

2.) In General Round Robin algorith with different time quantum is used for such maintainance.

NOTE :- The the Number of ready queue algo inside a queue & algo b/w the queue may change & From system to system.

Time quantum = 8 Priority → System Process →

Tq = 4 RR → Interactive Process

Tq = 2 FCFS → Background process

(2 marks) * Process Synchronisation *

The Idea of Process Synchronisation is to decide some ordering or have a schedule b/w the process so that shared resources can be used without a Conflict.

1. Race Condition :- P()
(
R(i)
i = i + 1
w(i)
)

	P_1	P_2
	10	10
	11	11
	11	11

Race Condition is a scenario where by changing the order of execution of Process we can change the output.

For Ex :- Here final value is 11. But if process execute one after another then it will be 12.

```
Pi()
{
 while(1)
   {
     Initial section
     entry section
     Critical section
```

Exit Section
Remainder Section
}
}

A process is said to be in Critical Section when it access shared Resources the Execution before the critical section is called as Initial section & After the Critical Section is called as Remainder section. It depends totally on the process whether it will never go in Critical Section or will go more than 1 time.

---

* Criteria For Solving Critical section problem:
① Mutual Exclusion :- It states that any at instance of time maximum 1 process can be in the Critical Section.

② Progress :- Only those process should compete to enter into Critical section which actually wants to enter.
e) Progress also states that the system should be deadlock Free and atleast 1 process should enter into Critical section.

These two criteria are Mandatory and if they are not Satisfied then the Solution

will be Considered as invalid.

③ **Bounded wait** :- By Bounded wait we means that there must be an upperbound after which a Process is guaranteed to Get the Resources. (in terms of number)

Q

```
P1()                    P2()
{                       {
  C=B-1 /I11              D=2×B  I21
  B=2×C  I12              B=D-1  I22
}
```

① I_{11} C=1/2
I_{12} B=2
I_{21} D=4
I_{22} B=3

② I_{21} D=4
I_{22} B=3
I_{11} C=2
I_{12} B=4

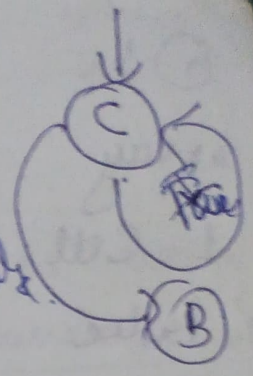
③ I_{11} C=1
I_{21} D=4
I_{12} B=3
I_{22} B=2

④ I_{21} D=4
I_{11} C=1
I_{12} B=2
I_{22} B=3

⑤ I_{11} C=1
I_{21} D=4
I_{12} B=2
I_{22} B=3

I_{21}
I_{11}
I_{12}
I_{13}

2 process solution :-



turn = 0/1

P_0	P_1

while(1)
{
initial section
while (turn != 0);
Critical section
turn = 1
remainder section
}
	while(1)
	{
	initial section
	while (turn != 1);
	Critical section
	turn = 0
	remainder section
	}

It is a two process Solution which uses a boolean variable turn It satisfies mutual Exclusion Condition but it fails on progress as it requires strict alternation.

Conclusion :- Solution is invalid.

Flag [F | F]
0 1

P_0	P_1

while(1)
{
initial section
Flag[0] = TRUE
while (Flag[1]);
Critical section
Flag[0] = F
remainder section
}
	while(1)
	{
	initial section
	Flag[1] = true
	while (Flag[0]);
	Critical section
	Flag[1] = F
	remainder section

(1) In this solution we use a Boolean array called flag where each process has 1 cell. where false means not interested and true means frustrated.

(2) This solution satisfies mutual exclusion but fails on progress because it suffers from deadlock.

Petersons Solution or Dekker's algorithm

Flag | F | F |
0 1

turn = 0/1

P_0	P_1
while(1) {	while(1) {
Initial section	Initial section
flag[0] = T	flag[1] = T
Set → turn = 1	turn = 0 ← Set (Context switch)
Test → while (flag[1] && turn == 1)	while (flag[0] && turn == 0) (Test)
Critical section	Critical section
flag[0] = F	flag[1] = F
remainder section }	remainder section }

This is a two process solution know as peterson's Solution or dekke's algorithm

here we have used two resources
1. array Flag :- which is used to tell interested or not interested

2. turn :- which is used to decide when both process are interested.

It is a valid solution which satisfies both mutual exclusion and progress -> (Bounded wait)

Q

P_1	P_2
while ($S_1 == S_2$)	while ($S_1 != S_2$)
C.S.	C.S.
$S_1 = S_2$;	$S_2 = not(S_1)$;

$S_1 = 0/1$

$S_2 = 0/1$

=> mutual Exclusion

Q

P_1	P_2
while (T)	while (T)
want1 = T	want2 = T
while (want2 == T)	while (want1 == T) ;
C.S	C.S
want1 = F	want2 = F
}	}

Ans = MG | DdG ✓

Q

Ans = mutual Exclusion

X	Y
while(T)	while(T)
varP = T	varQ = T
while(varQ == T)	while(varP == T)
{	{
C.S.	C.S.
varP = F	varQ = F
}	}

Conclusion :- Problems with Peterson's Solution :

1. It is only a two process solution and for a Generalised operating system we need n-process solution. It is extremely inefficient because it never Blocks a process and a lot of time is wasted in busy waiting.

Semaphore :-

```
int s = 1 ;

signal(s) / V(s)          |  wait(s) / P(s)
{                         |  {
   s = s + 1              |  while (s <= 0) ;
}                         |     s = s - 1
                          |  }
```

Solution :-

```
while(1)
{
   Initial section
   wait(s)
   C.S.
   signal(s)
   remainder section
}
```

semaphore are simple integer variables which are generally initialised with 1 for solving critical section problem.

② A part from initialisation they can be accessed only by two variables wait & signal.

③ Using semaphores we suggest a n-process solution

④ The solution is valid as it satisfies mutual exclusion and process but fails on bounded wait.

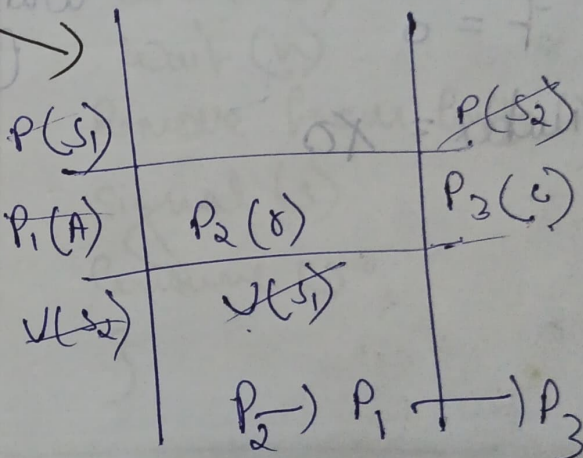
Applications of semaphore :-

① For solving C.S. problem :-

② For process ordering →

③ For Resource management :-

:- when we try to manage multiple copies for the same Resource type



P_1	P_2
$P(S_1)$	$P(S_2)$
$P(S_2)$	$P(S_1)$
C.S	C.S.
$V(S_2)$	$V(S_1)$
$V(S_1)$	$V(S_2)$

$S_1 = 1$ 0
$S_2 = 1$ 0

P_1	P_2
$P(S_1)$	$P(S_1)$
$P(S_2)$	$P(S_2)$
C.S.	C.S.
$V(S_2)$	$V(S_2)$
$V(S_1)$	$V(S_1)$

Deadlock :-

In order to avoid deadlock if multiple semaphore is one used before entering into critical section than the order must be same so that there is a head to head clash and 1 opponent is out from the system.

Bounded Buffer :-

e := n-1

f = 0

mutex = 1

P	C

while(1)
{
Produce();
wait(e)
wait(mutex)
append()
signal(mutex)
signal(f)
}
	while(1)
	{
	wait(f)
	wait(mutex)
	();
	signal(mutex);
	signal(e)
	Consume();
	}

Bounded Buffle Problem or Produce Consumer problem is a classical synchronisation problem where we have a Buffer with n-slots or Cell and there are 2 process produce and Consumer Can produce and Consume 1 article at a time Respectively.

Write a solution using semaphore to ensure Overflow Condition For Produce and

underflow for consumer and C.S. for buffer.

Solution :- we have used 3 semaphore 'e' take care of empty cell as well as underflow and f take care of filled cells as well as overflow and mutex is used for critical section.

Page-90
Q=30

n=20
s=1

P	C

while(1)
{
Produce();
wait(s);
add to buffer();
signal(s);
signal(n);
}
	while(1)
	{
	wait(s)
	wait(n)
	Remove from buffer();
	signal(s)
	Consume();
	}

∴ Reader writer Problem :- ① There is a shared piece of text and two types of Process are access this text Reader and writer.

② There is no clash b/w Reader and Reader Therefore when a Reader is inside the C.S. Then other Reader can make a entry but when a writter is inside a C.S then neither the Reader nor the

the writer gets the entry.

Write

```
while(1)
{
  wait(mutex)
  write
  signal(mutex)
}
```

mutex = 1 | Rc = 0

Read = X0

Read

```
while(1)
{
  wait(Read)
  RC = RC + 1
  if(RC == 1)
  {
    wait(mutex)
  }
  signal(read)
```

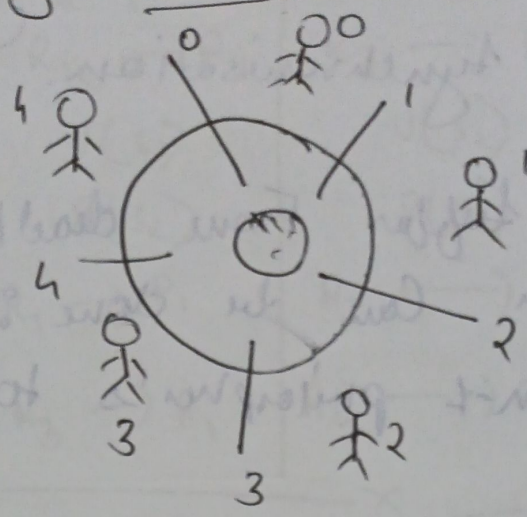
Read.

```
  wait(Read)
  RC = RC + 1
  if(R == 0)
  {
    signal(mutex)
  }
  signal(read)
```

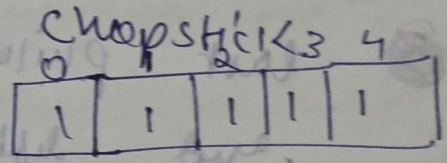
Solution:- ① Here in the Solution we have used 3 Resources i.e a Semaphore mutex for synchronisation b/w Reader, writer and writer, writer.

② while ReadCount is a simple integer variable which is given security by Read semaphore. (which works for synchronisation b/w Reader - Reader)

Dinning philosophes :-




```
Pi()
while(1)
{  think
   P(s)
   wait(chopps-stick[i])
   wait(chop-stick[(i+1)%5])
   V(s)
   eat
   signal(chop-stick[i])
   signal(chopstick[(i+1)%5])
   think
}
```



$S = 1_0$

① In dinning philosophe problem there is a circular table and odd number of philosophes are sitting on the table.

② There is a chop-stick placed b/w every philosopher.

③ philosophe performs only two process either they think or eat (using 2 chopstic

write a Solution using semaphores for Philosphers synchronisation.

Solution suffer from deadlock and following modification can be done :-

1) allow n-1 philospher's to sit on the table.

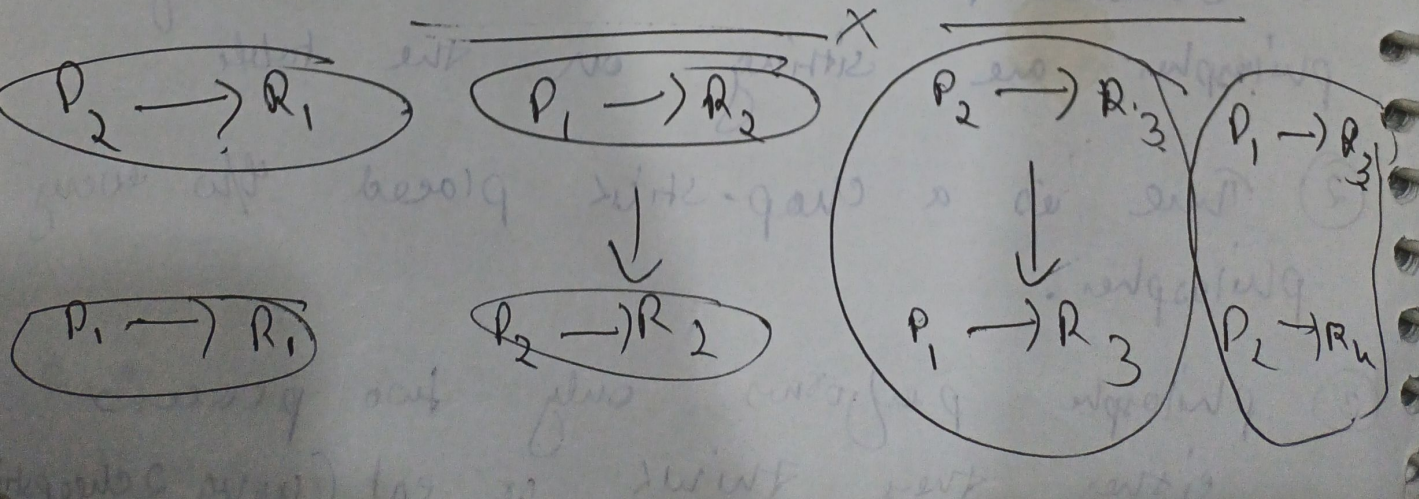
2) allow n+1 chop-stick to put on table

3) let n-1 philospher's pick the left chop-stick afirst and Right.

4) last philosph pick the Right chop-stick and left or vice-versa.

5) division can be done b/w odd and even number philospher's.

6) take 1 more semaphore and Consider 2-wait as a Critical section.



R → R (Resource Allocation)
P → R

$R_2 \rightarrow P_1$
$R_n \rightarrow P_1$
V(S1)
P(S2)
$R_1 \rightarrow P_1$
$R_3 \rightarrow P_1$

$R_1 \rightarrow P_2$
$R_3 \rightarrow P_2$
V(S2)
P(S1)
$R_2 \rightarrow P_2$
$R_n \rightarrow P_2$

$S_1 = 0$
$S_2 = 0$

Ans = 2 semaphore.

Q 32
Page = 40.

X	Y
Private i	Private i
for(i=0; i<n; i++)	for(i=0; i<n; i++)
{	
a[i] = f(i)	Entry Y(R,3)
Exit (R,S)	S(i) = o(a[i])
}	}

Conclusion :- Disadvantage of Semaphore.

1) Semaphores are extremely inefficient as a lot of CPU clockcycles are wasted in Busy waiting.

2) Semaphores are also do not ensure bounded wait therefore we do-not follow First Come First Serve

Improved Semaphores :-

```
P1(S)
{
  S = S-1
  if (S<0)
  {
    block();
    add it into queue();
  }
}
```

```
V(S)
{
  S = S+1
  if (S<1)
  {
    Dequeue();
    resume();
  }
}
```

Here there will be two improvement.

① the implimentation will become efficient as there is no wastage of CPU clock cycle.

② Bounded wait will also be ensured as we strictly follow FCFS.

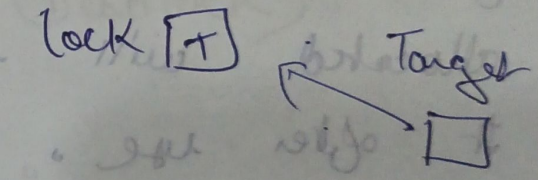
Hardware Solutions For Critical Section problem.

```
while(1)
{ P1()
  {
    Initial Section
    Disable Interrupt
    C.S.
    enable Interrupt
    Remainder Section
  }
```

Disable Interrupt is a privileged Command used by very very important o.s. processes where a process enters into critical section it simply disables the enterrupt and there no chance of mutual exclusion voilation because no-body else get the CPU

```
Boolean Tas (boolean *Target)
{
  boolean rv = *target
  *target = True
  return rv
}
```

```
while(1)
{
while (Tas (lock));
  C.S.
lock = F
}
```

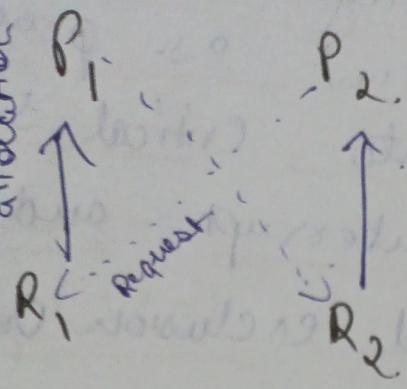


#.

Test and set is a Idea where first we set then we test and it is implimented with the help of special hardware where the processor can complete entire test and set in single clock cycle.

Ex:-

Deadlock



In a multiprogramming O.S there are a number of processes which fight for finite no. of Resources and some times a waiting process never get a chance to change its state because the Resource for which it is waiting are held by other waiting process.

NOTE:- Here every process will follow system model which means process Request a Resource if not allocated then wait otherwise if allocated will use the Resource and Release it after use.

4 Necessary Conditions For the existence of deadlock :-

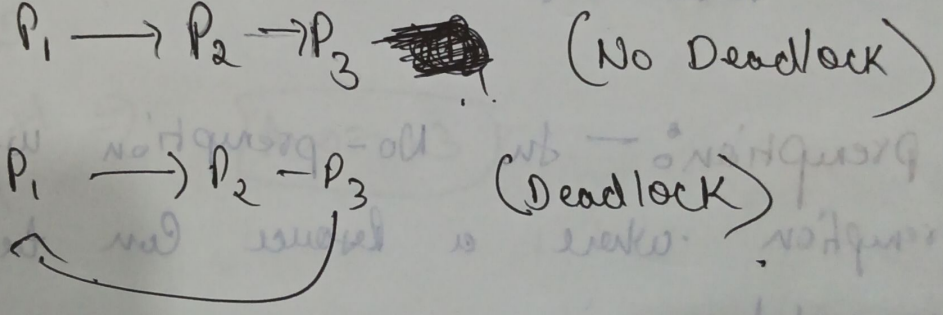
1. Mutual Exclusion:- By Mutual exclusion we mean there must be atleast 1 Resource in the system which is used by 1 process at a time and it is desired by more than 1 processes.

Ex:- printer

② Hold & wait :- Hold and wait says a Process 1st will aquire and hold the resource available then only it will wait for other resources.

③ No-preemption :- In No-preemption No process is allowed to forcefully prempt the Resource acquired by the other process.

④ Circular wait :- Circular wait Condition specifies that the wait must occur in a Circular fashion.



Deadlock Handling method :-

Prevention :- Prevention means that we design such a system where there is no chance of having a deadlock.

Mutual Exclusion :- It cannot be resolved as it is the hardware property.

Example :- printer.

(2) Hold & wait :- (1) Hold & wait can be resolved using Conservative approach where a process can start if and only if it has acquired all the resources.

(2) active approach :- Here process acquire only required Resources but when ever a new resource is required it must first Release all the acquired Resources.

(3) wait time out :- Here there is a maximum time bound upto which a process can wait for other Resources after which it must Release the Resources.

(3)
No preemption :- In No-preemption we allow forceful preemption where a Resource can be forcefully preempted.

priority must be given to the process which is in waiting state.

(4) Circular wait :- In order to remove Circular wait we assign a number to every resource and the process can request only in the increasing order otherwise, the process must release all the high number acquired Resources and then make a fresh Request.

Conclusion :- those system will use preemption method where the Risk of having a deadlock is either more several or more frequent.

Example :- Realtime system.

Deadlock

Q1 Page = 93

Resources = 6.

$P_1 = 2$

$P_2 = 2$

Ans = 2 Process

Q2

A - 3 ?

B - 4 3

C - 5 5

10 + 1 ⇒ Ans = 13

= 11

Q3

proces = 3

$P_1 = 2, 1$

$P_2 = 2, 1$

$P_3 = 2, 1$

Ans = 4.

Q4

P = 1

P = 1

P = 1

P = 1

P = 1

Ans 2 5

Q17

$P_1 = 3 \quad 2$

$P_2 = 3 \quad 2$

$P_3 = 3 \quad 2$

$= 6 + 1 = \boxed{7}$

Q18

Ans =

$P_1 = 1$

$P_2 = 1$

$P_3 = 1$

$\vdots$

$P_n = 1$

P_5

P_6

Ans = 4

1 NP ——— × ——— $S_1 \; S_2 \; \cdots \; S_i$

Process

$P_g = 93 \longrightarrow P_1, P_2, P_3 \cdots P_1 \cdots P_n$

Q=7

Resource $\longrightarrow R_1, R_2 \cdots R_3$ $\quad S_i > 0$

$(S_1 + 1) + (S_2 - 1) + \cdots (S_n + 1)$

$S_1 + S_2 \cdots \cdots S_n - n$

$$\sum_{i=1}^{n} S_i - n < M$$

$$\sum_{i=1}^{n} S_i < M + n$$

NOTE:- prevention guarantees independence from deadlock but the cause of implementing the Prevention method is very high solution is Avoidance.

Q. Consider 3 process P_1, P_2, P_3

	Max Demand				Current Alloc				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	3	3	2	2	1	2	2	1	2	1	0	1
P_2	1	2	3	4	1	0	3	3	0	2	0	1
P_3	1	3	5	0	1	2	1	0	0	1	4	0

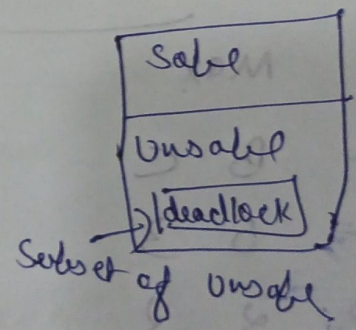
Total Resources			
6	5	7	6

Need = Max demand - Allocation

~~Available Resources = Current Allocation~~

Available Resources = Total Resources - Current Allocation

Safe state $P_1 \rightarrow P_2 - P_3$



Page = 95

	Allocation			Max			Need			Available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P_0	0	0	1	0	4	3	0	4	2	3	2	2
P_1	3	2	0	6	2	0	3	0	0			
P_2	2	1	1	5	3	3	3	2	2			

R_1 : P_0

X	Y	Z
0	0	2

R_2 : P_1

X	Y	Z
2	0	0

X	Y	Z	
3	2	2	
3	2	0	(P_1)
6	4	2	
+ 2	1	1	P(2)
8	5	3	
x 8	4	4	

Q20

Poor	Current	Max	Need
P_1	3	7	4
P_2	1	6	5
P_3	3	5	2

9
7
2

Available
2

$P_3 \rightarrow P_2 \rightarrow P_1$

Q

	Max			allocat:			Need		
	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3
P_1	3	2	2	2	0	0	1	2	2 + 2 2 4
P_2	9	0	2	3	0	2	6	0	0
P_3	2	2	2	2	1	1	0	1	1
P_4	4	3	3	0	0	2	4	3	1

Available

A	B	C
3	3	2

safe state

$P_1 \rightarrow P_3 \rightarrow P(0) \rightarrow P_2 \rightarrow P_4$

Request.

	A	B	C
P_1	1	0	2

↓

Safe

3 3 2
+ 2 0 0 (P_1)
5 3 2
2 1 1 P (P_3)
7 4 3
+ 0 1 0
7 5 3 P(6)
3 0 2
1 0 5 5 P(2)
0 0 2
1 0 5 7

Q.14 Page :- 95

P_1	P_2	P_3
t_0: $R_2 \leftarrow 2$	t_0: $R_3 \leftarrow 2$	t_0: $R_1 \leftarrow 1$
t_1: $R_3 \leftarrow 1$	t_2: $R_1 \leftarrow 1$	t_2: $R_1 \leftarrow 2$
t_3: $R_1 \leftarrow 2$	t_4: $R_1 \leftarrow 1$	t_5: $R_1 \rightarrow R$
t_5: $R_2 \rightarrow 1, R_1 \rightarrow 1$	t_6: $R_3 \leftarrow 1$	t_7: $R_2 \leftarrow 1$
t_7: $R_3 \rightarrow 1$	t_8: finish	t_8: $R_3 \leftarrow 1$
t_8: $R_1 \rightarrow 2$		t_4: finish.
t_{10}: finish		

Deadlock Avoidance: ① Here whenever a Process enters into the system it must declare maximum demand possible.

② At Runtime we maintain some datastructure like Current Allocation, Current need, Current Available etc.

③ when ever a process Request some Resources we first check whether system is in SafeState or not. If every process Require maximum Resources then is there any sequence Request in which Request Can be Entertain. If Yes then Request is allowed otherwise Rejected.

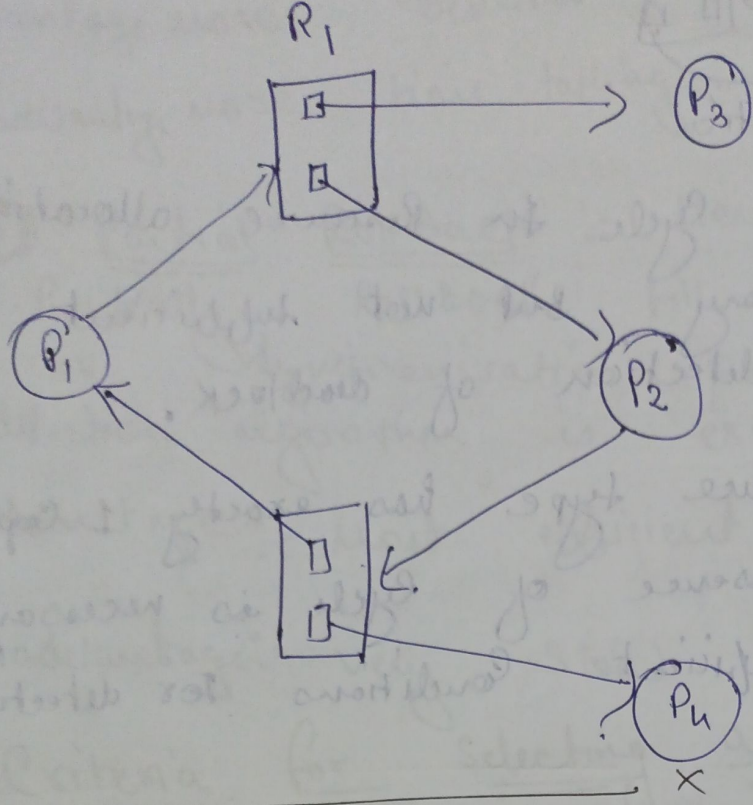
#Disadvantage of Banker's or Avoidance

① It Require Every process to declare maximum demand which is not possible most of the time.

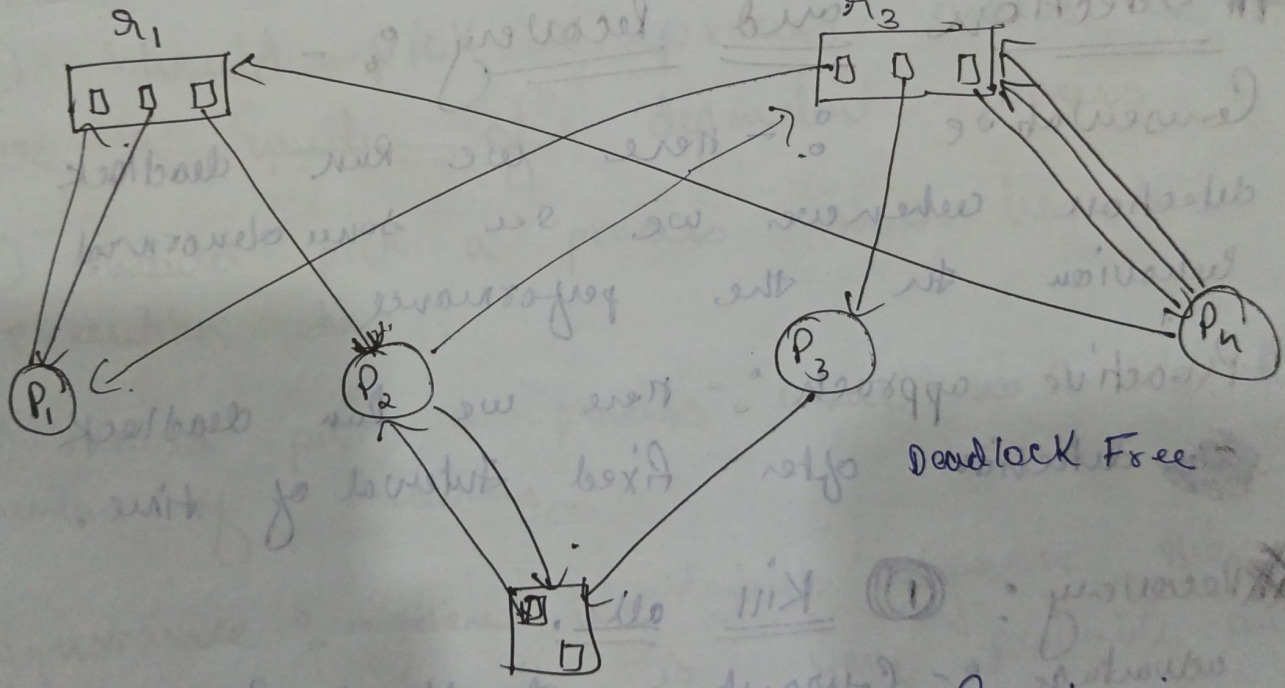
② It also assumes that the number of process in the system are Constant which is not true.

③ Banker's algorithm performs worst Case analysis which is very rare in practical time So even if it says Unsafe then deadlock rarely occurs.

Resource Allocation Graph :-

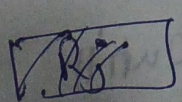


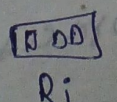
No Deadlock



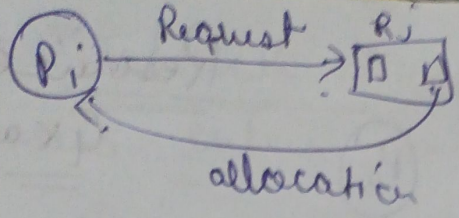
Deadlock Free

Resource Allocation Graph :- This Graph is also kind of Graphical Bankar's algorithm where a Process is denoted by a Circle (P_i) and Resource is denoted by a





. Dots inside the Resource represent the Copies.



① Presence of a Cycle in Resource allocation Graph is necessary but not sufficient Condition for detection of deadlock.

② If every Resource type has exactly 1 copy then the presence of Cycle is necessary as well as sufficient Conditions for detection of deadlock.

Detection and Recovery :-

Conservative :- Here we Run deadlock detection whenever we see some abnormal behavior in the performance

Proactive approach :- Here we Run deadlock detection after fixed Interval of time.

Recovery :- ① Kill all.

advantage :- Guarantees deadlock Removal

disadvantage :- less efficient or lot of workless

② Kill one by one :- Here we Kill one process at a time and then Run detection algorithm and will Repeat the process Until deadlock is Resolved

advantage more efficient Comparatively.

disadvantage more time taking.

③ Partial Roll Back :- Here a process is Partially Roll Back till the last check point or Synchronisation point and then detection algorithm is executed.

advantage :- most efficient.

disadvantage :- very slow

Criteria for selecting a victim :-

① should pick a process which do not have a number of dependent process.

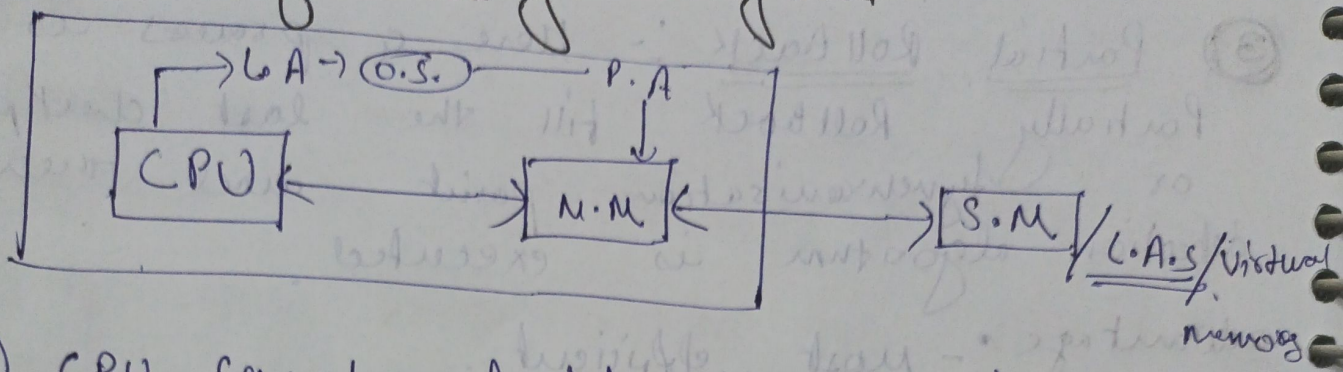
② should pick a process which has less execution time.

③ should pick a process where it has less number of Resources.

Ignorance :- Here we just ignore the problem as it is very probable and less severe.

UNIT ⇒ 3 Memory Management

Basics of Memory management



① CPU generates logical Address which is used to access secondary memory or logical Address Space.

② For access main memory or P.A.S. we need physical address. Therefore here O.S. has 2 major Responsibilities (A) Address Translation That is converting logical address into physical Address.

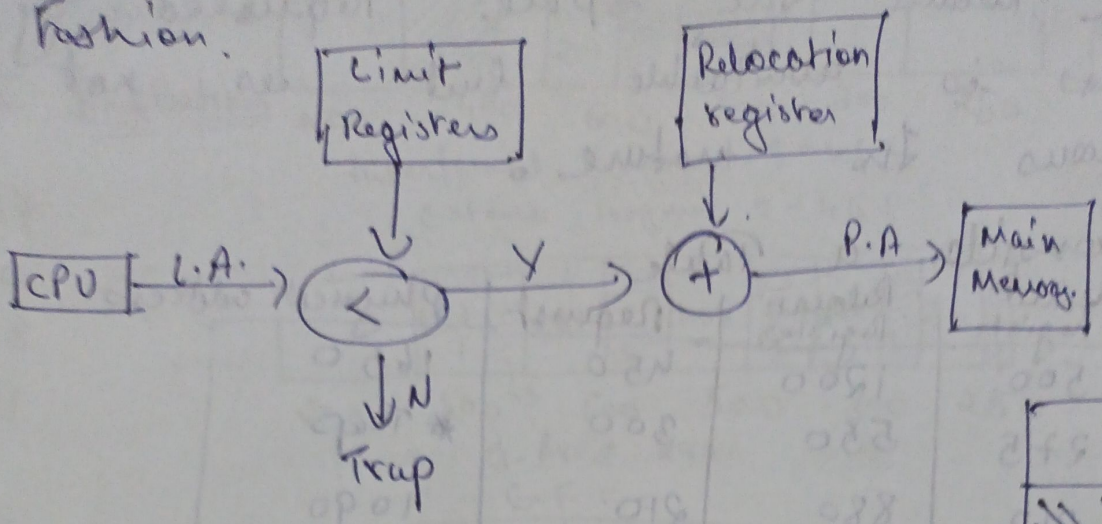
(B) Space allocation :- deciding which part of main memory will be allocated to whom.

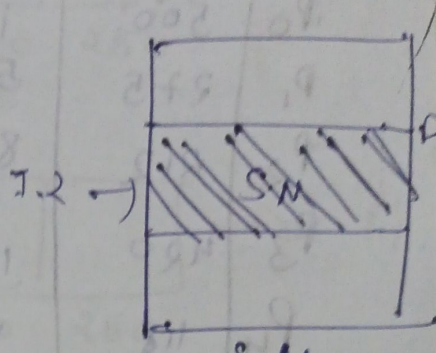
③ There are 2 methods For managing main Memory (A) Continuous (B) NON-Contigeous

(A) Contigeous Allocation :-

The Basic idea of Contigeous memory allocation is the Entire process will be loaded From secondary memory to main-memory and will be Stored in a Contigeous

Fashion.





Address Translation :- ⑪ In Contigeous Allocation address Translation is very simple like in the case of an Arrays.

Step :- ① A logical Address is Compared with a limit register to check whether Request is in limit or not if not then it will be trapped. otherwise ② it add the value of Relocation register to Get physical address

Advantage :- ① easy to understand, Read, write and Debug

② Extremely fast. Single memory access is sufficient.

disadvantage :- suffers From External Fragmentation.

External Fragmentation :-

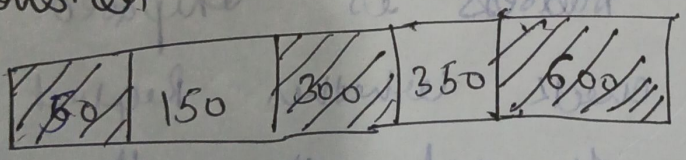
which means the space Required by the process is available but is not Contigeous in nature.

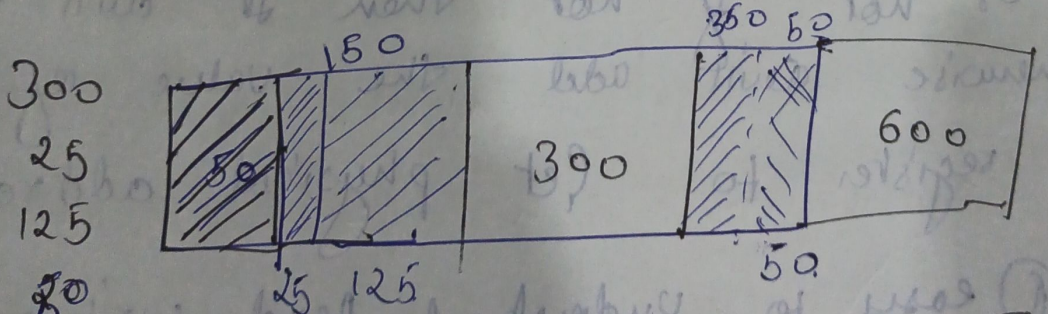
Q. Consider a Table

	Limit Register	Relocation Register	Request	Physical address
P_0	500	1200	450	1650
P_1	275	550	300	* Trap
P_2	212	880	210	1090
P_3	420	1400	450	* Trap
P_4	118	900	80	280

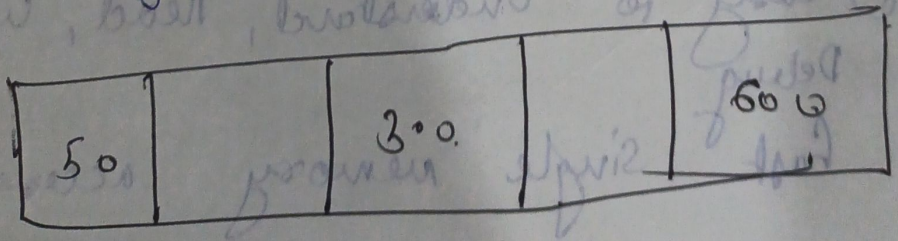
Space Allocation :-

Q. Consider





First Fit

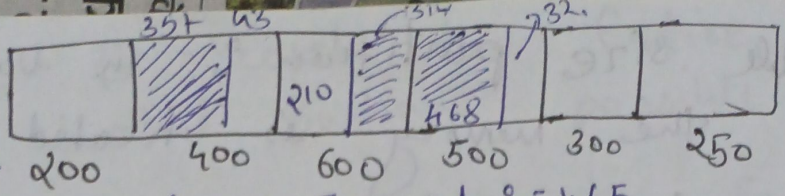


Best Fit

Q 02 H

Page no:- 151

Fixed Partition

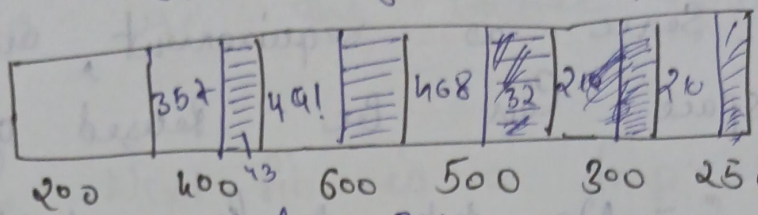


First Fit

Internal Fragment :- 465

External Fragment :- 441

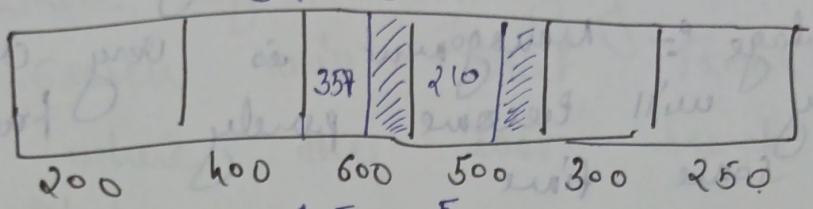
357
210
468
441



Best Fit.

I.Fr = 244

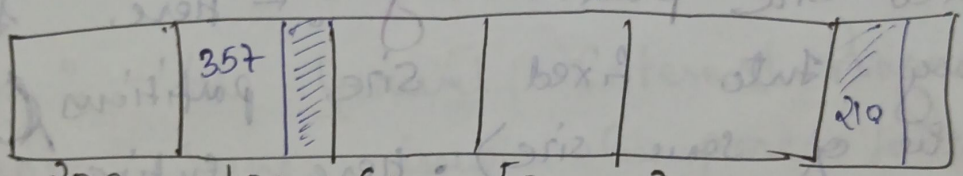
E.F = 0



Worst Fit.

I.F. 5

E.Fr = 959



First Fit.

Variable Partition.

Proportional frame allocation scheme :-

Total number of frames = M = 62.

S_i is the Memory Require of Its process =

$S_i = 10$

$S_1 = 127$

$S = S_1 + S_2$

$S = 137$

Formula =) $\frac{S_i}{S} \times M$ =

$\frac{10}{137} \times 62 = 4$ Frames

$\frac{127}{137} \times 62 = 57$ Frames.

Variable size partition :- In Variable size the memory is treated as 1 unit and the space allocated to a process is exactly same as requirement, and the left over space can be reused again.

Advantage :- No internal fragmentation.

disadvantage :- Management is very difficult :- Memory will become purely fragmented after some time

Fixed size partitioning :- Here system divides memory into fixed size partitions (may or may not be of same size). Here entire partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

Advantage :- Managment or Book keeping is easy.

disadvantage :- Internal fragmentation.

In Both policies there are 3 different space allocation method.

1) ~~Best~~ first fit :- It uses the first memory cell available.

Best fit :- It search the Entire memory and use smallest partion possible.

Worst fit :- It search the Entire memory and use the largest partition possible.

Next Fit :- Next fit is a improvement of first fit where after satisfying a request the next search will start from the next point where the last search or request was complited.

Conclusion :- In General External Fragmentation is a more Severe issue Compare to Internal Fragmentation therefore in order to remove External Fragmentation we must Go for non-Contigeous Space allocation policy.

Q optimal Page size :-

optimal Page size $= \sqrt{2 \cdot S \cdot e}$

where S = process size $= 128\,KB$

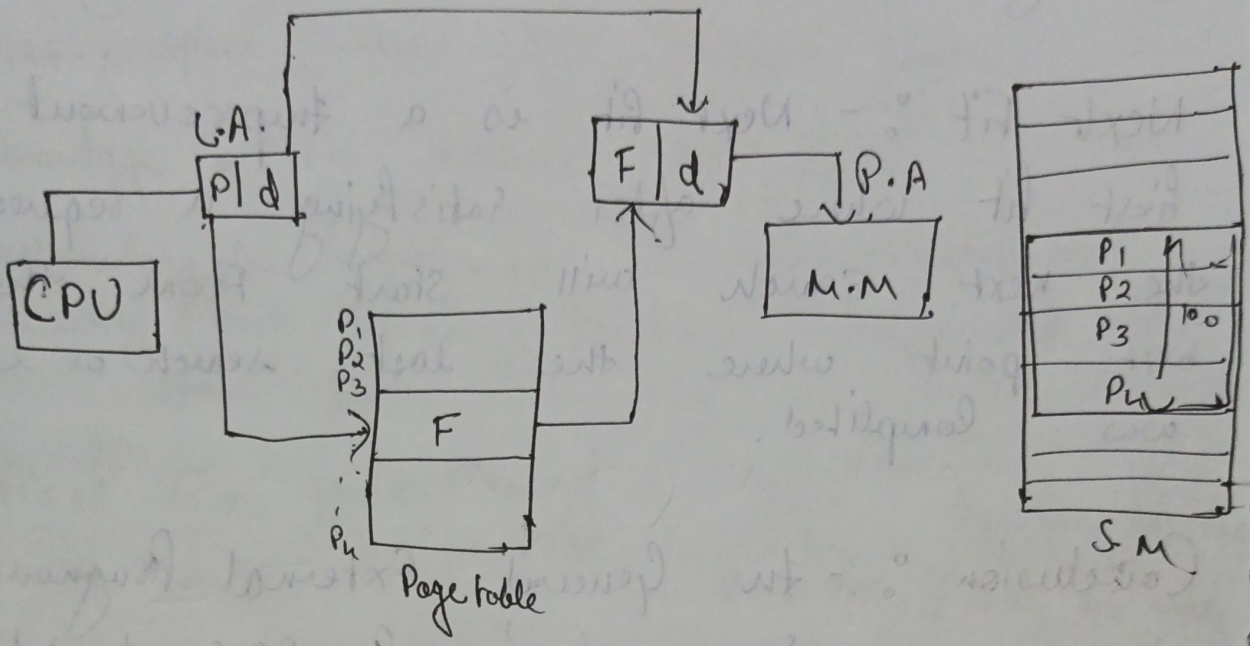
e = Page table entry $= 8\,Byte$

$= \sqrt{2 \times 2^{10} \times 2^{7} \times 8}$

$2^{10}\sqrt{2}$

Paging

(B) **Non-Contigeous** policy with fixed size partitioning is called paging.



① Secondary memory is divided into equal size partition (Fixed) called Pages.

② Main Memory is divided into equal size partition called Frames.

③ Size of Frame is equal to size of Page.

④ Every process will have a seperate Page table.

⑤ The Entries in the Page table is the number of Pages a Process have.

⑥ At Each Entry Given we have a Invalid Pointer which means the Page is not in main memory or we will Get the Corresponding Frame number.

when the Frame number is Combined with the instruction 'd' then we will get the corresponding physical address.

⑧ the size of Page table is Generally very large so Cannot be accomodated inside the PCB therefore PCB Contains a Register PTBR (Page table Base Register). which leads to the Page table.

Advantage :- Independance From external Fragmentation.

disadvantage :- ① It makes the translation very slow as main-memory is accessed 2 times.

② Page table is limited over the system which occupies the Considerable space.

③ A lot of space is wasted in internal Fragmentation.

④ Difficult to impliment and Understand.

$2^1 \rightarrow 2$
$2^2 \Rightarrow 4$
$2^3 \rightarrow 8$
$2^4 \rightarrow 16$
$2^5 \Rightarrow 32$
$2^6 \rightarrow 64$
$2^7 \rightarrow 128$
$2^8 \rightarrow 256$

$2^9 \rightarrow 512$
$2^{10} \rightarrow K$
$2^{20} \rightarrow M$
$2^{30} \rightarrow G$
$2^{40} \Rightarrow T$

Q Consider a system where main memory is 32 MB & the length of logical address is 32 bits. If size of each Page is 1 KB find everything ?

MM = 32 MB $2^{20} \times 2^5 = 2^{25}$

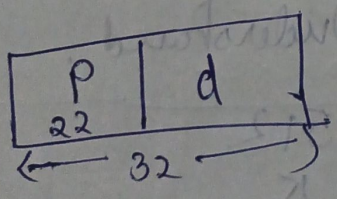
P.A ⇒ 25 bits

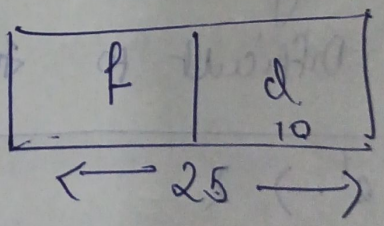
L.A ⇒ 32 bits

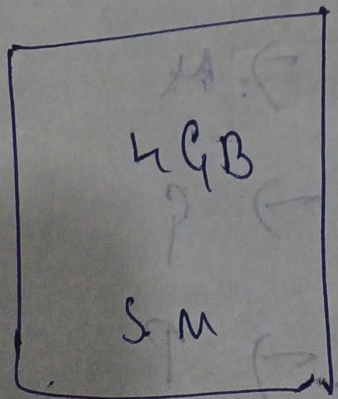
If address is of n bit then memory size is

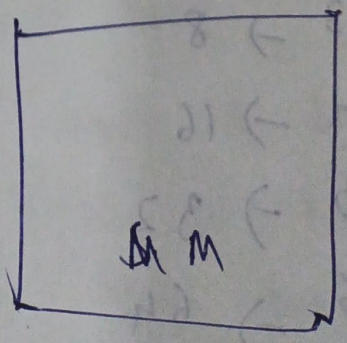
n ⇒ 2^n x size of each location (B)

$\lceil \log_2 \left(\frac{x\text{Byte}}{\text{size of each location}} \right) \rceil$





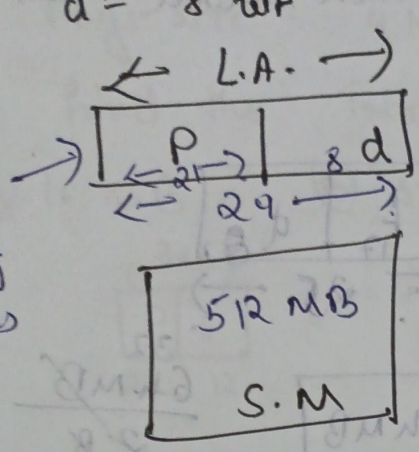




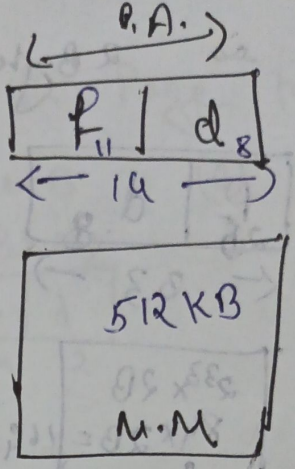
Q L.A. = 29 bit

MM = 512 KB

d = 8 bit

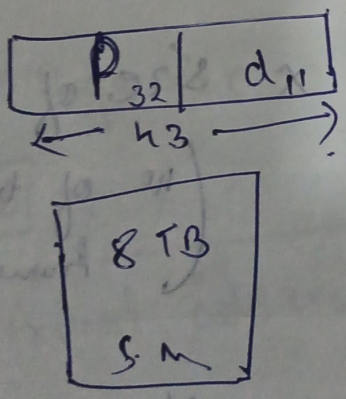


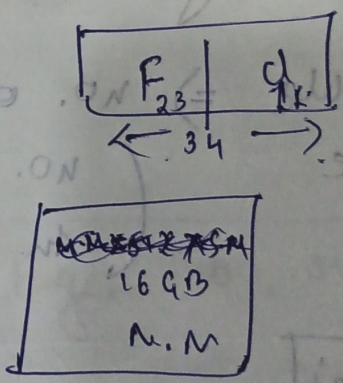
Total no. of Pages



512 KB / 1 B

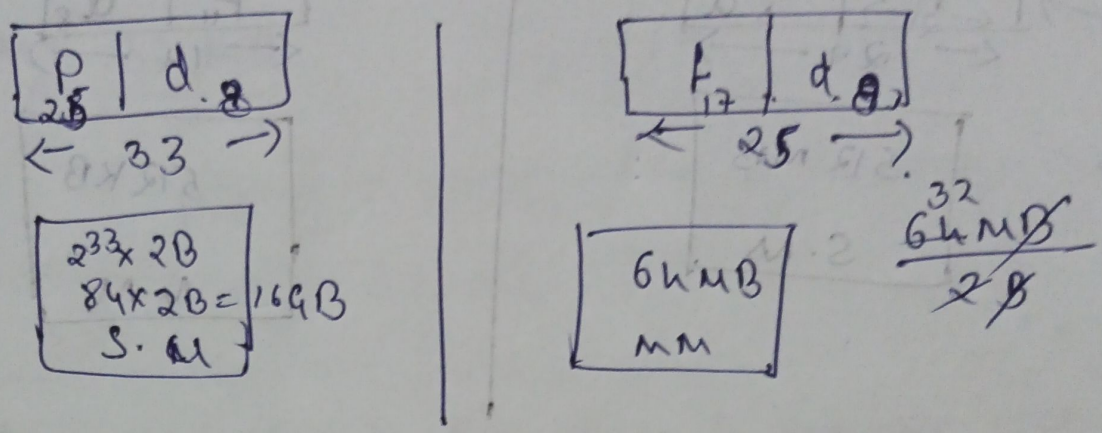
Q Consider a system where secondary memory is 8 Tera Byte and the main M = 512 times smaller than secondary memory if each Page is of 20K8 Byte find.





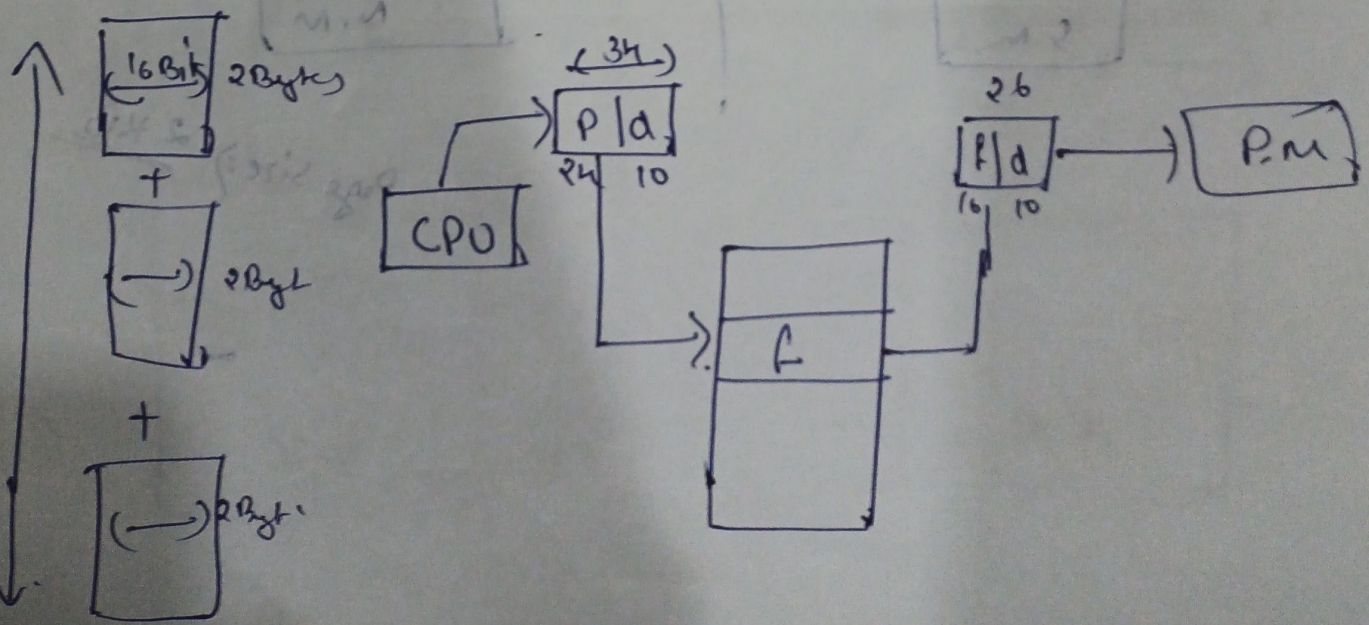
Page size? 2 KB

Q. In a system main memory is of 64 MB each Page is of 512 Bytes and the system contains 256 Pages on each frame and system is 2 Byte addressible.



Q. Consider a main memory 64 MB each Page 1KB and the secondary memory is 16 GB find the Total space wasted in maintaining the Page Table ?

Page table size = (no. of location in system) × (size of location = no. of bits in frame no.)



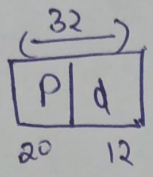
Q M.M = 64 MB

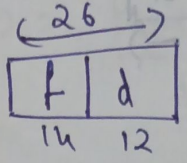
V.A => 32 Bit

Page size = 4KB

Find the Total wastage in Page Table

(a) 16 MB (b) 8MB (c) 2MB (d) 24MB





2^{20} 1wb = 2B

1M x 2B

=> 2MB

Booklet 58

Q. C.A => 32 bit

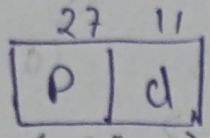
Page size = 4KB

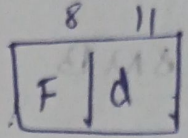
Ans- 4MB

Virtual address space = Page Table size

Page Table size = no. of frame # other Information (protection bit, modified etc)

Q Consider a system where Secondary memory is 256 GB. each Page is 2KB If MM = 512 KB find the space wasted in Page Table because of Process of size 16 MB.





$$\frac{\text{Process Size}}{\text{Page Size}} = \text{number of Pages}$$

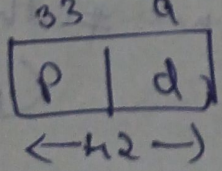
$$\frac{16\,MB}{2\,KB} \Rightarrow 8K$$

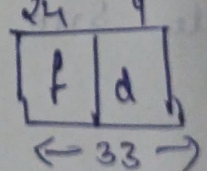
Q Consi MM = 8 GB, SM = 4 TB If each Page is of 512 Byte

1) If Page table of a Process is = 42 MB find the Size of Process

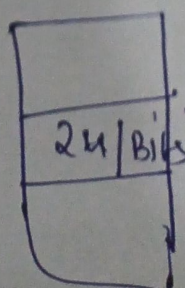
2) If the Size of Process is 64 MB then find the Size of its Page table

Ans = (2)





Process $\Rightarrow \frac{72\,B}{2^9\,Byte}$



144 K x 24 Bit

Q How to find Page table Size From Process size ?

Step ① Page table size = Size of each entry (F) × number of entries (number of Pages Process have)

Step ②. Number of Pages = $\frac{\text{Process size}}{\text{Page size}}$

TLB Topic

Q Consider a system where MM access time is 200 ns find the average instruction access time assuming that system support

① If a system supports a TLB Buffer with an access time of 10 ns and a hit Ratio of 90%. find the improved instruction access time ?

200 ns

AT AT ⇒ 2 MM

2 × 20 ns

⇒ 400 ns

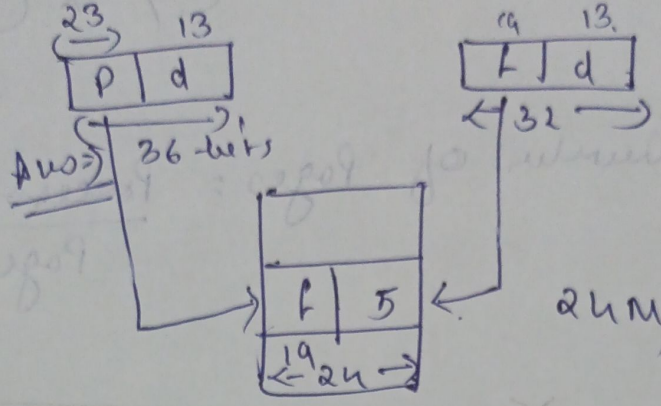
$= 0.9[10+200] + (1-0.9)[10+200]$

= hit ratio of TLB [TLB Time + M.M] + Miss ratio [TLB time + 2.MM]

Booklet Q.72 Page size = 8 KB

Page = 151 MM ⇒ 32 bit

Page table size ⇒ 24 MB



2^{19}

24MB = P x 3B

P = 8M

How to improve Memory Access time with Paging ?

Q MM = 400 NS

TLB access time ⇒ 100 NS

Hit Ratio 80%

find the percentage improvement with parallel TLB access compared to sequential

(A) Sequential.

$0.8[100 + 400] + (0.2)[100 + 2 \times 400]$

$0.8\ 500 + 0.2 \times 900$

$400 + 180$

⇒ 580

(B) Parallel

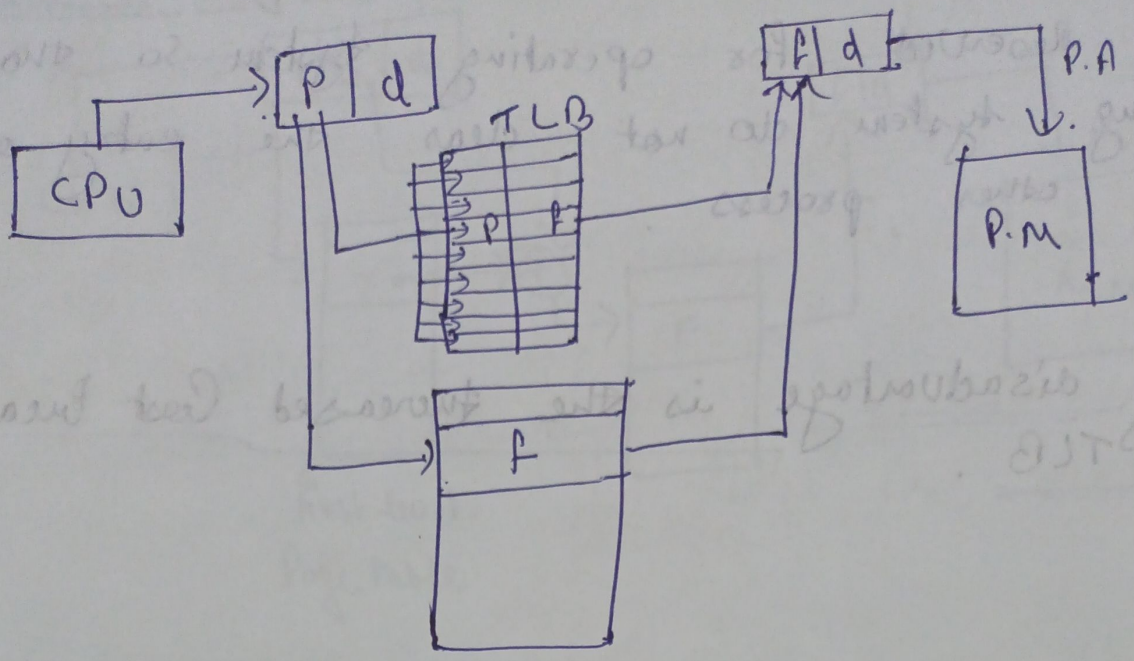
$0.8[100 + 400] + 0.2[800]$

$0.8 \times 500 + 160$

$400 + 160 \Rightarrow 560$

$\frac{20}{580} \times 100$

⇒ 3.48 %



The major disadvantage of Paging is :-

If main memory is access 2 times in order access any instruction because of which average instruction access time will be 2x main memory Access time.

Solution :- Translation look aside buffers.

① It is a specialed buffers which Contain two Colum first Contain Page number and second Corresponding Frame number.

Conclusion :- Because of locality of Reference we assume hit ratio of TLB will be more than 90% and Hence time will be again close to main memory access time.

Further Improvements :- If we have multiple Copies of TLB (set associative fashion) then the Current data of more than 1 process Can be handled in TLB.

3) wired down entries :- Some entries in TLB are Reserved for operating system so that operating system do not clear the entry of some other process.

Imp

Only disadvantage is the increased Cost because of TLB.

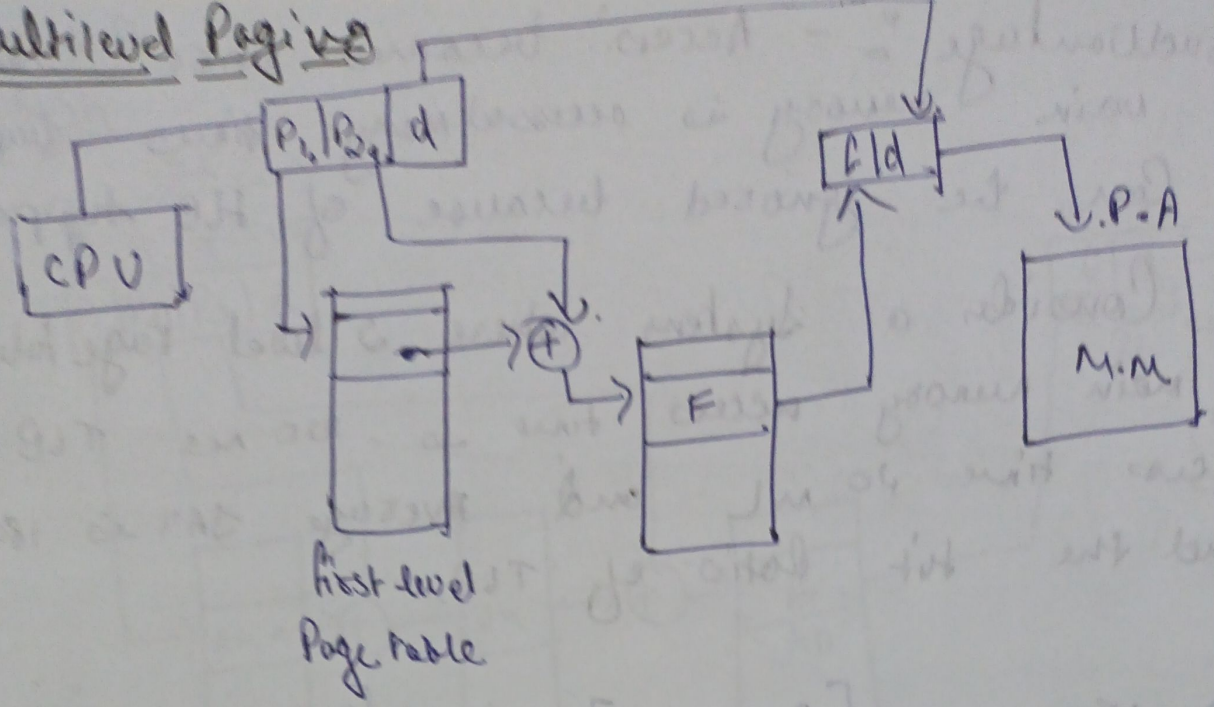
(2)

when we increase the Page size then the number of Pages in a Process decrease. Because of which the number of entries in pagetable also decrease.

Conclusion :- less wastage because of Page table.

② But in General we assume for every Process the last Page suffers of internal fragmentation on an average half Page and when we increase the Page size ~~xxxxxx~~ of wastage Because of internal fragmentation increases. Therefore we must choose that value or size of Page when Both wastage are optimal.

Multilevel Paging



Q.

Some times the size of Pagetable is so large that it cannot fit in a frame then we divide the Page table into equal size of Partitions and design 1 more Page table that points to these Partitions. the entire idea is called multilevel Paging.

① Based on Requirement this can further be expended by using 3 level Paging is a normal scenario in the industry.

Advantage :- ① main memory utilisation is more efficient as it is not required to have a entire Page in main memory.

② here access is more logical as it is not required to traverse the entire Pagetable

Disadvantage :- Access becomes more slow as main memory is accessed many times but it can be ignored because of TLB support.

Q. Consider a system where 2 level Page table if main memory access time is 100 ns TLB access time 20 ns and Average TAT is 180 ns find the hit Ratio of TLB.

- $180 = x[20 + 100] + 1 - x[20 + 3 \cdot 100]$

Booklet P.g :- 147

Q = 38

TLB time, cache hit time, cache miss time.

$.96\,[1ns + .9\,[1ns] + .1\,[1ns + 10ns]]$

TLB hit time, cache miss, main memory.

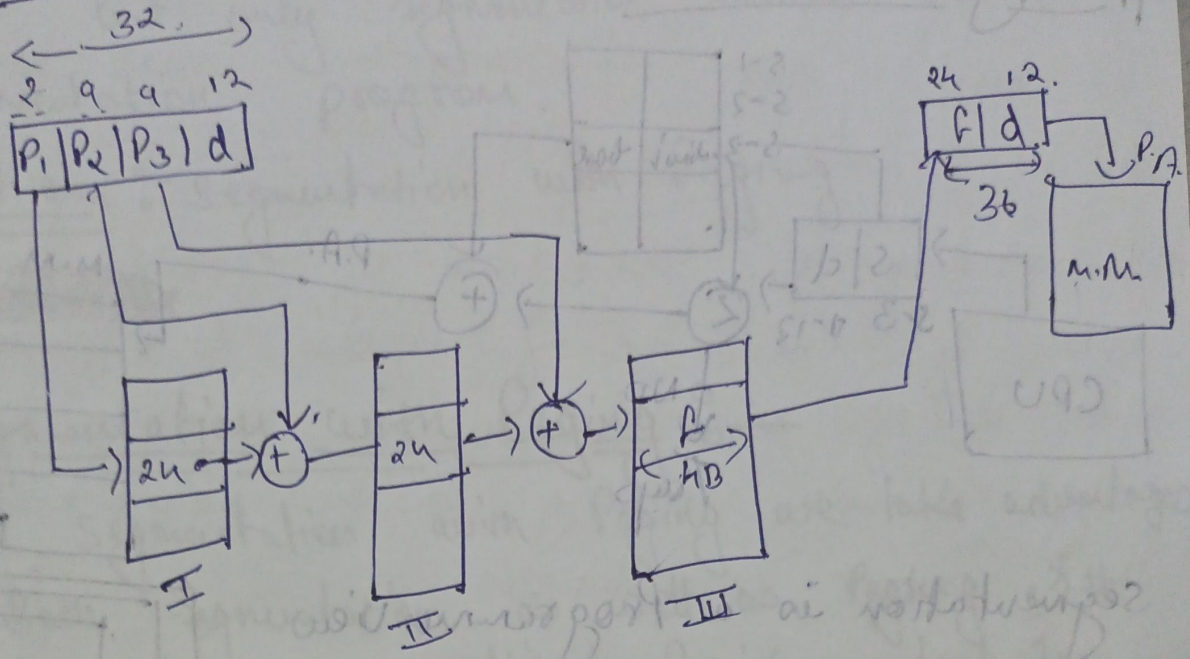
$+$

$.04\,[1ns + 100 + 100 + .9\,[1ns] + .1\,[1ns + 10ns]]$

TLB miss, TLB miss time

Pg no = 149

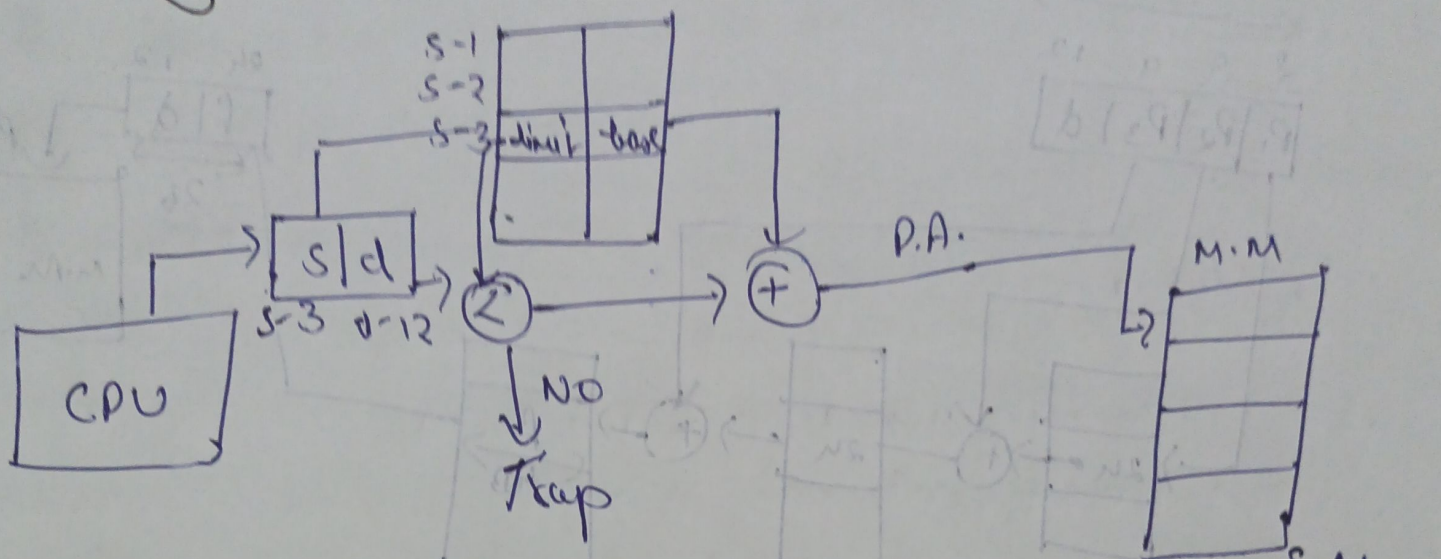
Q54

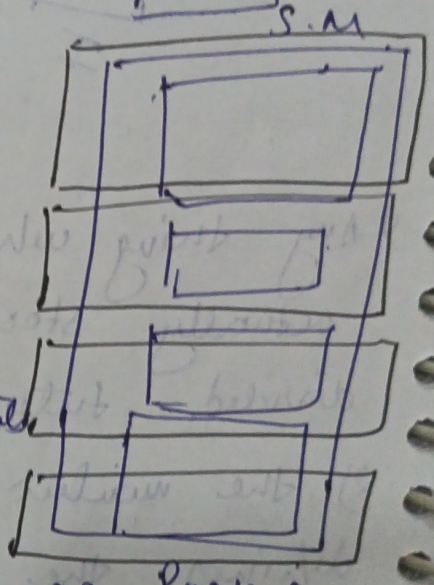


1) Any thing which is stored in main memory is actually stored in frames as main memory is divided into frames.

2) the number of bits allocated to P_1, P_2, P_3 defines the number of entries in the table but the size of entry in each table depends on F or frame number.

Segmentation :-





segmentation is a Programmer view of the memory where instead of dividing a process into equal size. partition we divide it according to programme into partition called Segment.

Translation is more or less same as Paging but segmentation is independent From internal Fragmentation. but suffer from external Fragmentation.

The reason of external Fragmentation is The programme Can be divided into segment but segment must be Continuous in nature.

#Conclusion :- The Idea of segmentation is Good but only segmentation Creates external Fragmentation program.

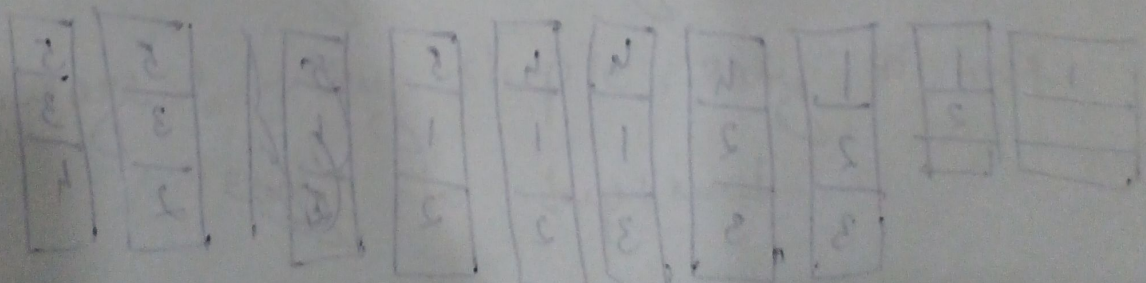
#Solution :- Segmentation with Paging

[redacted]

Segmentation with Paging :-

1) In segmentation with Paging we take advantages of Both segmentation as well as Paging. It is a kind of multilevel Paging but in multilevel Paging we divide a Page table into equal size partition but here segmentation with Paging we divide it according to segment.

3) All the properties are same as that of Paging because segment are divided into Pages.



Pure demand Paging :- ① Pure demand Paging is a Policy according to which a Page will not be loaded into main memory until it is refered by the CPU.

Virtual Memory :- the Idea of executing a Process even if the size of Process is greater then the size of main memory is virtually and using secondary memory as the main memory is the Concept or Idea.

Page fault :- Page fault is a scenario that if a Page is refered by a CPU but it is not present in main memory.

Page fault Service time :- It is the time used to load the required Page on updating the Page table.

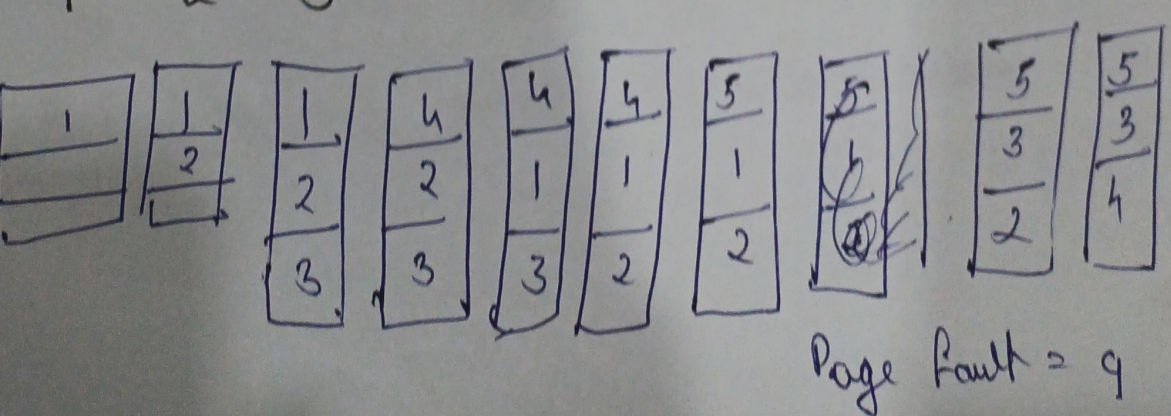
Page Replacement Policy :-

In FIFO.

Consider a Page Reference string

Frame = 3

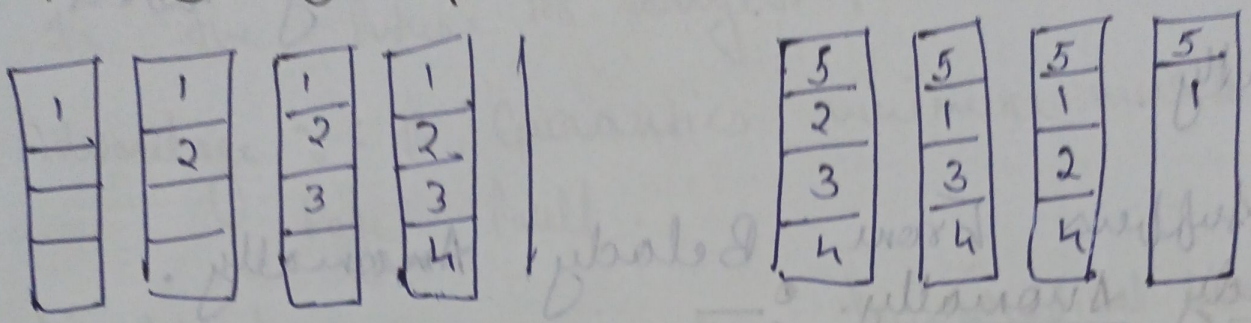
1 2 3 4 1 2 5 1 2 3 4 5

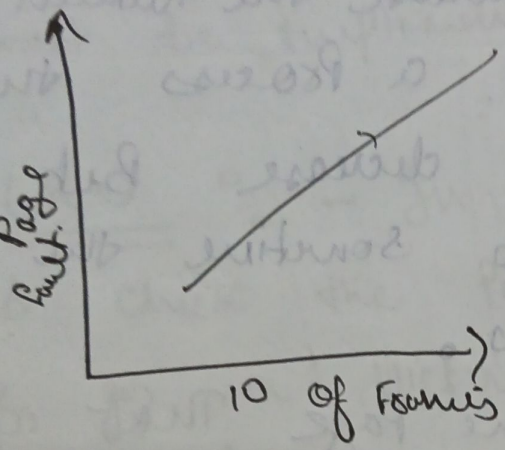


Page fault = 9

In FiFo, Frame = 4. (Oldest Page in the memory will be Replaced)

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





Page replacement Policy :- In PRP we discuss various algorithm to decide that which Page will be selected as a victim whether new Page is referred by CPU and main-memory is already full.

#FiFo :- this algorithm says the Page will be selected as a victim which is there in the memory for long time.

Advantage :- Very simple, easy to use
easy to implement.

Disadvantage :- ① It is not a Right approach because if a Page is there for longest it doesn't mean It is not being used Currently.

② It Suffers From Belady Anomally.

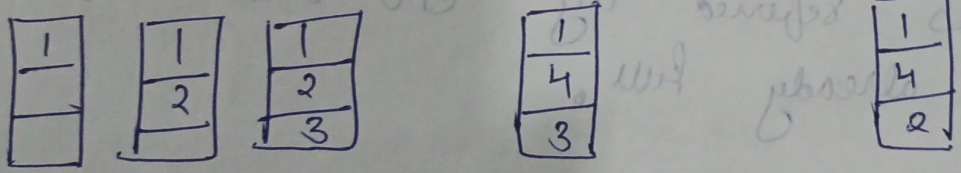
Belady Anomally :-

In General when we increase the number of Frames of allocated to a Process then Page fault Rate must decrease But as an exception In FIFO sometime the Page fault Rate increases.

Optimal :- Replace the Page That will Not be used for the longest Time. It has lowest Page fault

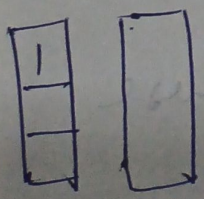
Frame = 3

1 2 3 2 4 1 3 2 4 1



Page fault = 5

LRU :- Page which has Not Been used for the longest Time is Replaced.



#Optimal :- this algorithm says that Page will be selected as victim which is not used in the future for longest.

Advantage :- It Guarantics minimum number of Page fault.

disadvantage :- It is a hypothetical Version and Cannot be implimented.

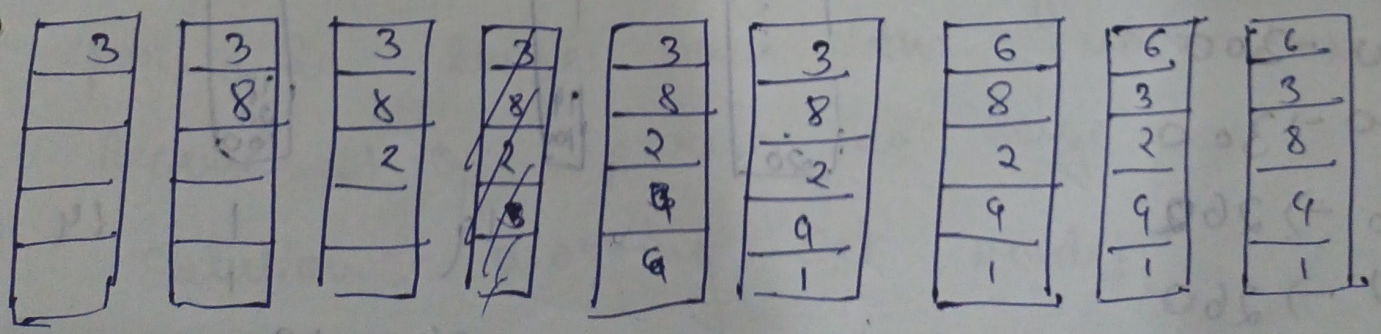
LRU :- In Case of Page fault we go and check the past string and select that Page as a victim which is not used for longest.

Advantage :- It is an efficient Version which is implimentable.

disadvantage :- Implimentation is Costly.

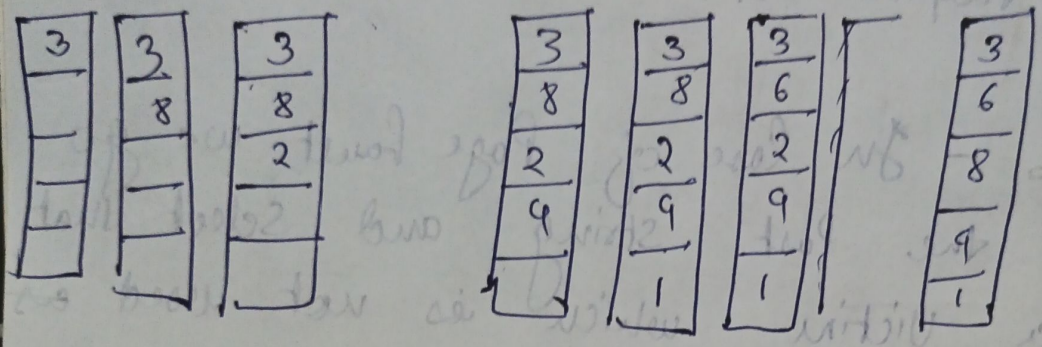
Booklet
Page 151
Q=64.

3 8 2 3 9 1 6 3 8 9

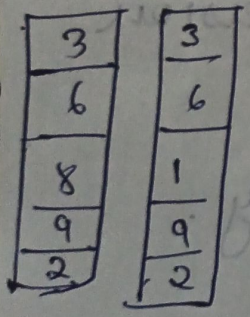


LRU

3 8 2 3 9 1 6 3 8 9 3 6



2 1 3



P.f = 9

Booklet P.g = 151

Q65.

LRU - 300

FIFO → 300

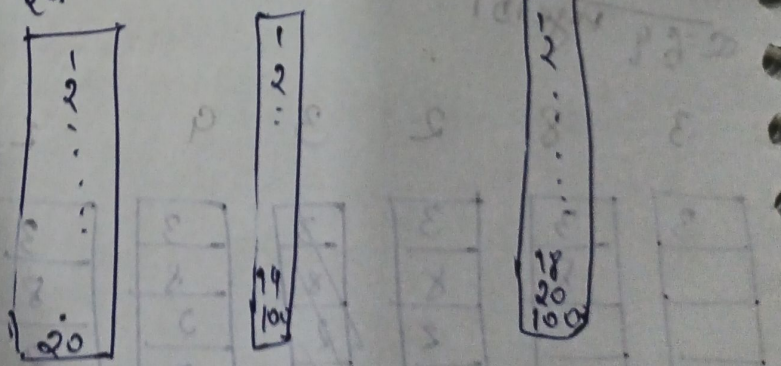
LIFO → 262

MRU → 260

Optimal = 262

MRU.

1 2 3 - 20 21 - - - 100 123 - - - 20 - 100 - - -



P. fault = 40

Space allocation in main memory :-

1) Equal Allocation :- This method says all the Frames available in main memory must be divided equally among the processes.

$P_1, P_2 - \ldots P_i - \ldots P_n$

Frames allocated to P_i $\boxed{P_i = \frac{F}{n}}$ (F = no. of Total Frames, n = number of Process.)

this algorithm is not logical because we don't know the actual requirement of the processes.

2) proportional allocation :- In this method the Frame allocated to a process is directly proportional to the size of the process.

S_1	S_2		S_i		S_n
P_1	P_2	..	P_i	..	P_n

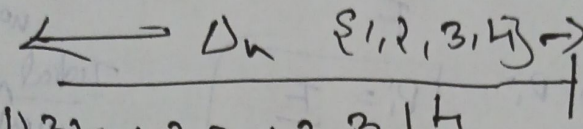
$$\text{Frames allocated to } = \frac{S_i}{\sum_{i=1}^{n} S_i} \times F$$

It is not necessary that if the size of process is large then the main memory Requirement of process is large.

Solution :- Working Set model.

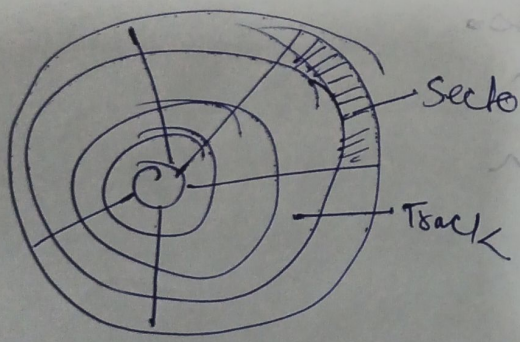
Working Set model :-

In this approach we check previous reference string for n number of References and the number of unique Pages in that set is called working set and same number of Process frames are allocated to process.



Transfer time for a Circular disk

Total transfer time = Seek time + rotational latency + file transfer time



Q. Consider a disk containing 512 bytes per sector 400 sector per track and 1000 tracks on the disk. If the disk is rotating at 15000 rpm. find the time required in order to transfer a file of size 1MB. Considering a seek time = 4 Millisecond.

15000 rpm → 250 rps

250 → 1s

1 → $\frac{1}{250}$ s

$\frac{1}{250}$ s → 400 sectors

1 sector → $\frac{1}{250 \times 400}$ s = 4 ms

Total Transfer time = Seek time + Rotational latency + transfer time

$= 4ms + \frac{4ms}{2} + 4ms \times \frac{1MB}{512 \times 400}$

Track Size = 512B × 400

file size = 1MB

no. of tracks = $\frac{1MB}{512B \times 400}$

file transfer time = time taken to * no. of track read on track.

* Seek time :- It is the time taken by the Read-write header to reach the desired track. (It is physical problem so we treat it as Constant).

Rotational latency :- It is the time taken by the disk to provide desired sector under the header. In Best case it can be 0. In worst case it can be time for 1 revolution so we take a average as time for half revolution.

file transfer time :-

time taken to x no. of track Read 1 track

OR

time taken to Complete 1 revolution.

no. of track = $\frac{\text{file size}}{\text{track size}}$

Q No. of sector per track is 16 each sector is of 1KB disk rotates at 3000 rpm find the time required to transfer a file of size 64KB? Seek = 15 milli

seek time + Rotational + file transfer time

15ms + $\frac{20}{2}$ ms +

$\frac{3000}{60}$ 50 rps

50 rps — 1s

1 rps = $\frac{1}{50}$ rps × 20.

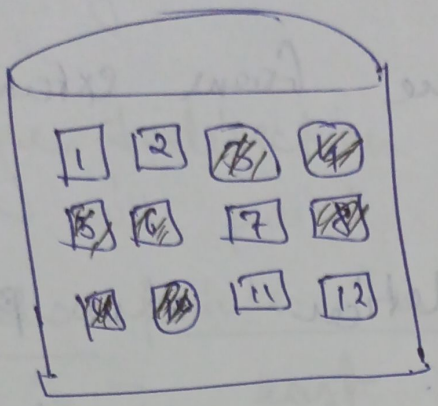
1

b) If file is stored in a non-contiguous fashion then file transfer time will be same but seek time + Rotational latency will be multiplied by no. of sector occupied by the file.

c) Same time hard disk support disk interleaving If it is single interleaving then transfer time x2. If it is double interleaving then transfer time x 2.75.

File allocation Pointer :-

Contigeous Allocation



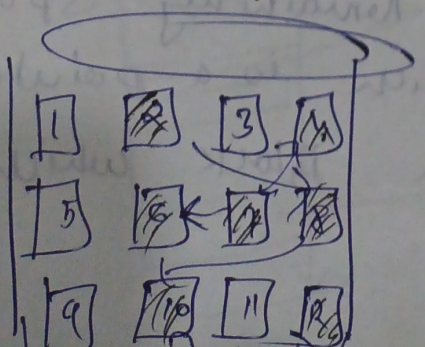
S.NO	Name	Starting DBA	Size
1.	POR	3	4
2	X Y2	8	

In Contigeous Allocation file are stored in Contigeous location and for each file Generally we remember Starting disk Block Address (DBA) And the number of Blocks Acquired.

Advantage :- (1) Extremly fast.
(2) Random Access is possible
(3) easy to Understand, easy to Implement

disadvantage :- 1) Suffers from Both Internal External fragmentation.
2) modification of file is difficult.

NON-Contigeous Allocation :-



SNo	name	Starting DBA	ending DBA
1	POR	2	12
2	X42	4	6

1) In this policy we implement a file like a linked list and last line of a Block Give the address of the next block.

Advantage :- 1) Independence from external Fragmentation.

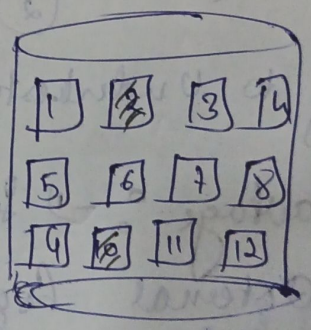
2) Modification :- Insertion / Deletion also possible

Disadvantage :- It is very slow like a linked only Sequential Search is possible.

1. we want the advantage of Both policies therefore we will Go for Index allocation.

Index Allocation :-

S.No	name	I-node	size
1	PQR	2	
2	XYZ	10	



In Index Allocation I-node will be reserved as a information which will Contain some metadata about the file and the remaining space stores direct DBA which is a pointer which pointer point to a Block which

Advantage :- Fast as well as flexible.

Disadvantage :- I-node is a limited and Capacity is limited.

Single Indirection DBA :- It is a pointer which will point to a Block which further Contains pointer which.

Formula :- # Total File Size / # Max File Size → last Page

Gate 2014 Consider a disk with 300GB capacity. I-node Contains 8 DBA. 1 single indirect DBA and 1 double indirect DBA. If each Block is of 128 Bytes. and each address is of 8 Byte find the maximum file size supported by file system.

No. of Disk Block address store inside one Block = 16

$= 8 \times 128B + 1 \times 16 \times 128B + 1 \times 16 \times 16 \times 128B$

$\Rightarrow 1KB + 2KB + 32KB$

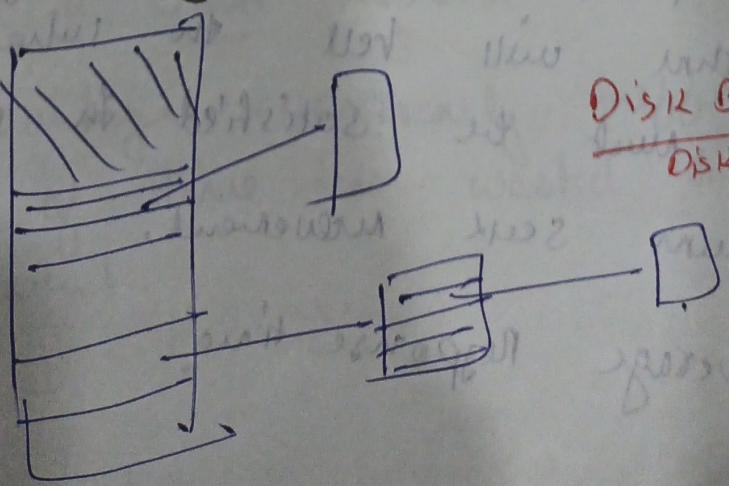
$\Rightarrow 35KB$

(a) 3KB

(b) 35 KB

(c) 280 KB

(d) Can't Say



Disk Block size / Disk block address $= \frac{128}{8} = 16$

Q Each Block 512 each address 16 Byte. I-node contains 16 direct dBA. 4 single indirect DBA and 2 double indirect DBA and 1 tripple indi...

(a) 256 KB

(b) ~~≈~~ 17 MB

(c) ≈ 18 MB

(d) Can't Say

~~512~~ ~~256~~ ~~128~~ 32
~~16~~ ~~8~~ ~~4~~ 2 1

No of Blocks = 32 = 32 Blocks

$16 \times 512B + 4 \times 32 \times 512 + 2 \times 32 \times 32 \times 512 + 1 \times 32 \times 32 \times 32 \times 512$.

⇒ 8KB + 64KB + 1MB + 16MB

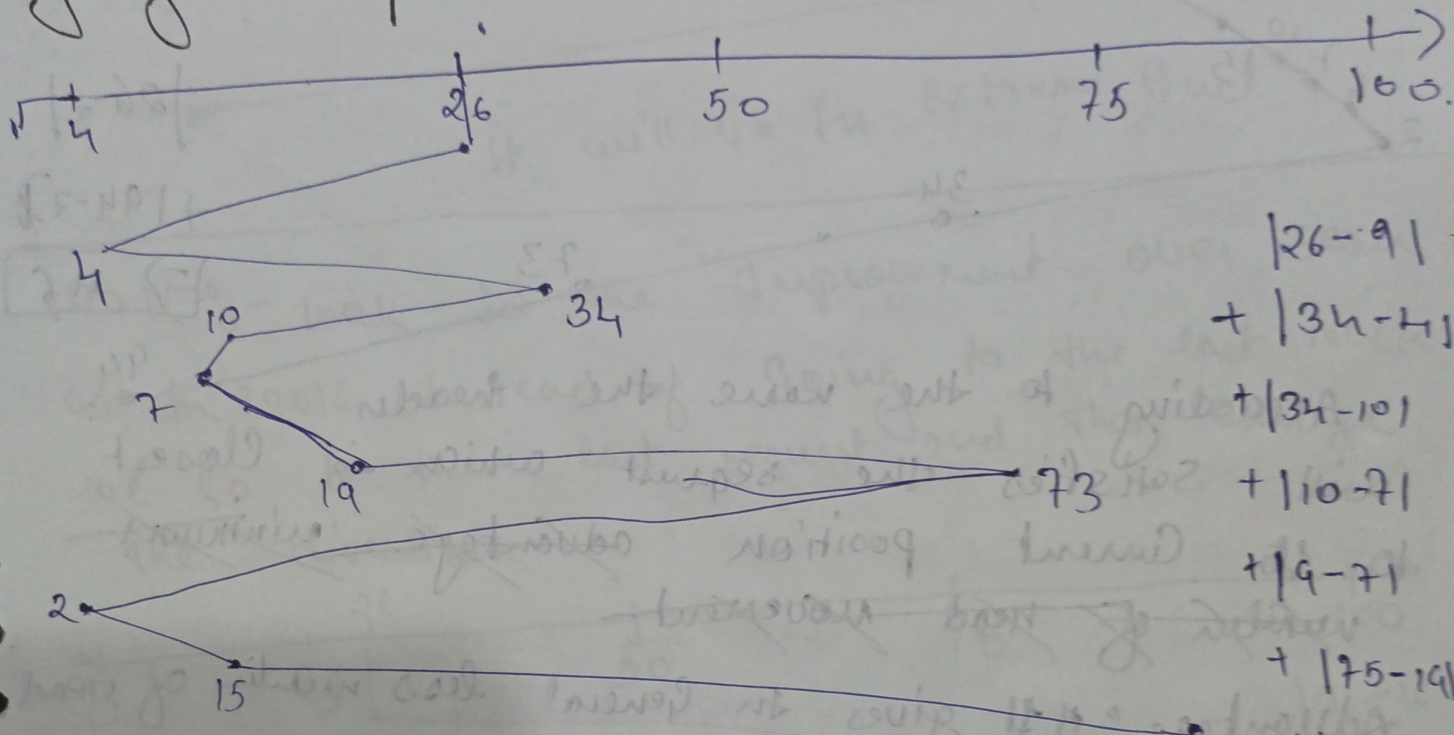
⇒ 17 MB

Disk Scheduling Algorithm (1 mark) (1-2)

Here we understand that on a large disk there could be a number of tracks and we may have request for a number of track this algorithm will tell in what order the request must be satisfied in order to provide ① minimum seek movement.

② Better Average Response time.

Q. Consider a disk with 300 tracks if there is a request in order 4, 34, 10, 7, 19, 73, 2, 15, 94. Find the minimum number of head movement if we use first come first serve algo - Considering ~~Header that~~ header is originally going 26 up.



|26-9| +
+|34-4|
+|34-10|
+|10-7|
+|9-7|
+|75-19|
+|73-2|
+|15-2|
+|94-15|

sum = 308

FCFS

This algorithm says request will be satisfied in the order in which they arrive.

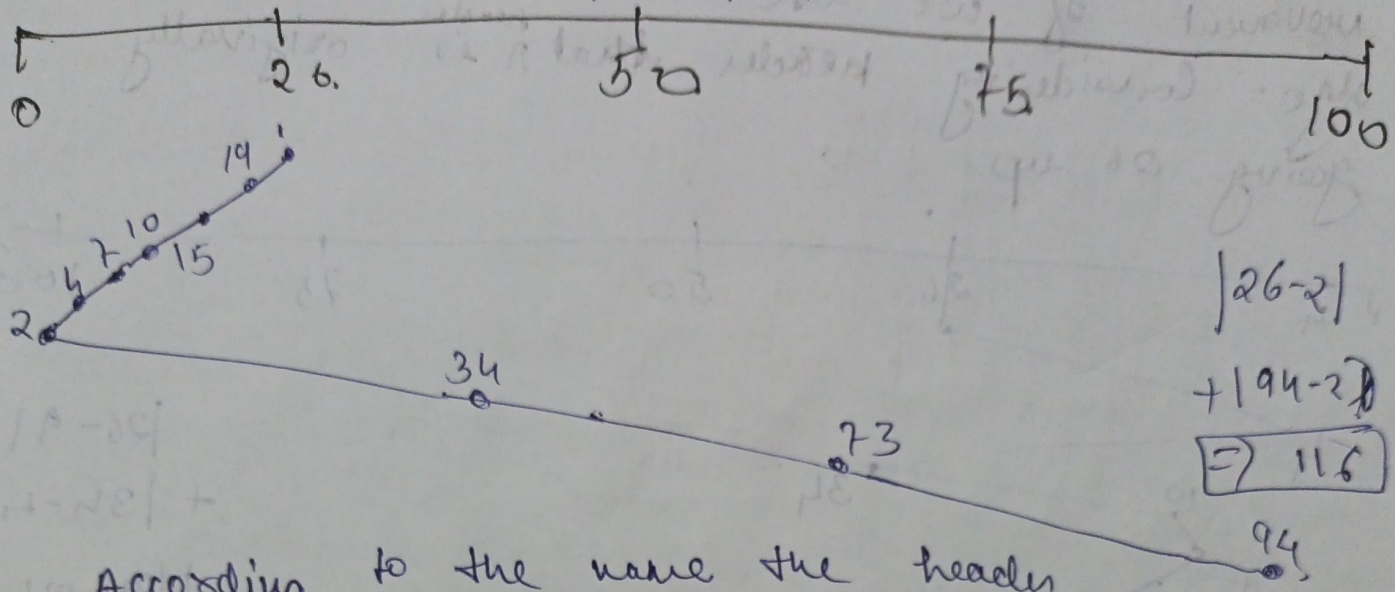
Advantage :- 1) Easy to use, Easy to understand.
2) Provide a Good Response time.

disadvantage :- 1) Extremely inefficient
2) A lot of time is wasted in seek or head movement.

Solution :- SSTF (Shortest seek time first)

SSTF

4, 34, 10, 7, 19, 73, 2, 15, 94



|26-2|
+|94-2|
= 116

According to the name the header will satisfies the request which is closest to its current position ~~advantage minimum number of head movement~~

Advantage :- It gives in general less number of head movement.

Disadvantage :- Here the request which are far from head will starve or will provide poor response time.

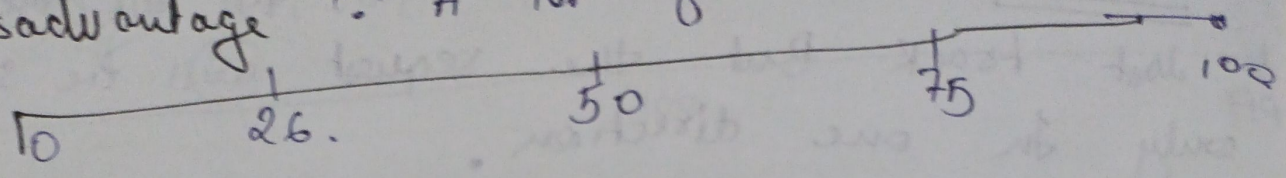
Scan (Elevator algorithm)

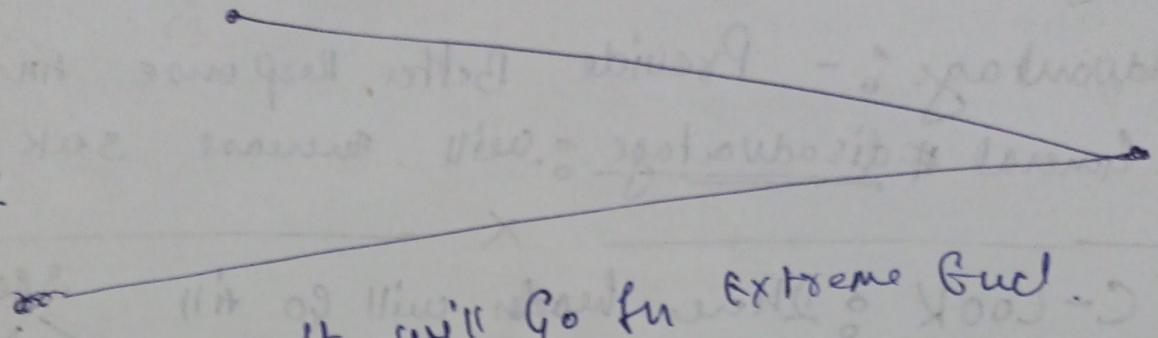
Scan algorithm will start from a point and will satisfy every request which will come in between and will go to extreme points in both ends

Advantage :- Relatively Better Response time than SSTF & FCFS

Provide Poor Response time as satisfies in both directions

Disadvantage :- A lot of Extra movement.

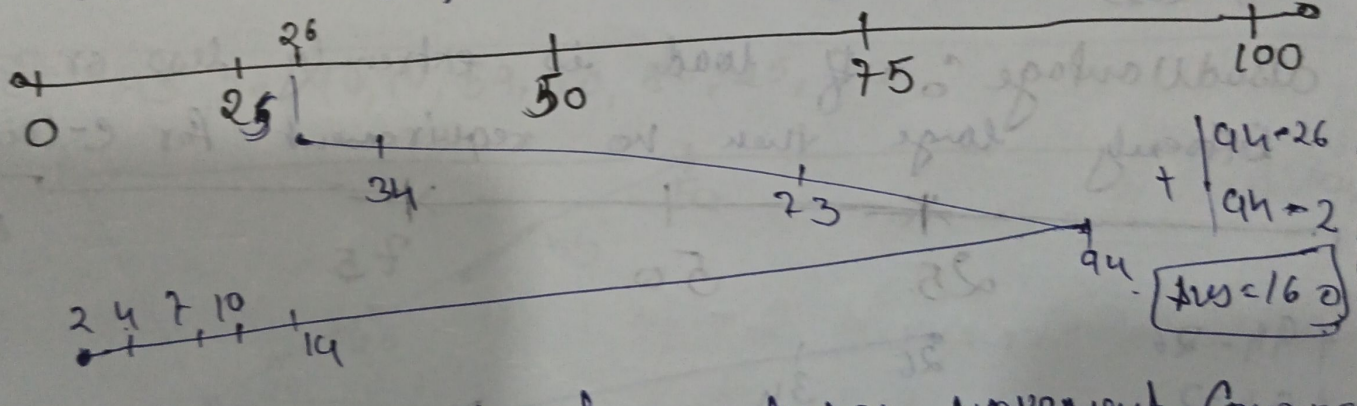




=> 172

It will Go to Extreme End.

100K :- 100K is the Improvement over Scan as in 100K instead of Going to the last track we Go to the last request and then change the direction.

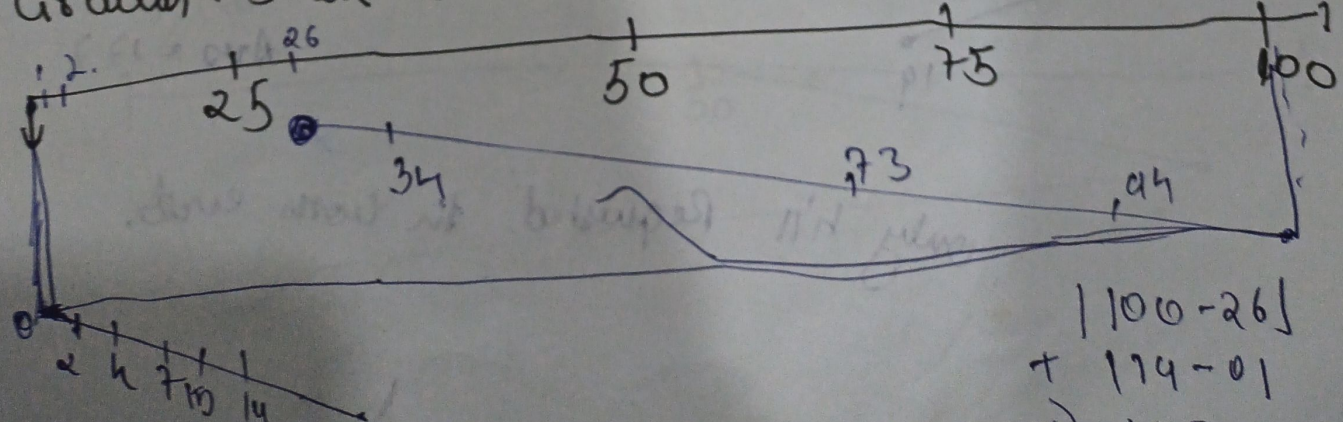


Advantage :- It will have less arm movement Compare to Scan it may be lesser movement than Scan

---

~~J.M.P~~

Circular - Scan :-



|100-26|
+ |14-0|
=> 193

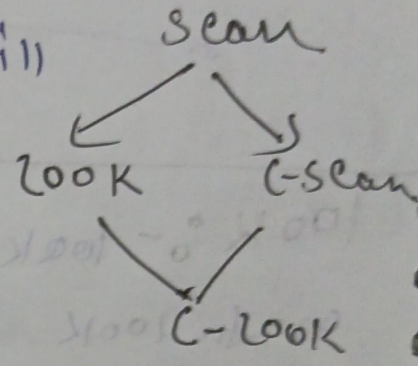
C-Scan :- In C-Scan will Go till the last track But the request will be satisfied only in one direction.

Advantage :- Provide Better Response time in General

#disadvantage :- will increase seek time

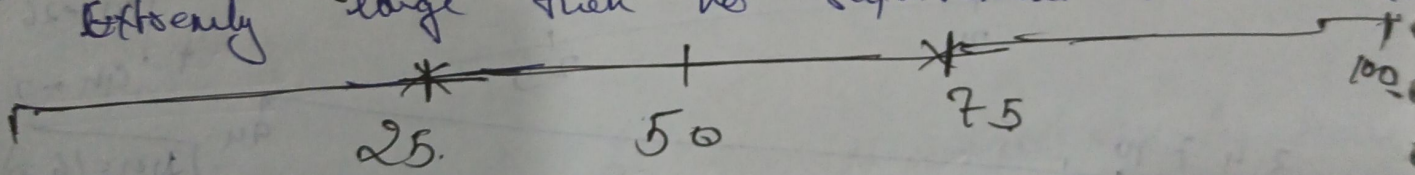
C-LOOK :- Here header will Go till the last Request.

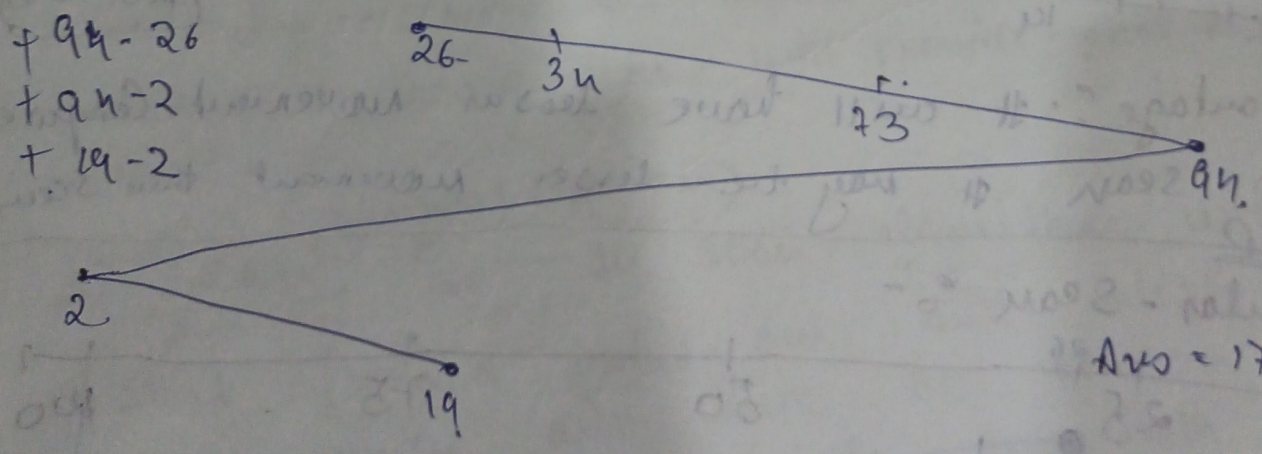
2) will satisfies ~~only~~ Request only in one direction



Advantage :- 1) Better Response time.
2) less seek Movement

disadvantage :- If load is extremly less or Extremly large then no requirement for C-LOOK.





Ans = 177

only till Requested. in both ends.

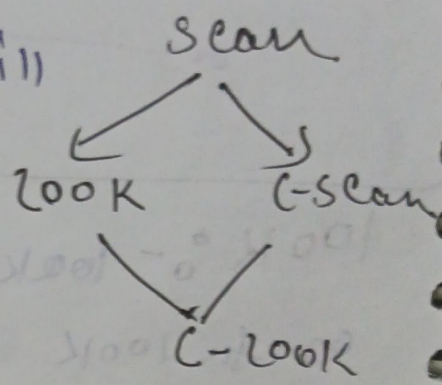
C-Scan :- In C-Scan will Go till the last track But the request will be satisfied only in one direction.

Advantage :- Provide Better Response time in General # disadvantage :- will increase seek time

---

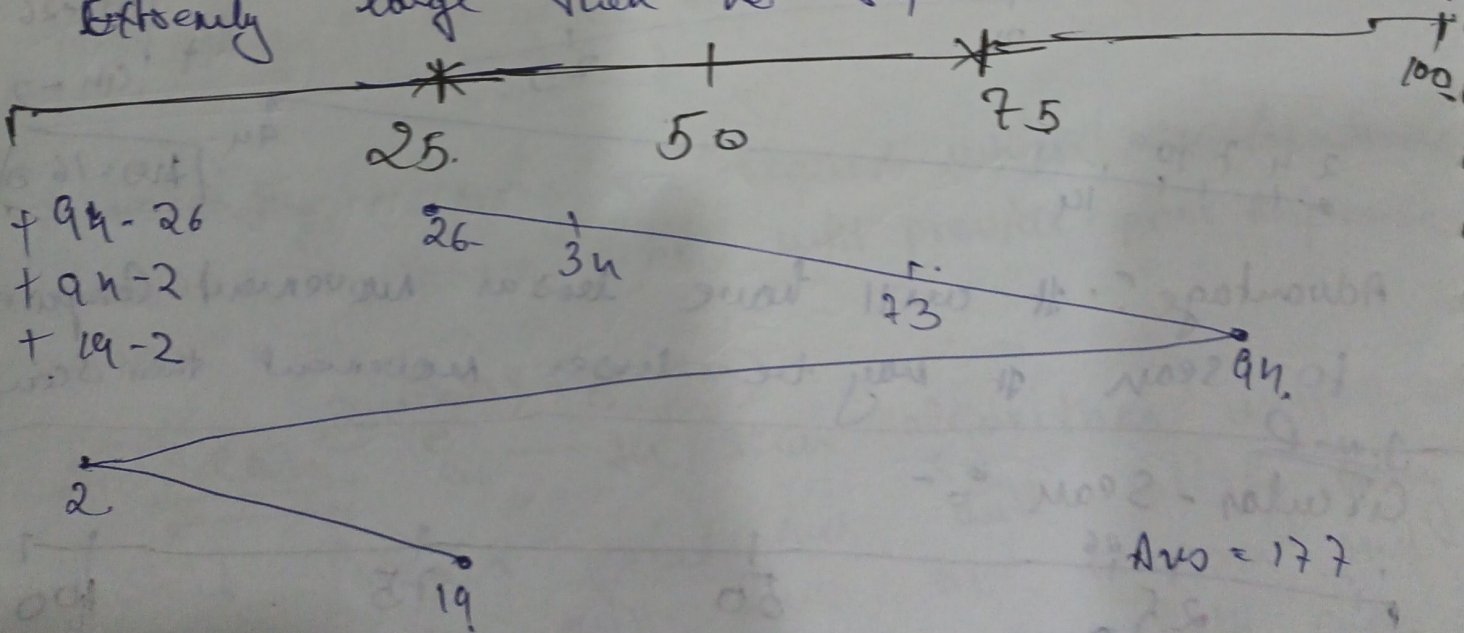
C-LOOK :- Here header will Go till the last Request.

2) will satisfies only in one direction



Advantage :- 1) Better Response time.
2) less seek Movement

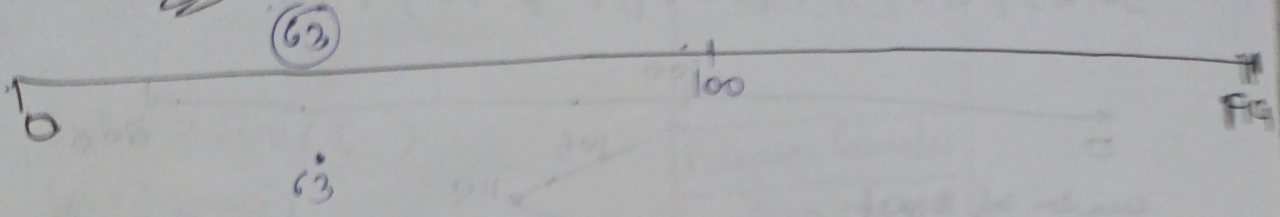
disadvantage :- If load is extremly less or Extremly large then no requirement for C-LOOK.



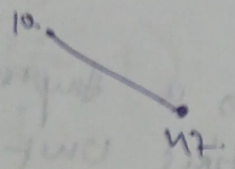
Avg = 177

only till Requested. in both ends.

Q 97, 38, 121, 141, 81, 11, 92, 10.

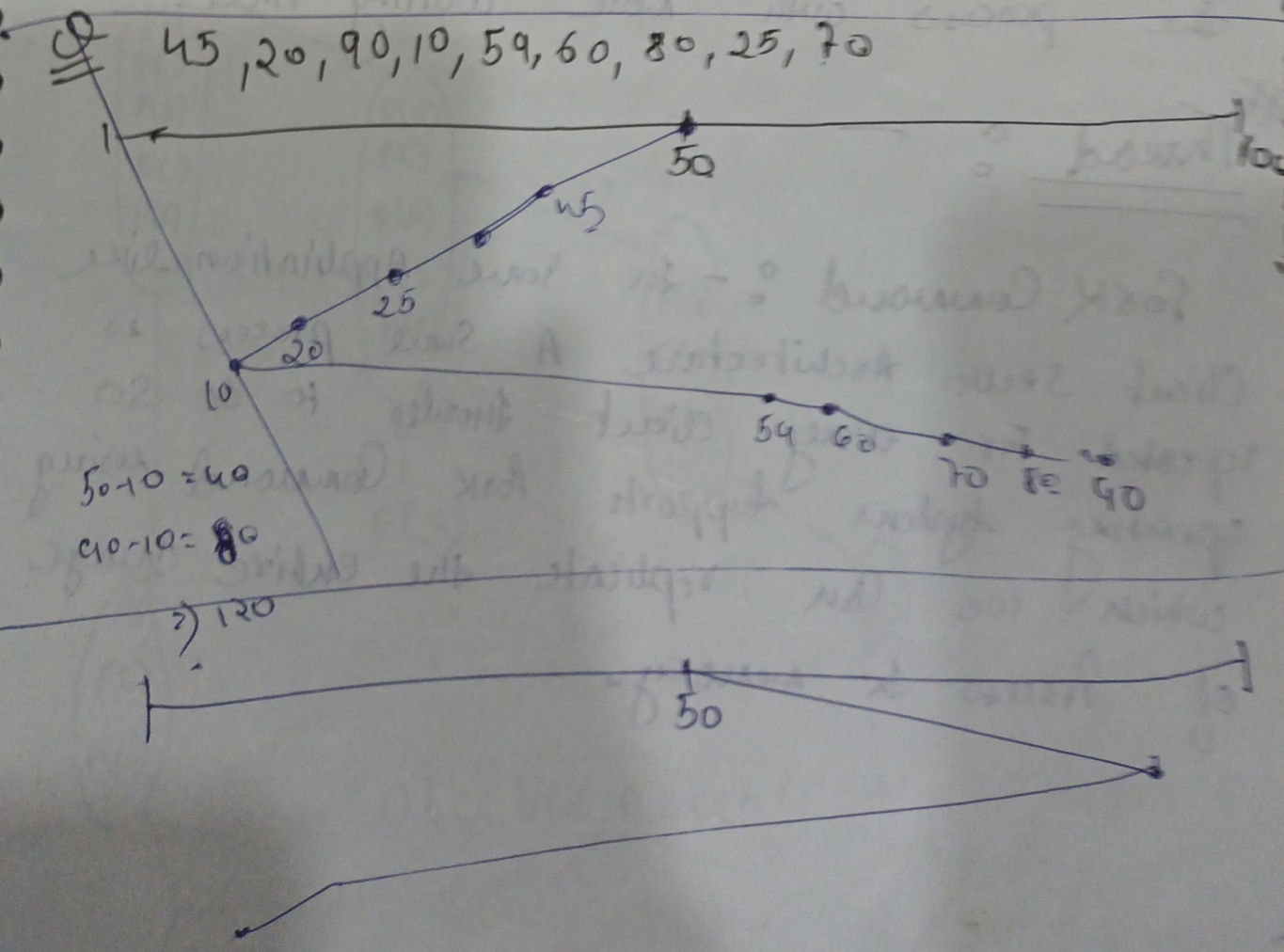


63

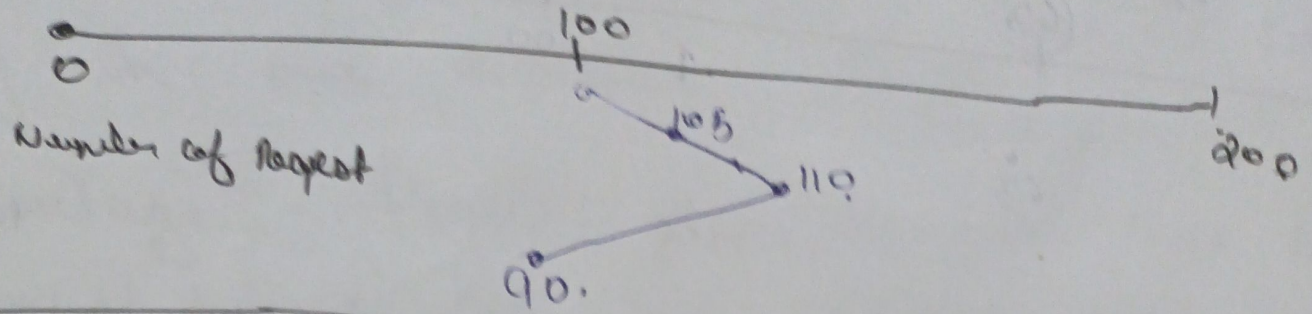


$|141 - 63|$
$+|141 - 10|$
$+|47 - 10|$
$\Rightarrow$

Q 45, 20, 90, 10, 59, 60, 80, 25, 70



90, 100, 105, 110, 135, 145.



UNIT-1 OS

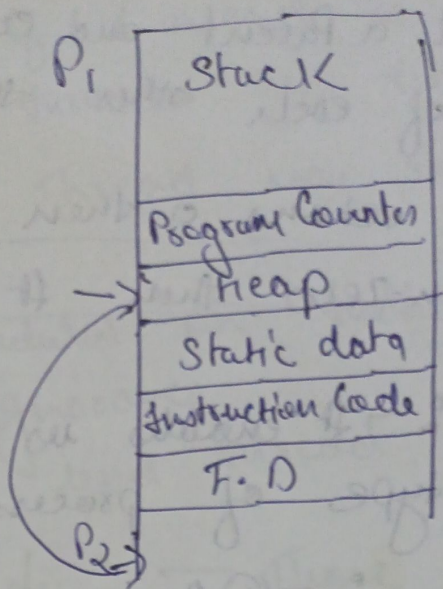
Highest Response Ratio Next :- It is a improvement in SRTF algorithm after every time unit we Compute priority function which is $\frac{B.T. + W.T}{B.T.}$.

Conclusion :- It Gives priority to process which have lesser Burst. But it also Gives value to the process with more waiting time.

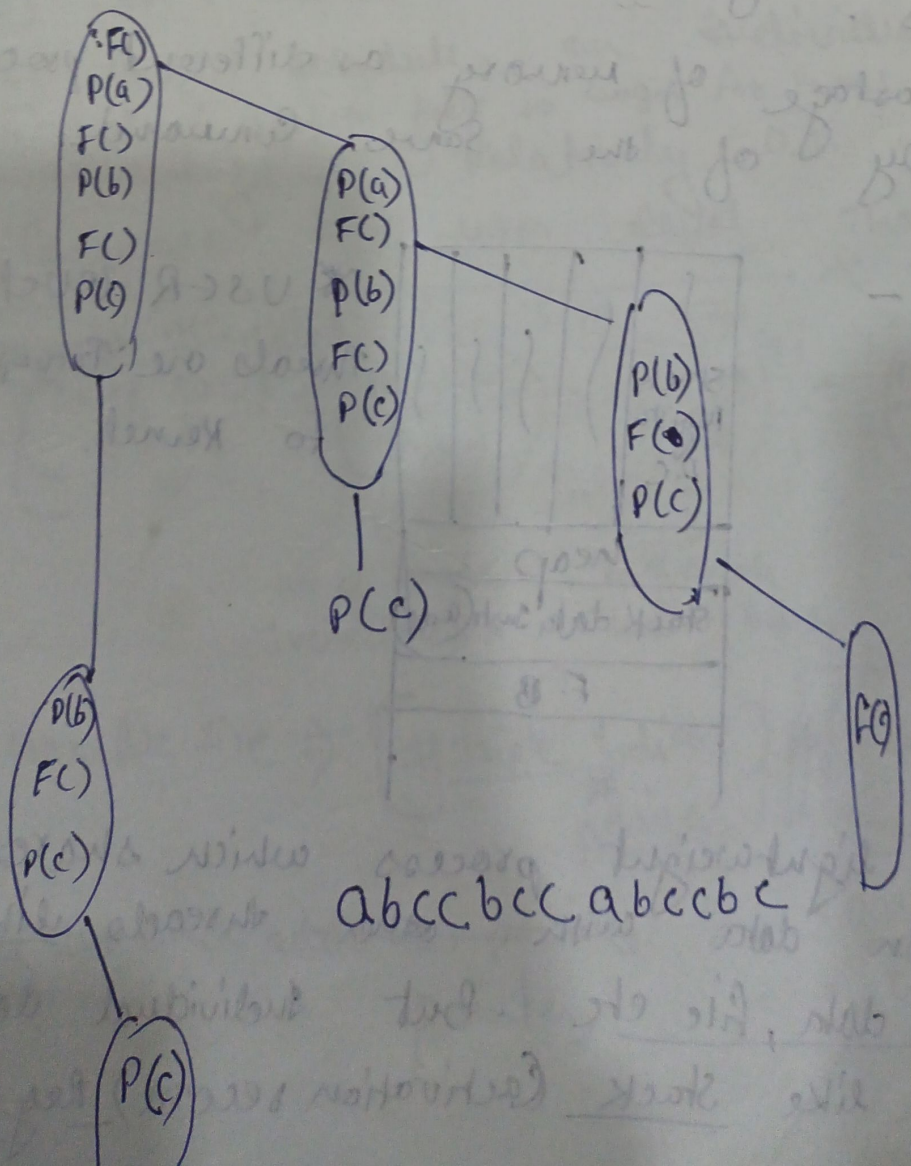
Thread :-

Fork Command :- In some Application like Client Server Architecture A same process is repeated for every client therefore to do so operating system supports Fork Command using which we can replicate the entire Image of Process in memory.

```
Fork()
{
  Int i = Fork()
  if (i==0)
    {
      child
    }
  if (i==1)
    {
```



Using Fork Command the same code after the fork will be replicated once. ∴ If we have n number of Fork Command in code it will Generate $2^n - 1$ child.



abccbccabccbc

In General: a Parent and child are exactly the identical of each other therefore to different.

If fork returns 0 then its child process if 1 or non-zero then it is parent process.

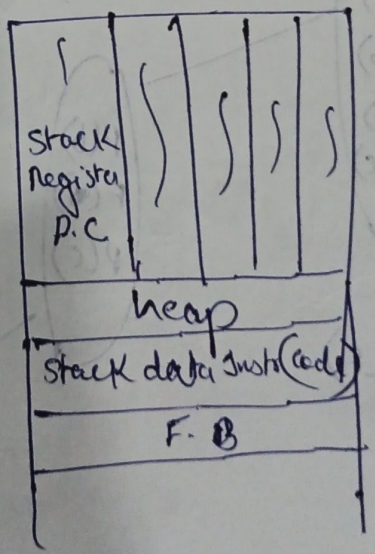
Advantage :- It enables us Creating multiple Copies of same type of process in very less time.

disadvantage :- ① It is a priviledged Command and whenever a child process is required then it must be done by OS.

② As new child process is also indepenent process because of this the Burden on OS is increases.

③ It is wastage of memory as different processor maintains Copy of the same Command

Threading :-



* USER level Threads are Transparent to Kernel.

Threads are lightweight process which shares some Common data with other threads like Code, static data, file etc. But individual data are different like Stack (activation record) Register P.C.

Advantage :- ① If 1 thread is Block by Os all the other threads will be blocked.

② Some individual Thread may starve for Resources.

③ Cannot be scheduled on individual process because of O.s ignorance.

Solution :- Kernal level thread.

Kernal level Threads :- Those Threads which are manage or maintained by O.s. Called O.s Kernal level Threads.

Advantage :- ① Can block individual Threads

② Can allocate additional Resources

③ Can schedule on individual Processores.

④ Context switch time is longer for Kernel level than for user level

disadvantage :- ① Relatively O.S. is in Burden Compare to user level Threads.

Formula # Total file size =>

$$\left[\text{no. of direct} * \text{D.B.Size} + \left(\frac{\text{D.B.Size}}{\text{D.B.A}}\right) + \left(\frac{\text{D.B.Size}}{\text{D.B.A}}\right)^2 + \left(\frac{\text{D.B.Size}}{\text{D.B.A}}\right)^3\right] * \text{D.B.Size}$$

no. of Single Indirect D.B.A

no. of double Indirect D.B.A.

no. of Tripple D.B.A.

Max file size => [# Triple Indirect * D.B.Size] * [D.B.Size × (DBSize / D.B.A)^3]

∴ no. of disk Block address possible => Disk Block Size / Disk Block address.

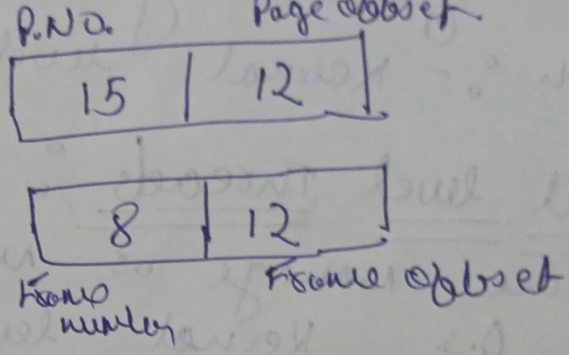
pointer

Ravindra Baley

Q LAS = 128 MB
PAS = 1 MB
Page size = 2^{12} B.

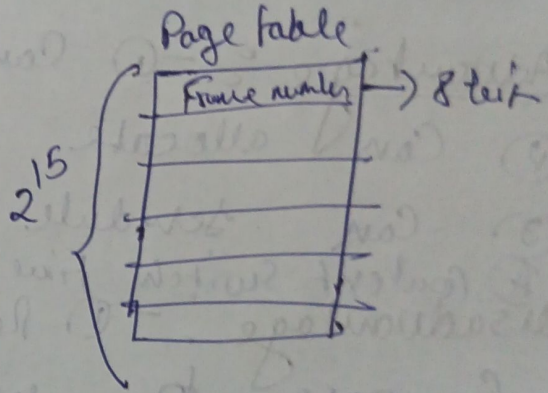
Find no. of Entries in Page table?
Find the size of Page table?

L.A = 27
PA = 20
Page size = 12



Number of Pages = 2^{15}

As = 2 so number of Entries = 2^{15}



Page Table size = 2^{15} x 8 bits
$\Rightarrow 2^{15}$ Bytes.

Logical Address space => No. of Pages * Page size

Kernel level thread can share the code

LAS	PAS	LA	PA	Page size	Page offset	Pages	Frames	PTE (Page Table Entries)	No. of Pages × size of each Page
128 KB	128KB	17	17	4KB	12	2^5	2^5	5	$(2^5 \times 5)$ bits
256 KB	2MB	18	20	4KB	12	2^6	2^8	2B	$(2^6 \times 2)$ B
1MB	256 KB	20	18	2^{10}	10	2^{10}	256	8	$(2^{10} \times 8)$ bit
1MB	512 MB	20	29	2^{12}	12	256	2^{17}	17	(256×17) bits
2^{22}	2^{21}	22	21	2^{12}	12	2^{10}	2^9	9	$(2^{10} \times 9)$ Bits
2^{22}	2^{22}			2 1H.		2^8	2^8	8	256 B

L.A.S = No. of Pages × Page size

Numerical on Paging

① spooling device

② Thrashing

---

① (a) Rotational latency.

3600 .. per min (rpm)

3600 rotation / 1 min (60 sec)

(Rotation time) 1 rotation = $\frac{60}{3600}$ sec = 16.6 ms

Rotational latency = $\frac{1}{2}$ (16.6) = 8.3 ms

(b) Transfer time :— $\frac{1}{2}$ (Rotation time)

Transfer Rate = $\frac{\text{Bytes on Track}}{\text{Rotation time}}$

then, transfer time = $\frac{\text{Total amount to be transfer}}{\text{Transfer Rate.}}$

CPU Utilization :- $1-(p)^n$

n = no. of Process.

'p' : It is the fraction of time to do I/O operation

simply CPU utilization depends on degree of multiprog

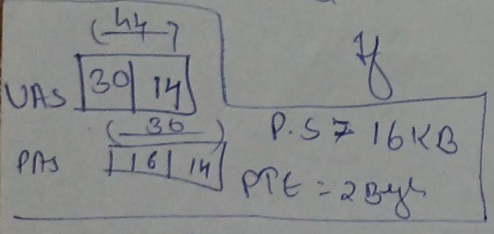
Linker :- Function of linker

1. Relocation.
2. Symbol resolution.

Loader :- Program loading + Relocation

- Linked allocation does not support direct access but indexed and Contiguous allocation support direct access.

- To find number of level of Paging



Page size < Page Table size

then

PS < PTS [So we need PS = PTS]

level 1 = $\frac{2^{44}}{2^{14}} \times 2\text{Byte} = 2^{31}$ Byte $\frac{VAS}{P.S} \times PTS$

level = 2^n $\Rightarrow$ $\frac{2^{31}}{2^{14}} \times 2\text{Byte} = 2^8$

level = 3 $\frac{2^{18}}{2^{14}} \times 2\text{Bytes} = 2^5$

[Deg = 3]

- Processes and Kernel Threads can Run in Parallel in multiprocessor system.
- Reading the clock of system can be done in user mode.
- Accessing the I/O needs privileged instruction.
- Locality of Reference :- Implies that the Page reference being made by a Process is likely to be the Page used in the previous page reference.
- In a system with virtual memory context switch includes extra overhead in switching of address space.
- Kernel level threads are best suitable for I/O bound processes.

- Thread does not share stack.
  Thread can share both Heap & Global variables and address space.
- System Calls are usually invoked using a Software Interrupt.
- Software interrupt are system services which needs execution of Privileged instruction.

1) fork
2) Thread
3) Rotational latency
4) double indirect, Direct, Triple indirect

OS Questions

Q 1.33 4.50 4.46 5.18
Q 1.34 4.49 4.60 5.29
4.47 5.16

---

* SRT scheduling may cause starvation.
* Preemptive scheduling may cause starvation.
* Kernel level threads shares the Code Segment.
* In deadlock prevention, the request for a resource may not be Granted even if the Resulting state is safe.

But in deadlock avoidance it is ~~Resources~~ which is always allocated.

#

Virtual Memory. → Spatial locality
Shared memory. → ~~Mutual exclusion~~ ~~Temporal locality~~
look ahead buffer. → ~~Address translation~~ Temporal locality.
look aside buffer. → ~~Mutual exclusion~~ Address translation.

Critical region → Mutual exclusion.
wait/signal → Hoare's monitor.
working set → Principle of locality.
Deadlock → Circular wait.

a) Buddy System → Memory allocation
b) Interpretation → Runtime type specification
c) pointer type → Garbage collection
d) Virtual memory → segmentation

* Link editor acts as a link between Compiler and user program.

* 2 pass assembler → 1 pass :- allocate space for literals
2nd pass :- object program is generated

Q The Capacity of a memory unit is defined by the number of words multiplied by the number of bit/word. How many seperate address and data lines are needed for a memory of 4K x 16 ?

Ans :- size of Memory 4K x 16 bits
$2^{12} \times 16$ bits

Ans :) 12 address lines & 16 data lines are needed

Q Suppose the time to service a Page fault is on the average 10 miliseconds, while a memory access takes 1 micro-seconds. then a 99.99% hit ratio results in average memory access time of

Ans :- Average memory Acess time = [(% of Page miss) * (time of service a Page fault) + (% of Page hits) * (Memory access time)] / 100

So, average memory access time :) $\frac{[99.99 * 1 + 0.01 * 10 *}{100}$

:) $\frac{199.99}{1000}$:) 1.9999 usec.

* Virtual memory increases the Context switching overhead.

* The minimum number of Page frames allocated to a Running process is determined by the instruction set architecture.

* Effective average instruction Execution time.

=) CPU time + P x S + (1 - P) EMA

Page Fault Rate = P = 1/104

EMA = 165 ns

S = 8 MS

CPU time = 100

Ans - 1229.997 ≅ 1230 ns

* Effective TLB access time =)

hit ratio * (TLB access time + Memory access) + miss ratio *

(TLB access time + Page table access time + Memory access time)

=) 0.9 * (10+50) + 0.1 [10+50+50] = 65

* Dirty → write-back policy.

R/W → Page Protection.

Reference → Page replacement policy.

Valid → Page initialization.

Discrete Mathematics

Trick To solve.

Q $(x \vee y) \leftrightarrow \sim x \rightarrow \sim y$

Sol:- $(x \vee y) \leftrightarrow [(\sim(\sim x)) \vee \sim y]$

$(x + y) \leftrightarrow [x + \bar{y}]$

$((x+y) \cdot (x+\bar{y})) + (\overline{(x+y)} \cdot \overline{(x+\bar{y})})$

Ans * The data blocks of a very large files in the unix file system are allocated using extension of indexed allocation or EXT 2 file system.

Made easy :-

* FCFS, SRT and Shortest Job First are used by short term scheduler to move the process from ready state to running state but not by long term scheduler.

— x —

* If all jobs ~~have~~ are identical, RR is a horrible for turnaround time because all jobs will complete at nearly the same time.

* In MLFQ jobs that have long CPU Burst are given low priority.

— x —

Q A bit map can be used to keep track of which block are free in a file system's partition on disk. Assuming 1KB block size and disk size of 40GB what is the size of bit map ?

⇒ Disk size = 40 GB
Block size = 1 KB

Number of blocks = $\frac{40 \times 2^{30}}{8 \times 2^{10}}$

=) 5 MB

— x —

* Long term scheduler controls degree of multiprogramming.

* Short term scheduler loads the process from ready state to Running state.

* DMA I/O → Disk
Cache → High speed RAM.
Interrupt I/O → Printer.
Condition Code Register → ALU.

* Disk requires a device driver.

* TLB is a hardware data structure.

* The operating system automatically loads Pages from disks when necessary when it is needed.

* The CPU utilization increases as the degree of multiprogramming increase up to threshold (some limit) after that utilization start decreasing.

* Bounded waiting does not always satisfied if there are only two process.

* As the size of block in fixed partition scheme increases, chance of internal fragmentation also increases. Because with larger block size, if a smaller process arrives, remaining block will be wasted. This is because one block can be allocated to only one process at time.

* Variable size partition scheme suffer from external fragmentation. segmentation is one of the variable partition scheme.

Q. Consider two Process P_1 and P_2 each needed 3 resources 1, 2 and 3 in a database, If each process ask them in any order then the number of ways possible in which system is granted to be deadlock free :— A

Sol:- Minimum number of Resource Required =
(Number of resources required by $P_1 - 1$) +
(Number of resources required by $P_2 - 1$) + 1

$\Rightarrow (3-1) + (3-1) + 1$

$= \underline{5}$ Ans.

---

* An operating system implements a policy in which resources are numbered uniquely and Process are allowed to request for resources only in increasing Resource number then starvation may occur but not dead lock.

* Dirty bit is used to check for modified Pages not for reference Count.

Valid bit is used to check availability of Pages in the main memory.

* Kernel stack is not used to store the user Program functional arguments [False]

True statements.

* Kernel stack can be used to store Context of Process.
  - User stack is used to store the function detail during function Calls.

In Kernel mode things allowed are :-

1. changing mapping from virtual to physical addres
2. Disabling all Interrupts.
3. Mask and unmask Interrupt.

---

* Interprocess Communication message Passing require Kernel supports.

* Test & set lock Generates.
  1. Progress 2. Mutual Excl. 3. No deadlock

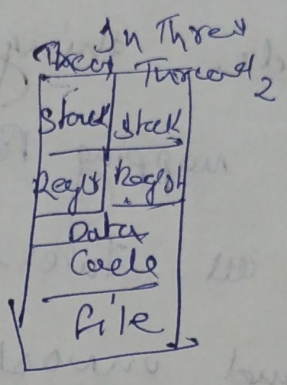
* The set of Pages that a Process is Currently using is called working set.

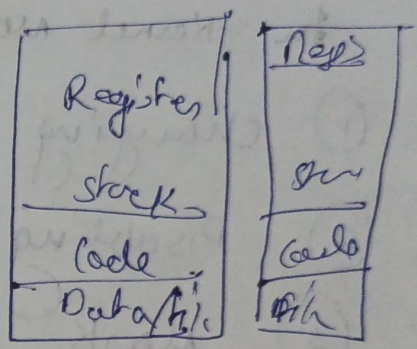
---

* on Per thread basis, operating system does not maintain virtual memory state, it maintains address space for the process not for threads

* One per thread basis, OS maintains Register state for every thread.

10^{12} — Tera T
10^{9} — Giga G
10^{6} — Mega M
10^{3} — Kilo [K]
10^{2} — hecto [h]
10^{1} — deca [da]
10^{0} — none [G, litere, second]
10^{-1} — deci d
10^{-2} — centi c
10^{-3} — miti [m]
10^{-6} — micro [μ]
10^{-9} — nano [n]
10^{-12} — pico [p]





Process

1) System Calls Involved in Process.
2) OS treats different process differently.
3) Different process have Different copies of Data, files, Code.
4) Context Switching is * Slower
5) * Blocking a process will not block another process.
6) Independent.

Threads

1) There is no system calls involved
2) All user level thread are treated as single task for os
3) Threads share the same copy of code & data.
4) Context Switching is faster.
5) Blocking a thread will block entire process
6) Interdependent.

USER LEVEL Thread

1) USER level thread are managed by user level library
2) User level threads are typically fast.
3) Context switching is faster
4) If 1 user level thread performs Blocking all the other thread and process gets Blocked

Kernel level Thread.

Kernel is the integral part of O.S.

1) Kernel level threads are managed by OS System calls.
2) Kernel level threads are slower than user level.
3) Context switching is slower
4) If one Kernel level thread blocked, No effect on other.

Q A UNIX I-Node has 10 direct pointers and one single, 1 double and 1 triple indirect pointer. Disk block size is 1 KByte. disk block address is 32 bits and 48 bit integer are used what is the maximum possible file size?

⇒ Max file size = [Triple Indirect]^3 × Disk block size

I.M.P

[NOTE :- Disk block address must be in Bytes. OR it can be Blocks.]

Disk block address = $\frac{32}{8}$ 4 Byte.

Disk block size = 2^{10} Bytes = 1 KB

Triple Indirect :- $\frac{2^{10}}{2^2} = 2^8 = 256$.

$256 \times 256 \times 256 \times 1\,KB$

$2^8 \times 2^8 \times 2^8 \times 2^{10}$

2^{24} KB

Multilevel Paging :-

VA	Page size	PTE	Page Table 1	Page Table 2	Page Table 3	address split
48b	16KB	4B	$2^{34} \times 2^2 = 2^{36}$ B	$2^{22} \times 2^2 = 2^{24}$ B	$2^{10} \times 2^2 = 2^{12}$ B.	10 12 12 14
64b	1MB	4B	$2^{44} \times 2^2 =$ B	$2^{26} \times 2^2 =$ B	$2^{8} \times 2^2$ B	8 | 18 | 18 | 20
72b	1GB	4B	$2^{42} \times 2^2 =$ B	$2^{14} \times 2^2$	—	14 | 28 | 30 →
72b	256MB	4B	$2^{44} \times 2^2$.	$2^{18} \times 2^2$		18 | 26 | 28.
72b	16MB	4B				P_{i_2} P_{i_1} Poffset

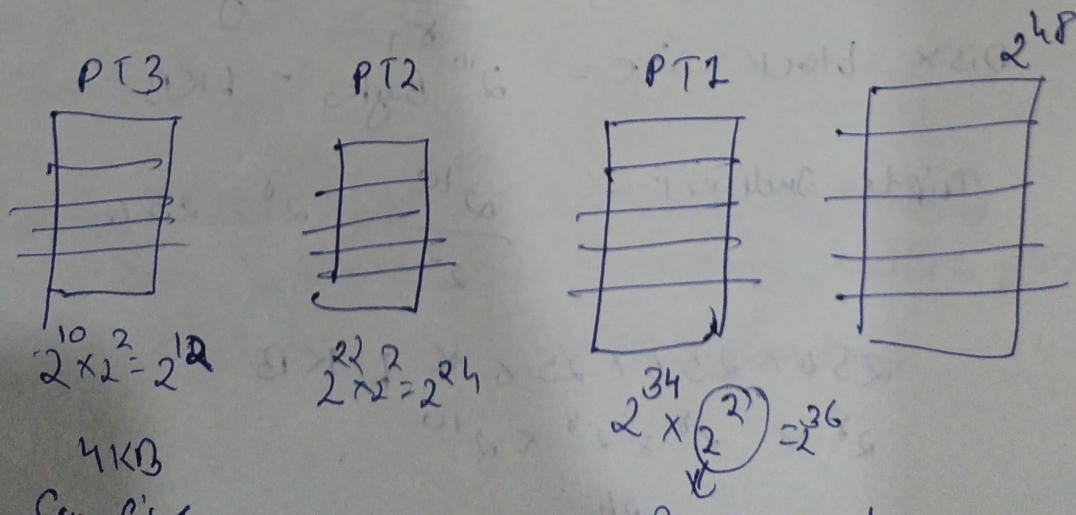
As we can see 36 cannot fit in Page size so we make another Page Table. (2)

NOTE ⇒ Page Table size must fit in Page size.

To fit it $\frac{2^{36}}{2^{14}} = 2^{22}$ level = 2

again = 2^{24} cannot fit so = $\frac{2^{24}}{2^{14}} = 2^{10} \times 2^2$ Now.

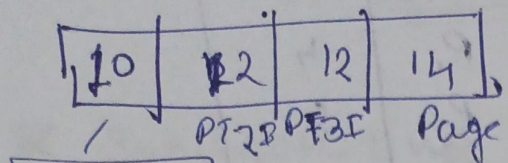
2^{12} can fit in Page size. (single Page)



4KB can fit in Page size

Page Table outer size

address split



Page size = 16KB
PTE = 4B

Entries = $\frac{16KB}{4B} = 4K$

$4K = 2^{12}$

Page Fault Effective memory Access time.

$$EMAT = P[\text{Page fault time} + MA] + (1-P)(MA)$$

↓ when there is Page fault

↓ when there is no Page fault.

$$EMAT = M + P(PS) \rightarrow \text{Page fault service time.}$$

---

* OS Communicates with hardware devices directly.
* A process must have atleast 1 thread.
* Threads provides finer Grains control for tasks, scheduling etc.

The Next CPU Burst time for process when CPU scheduler is shortest process next :-

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

T_n is the predicted value of the last CPU burst (Here $T_n = 5s$), t_n is the (actual) last run time of the process (here t_n is 4s) and $\alpha = 0.8$.

Q $\alpha = 0.5$, $\varepsilon_1 = 5$

$\varepsilon_5 = ?$

Process	BT
P_1	4
P_2	8
P_3	5
P_4	6

$\varepsilon_{i+1} = \alpha A_i + (1-\alpha)\varepsilon_i$

$\varepsilon_5 = (0.5)\,6 + (0.5) \times \varepsilon_4$ Ans = 5.8

$\varepsilon_4 = (0.5) \times 5 + (0.5)\,\varepsilon_3$ 5.625

$\varepsilon_3 = (0.5)\,8 + (0.5)\,\varepsilon_2$ 6.25

$\varepsilon_2 = (0.5)\,4 + (0.5) \times 5$ ($A_5 = \varepsilon_1 = 5$)

$= 4.5$ Putting In above equation

Q Exceptions are Caused by :- ① Page Faults ② when a user program attempted to write to read only Page.

* Interrupts are Caused by Hardware devices. Ex - when devices finishes I/O, timer fires.

Q 50 percent Rule :- 50 percent Rule States that the memory lost to External fragmentation is 50% the size of the allocated memory.