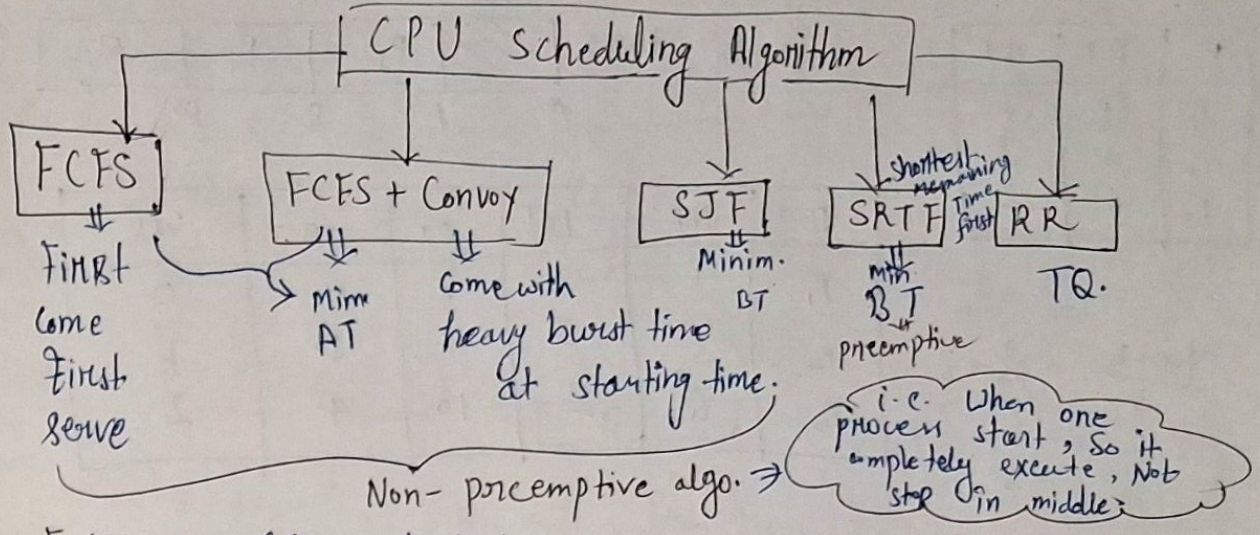
Operating System



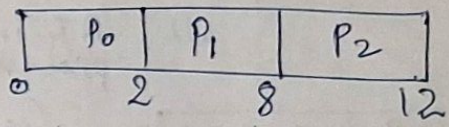
① **FCFS**: Give shortest arrival time as 1st priority.

Ex:-

P	AT	BT
0	0	2
1	1	6
2	2	4

(BT ka Service time also)

Gantt Chart



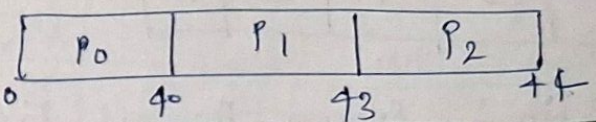
P (Process)	AT (Arrival Time)	BT (Burst Time)	CT (complete time)	TAT (Turnaround time)	WT (waiting Time)
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	10	6

TAT = CT − AT (CAT)

WT = TAT − BT (TAT)

(TWT) Total waiting time = $\frac{0+1+6}{3} = 2.3$ Ans.
(EAWT) Avg. " "

② **FCFS + Convoy**:



P	AT	BT (convoy)
0	0	40
1	1	3
2	1	1

P	AT	BT	CT	TAT	WT
0	0	40	40	40	0
1	1	3	43	42	39
2	1	1	44	43	42

TWT = $\frac{0+39+42}{3}$

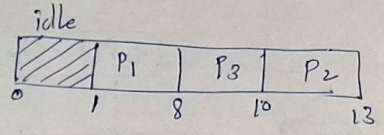
TWT = $\frac{81}{3} = 27$

③ SJF : Get 1st priority of shortest Burst time. but it also check the AT.

Ex:

P	AT	BT
P_1	1	7
P_2	3	3
P_3	6	2

Gantt Chart:



P	AT	BT	CT	TAT	WT
P_1	1	7	8	7	0
P_2	3	3	13	10	7
P_3	6	2	10	4	2

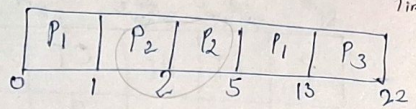
$AWT = \frac{7+0+2}{3} = \frac{9}{3} = 3$

④ SRTF : Shortest Remaining Time first, 1st priority basis on shortest burst time & It is preemptive algo. And also ready queue all the process for the all AT in running process i.e. Active all process using preemptive method. (worked only next arrival Time).

Ex:

P	AT	BT
1	0	~~8~~ 7
2	1	~~3~~ 3
3	2	9

Gantt chart:



P	AT	BT	CT	TAT	WT
P_1	0	9	13	13	4
P_2	1	4	5	4	0
P_3	2	9	22	20	11

$AWT = \frac{4+0+11}{3} = \frac{15}{3}$

AWT = 5unit Ans.

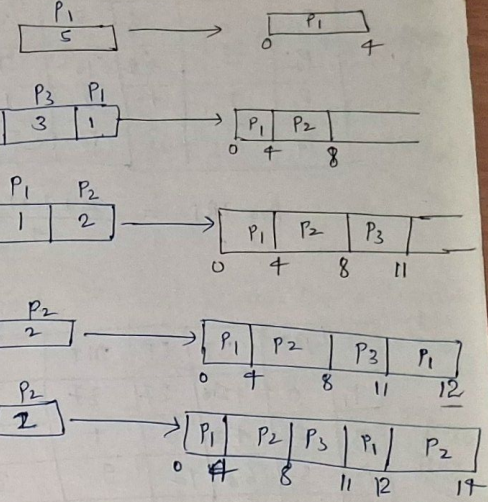
⑤ Round Robin : ~~Process the all~~ Execute the all process in the given Time Quantum only. It is based on the (TQ) basis of shortest Burst time.

Ex:

P	AT	BT
1	0	5
2	1	6
3	2	3

TQ = 4

Ready Queue (RQ) / Gantt Chart



	CT − AT = TAT	CT	WT = TAT − BT
1	12	12	7
2	13	14	7
3	9	11	6

$AWT = \frac{7+7+6}{3} = \frac{20}{3} = 6.6$ unit Ans

* BT = Execut^n Time = Process Time = Service Time

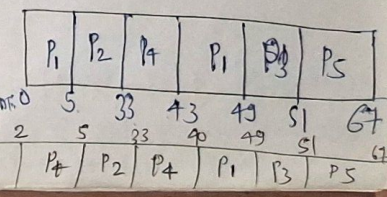
* Starvation creates in SRTF & SJF ⇒ wait for the BT & it is preemptive. → LRTF, LJF } also.

WT	TAT	CT	Process	Arrival Time	BT	Priority
38	49	49	P_1	0	~~11~~ 16	2
0	28	33	P_2	5	~~28~~ 5	0 highest
9	51	51	P_3	12	2	3
31	41	43	P_4	2	~~16~~ 7	1
42	58	67	P_5	9	16	4 lowest

$AWT = \frac{38+0+37+31+42}{5} = 29$ Ans

Solve using preemptive priority scheduling algo.

Gantt Chart:



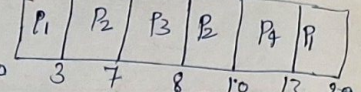
Q.

using SRTF

Shortest Remaining time First

P	AT	BT	CT	TAT	WT
P_1	0	~~10~~ 1	20	20	10
P_2	3	~~6~~ 2	10	7	1
P_3	7	~~1~~ 0	8	1	0
P_4	8	3	13	5	2

Gantt Chart



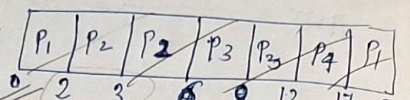
Avg TAT = $\frac{20+7+1+5}{4} = \frac{33}{4} = 8.75$ Ans.

Q.

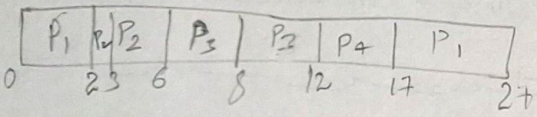
using SRTF

P	AT	BT	CT	TAT	WT
P_1	0	~~12~~ 10	27	27	15
P_2	2	~~4~~ 2	6	4	0
P_3	3	~~6~~ 4	12	9	3
P_4	8	5	17	9	4

Gantt chart:



AWT = $\frac{15+0+3+4}{4} = \frac{22}{4} = 5.5$ Ans.



Disk Scheduling Algorithms

FCFS | SSTF | C-SCAN (lowest) < SCAN < LOOK < C-LOOK (fastest)

* Disk Scheduling : It based on the priority.

① FCFS = First come First serve

② SSTF = Shortest Seek Time first = Find Nearest One = Starvation

③ SCAN = End (Elevator Algo.)

④ C-SCAN = END, BIG

⑤ LOOK

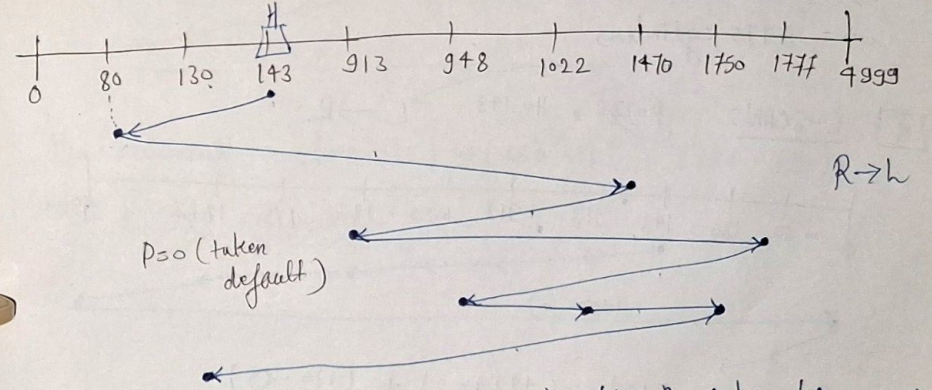
⑥ C-Look = To worlds End

starting pointer

R → L if (previous not given)
R → L By default it will be 0
R → L Direct Always from
R → L Right to left
Else
Left to Right

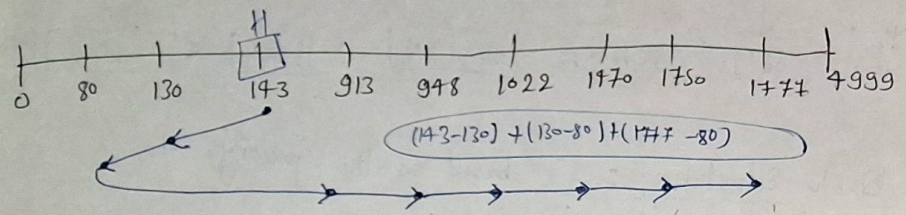
I FCFS : 80, 1470, 913, 1777, 948, 1022, 1750, 130

Header pointer = 143, Total cylinders = ?



Total = 6125 cylinders = (143-80) + (1470-80) + (1470-913) + (1777-913) + (1777-948) + (1022-948) + (1750-1022) + (1750-130).

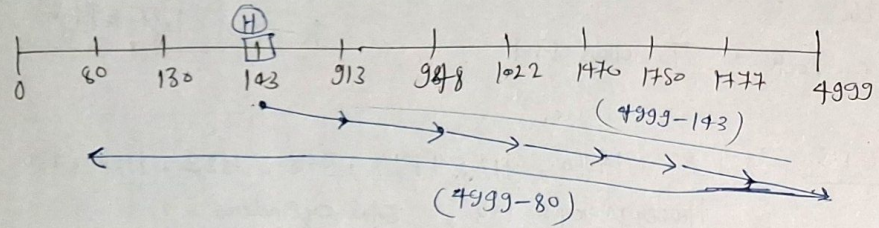
[2] SSTF : R → L & P=0 (taken default).



= (143 - 130) + (130 - 86) + (913 -8) + (948-913) + (1022-948) + (1470 - 1022) + (1750 -1470) + (1777 -1750).

Total Cylinder = 1760 cylinders Ans.

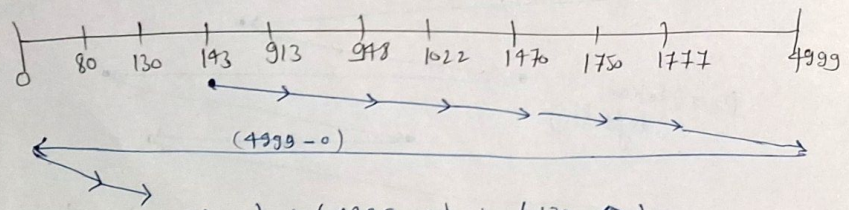
[3] SCAN : Previous = 125 & H= 143. ∴ L → R



⇒ (4999-143) + (4999-80)

⇒ 9775 cylinders

[4] C-SCAN: P=125 , H= 143 L → R

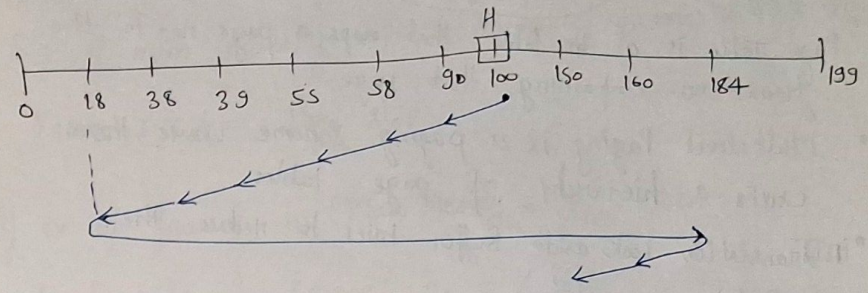


⇒ (4999 - 143) + (4999 - 0) + (130 - 0)

⇒ 9985 cylinders.

(5) C- Look : H=100 , P=0 → R → L (0 - 199)

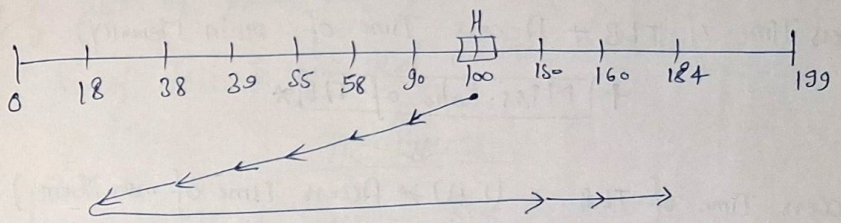
55 , 58 , 39 , 18, 90 , 160 , 150 , 38 , 184



Total = (100 - 18) + (184 - 18) + (184 - 150)

⇒ 282 cylinders Ans.

[6] Look : P=0 ⇒ R → L H=100



Head movement = (100 - 18) + (150 - 18) + (184 - 150)

⇒ 382 cylinders.

PAGING SRP

Unsequencial

- Paging is a non-contiguous memory allocation technique.
- Page Table is a ds table that maps a page no. to the frame no. containing that page.
- Multilevel Paging is a paging scheme where there exists a hierarchy of page tables.
- TLB Translation Look aside Buffer tries to reduce the effective access time
- Page Table is a data structure that performs the maping of page no. to the frame no.

(Effective Access Time) EAT = Hit Ratio * (Access Time of TLB + Access Time of main Memory) + Miss Ratio of TLB * (Access Time of TLB + (L+1) * Access Time of main Memory)

EAT = Hit R + Miss R

Size of Memory = 2^n * Size of one location

(2^n = No. of locatn)

* No. of Enteries in PT = $\frac{2^{\text{Physical MM}}}{\text{page size}}$
* frame size = $\frac{2^{\text{Physical MM}}}{\text{page size}}$

PT = Page Table

For Main Memory

- Physical Address Space = Size of main memory.
- Size of main memory = Total no. of frames * Page size
- Frame size = Page size
- If no. of frames in Main Memory = 2^X, then no. of bits in frame no. = X bits.
- If page size = 2^X Bytes, then No. of bits in page offset = X bits.
- If size of main memory = 2^X Bytes, then No. of bits in physical address = X bits

For Process

- Virtual address Space = Size of process
- No. of pages the process is divided = Process size / page size
- If process size = 2^X bytes, then No. of bits in virtual address space = X bits.

Q. Calculate the size of the memory if its address consists of 22 bits & the memory is 2-bytes addressable.

Sol: Number of location possible with 22 bits = 2^{22} locations (=X)

& size of one locatn = 2 bytes.

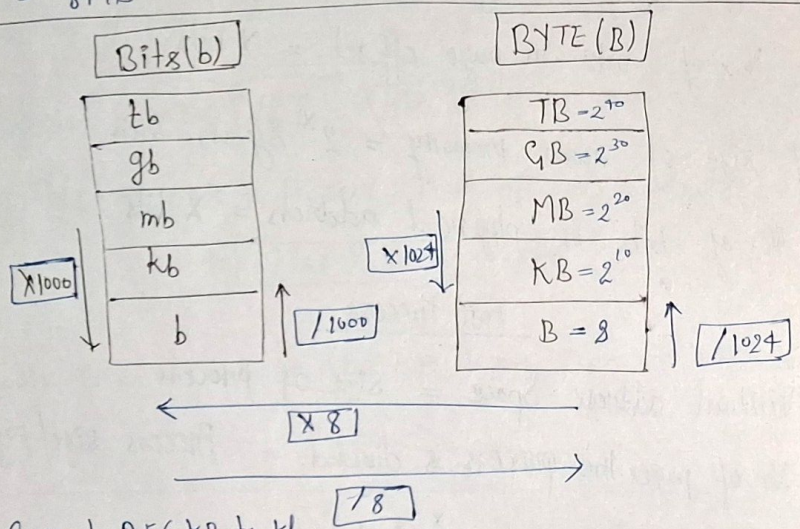
Thus, **Size of memory = No. of locatn * size of 1 location**

$= 2^{22} * 2$ Bytes

$= 2^{23}$ Bytes $= 2^3 * 2^{10} * 2^{10}$ Bytes

$= 8$ MB

SRP



Convert 256 KB to Kb

$= 256 * 1024 * 8 \div 1000$

$= 2097$ Kb Ans

Q. Calculate the size of the memory if its address consists of 24 bits & the memory is 2-byte addressable

Sol: X = 24 & size of 1 locatn = 2-byte

No. of possible locatn = $2^X = 2^{24}$

Size of memory = $2^{24} * 2$-byte = $2^{25} * 2^0 = 2^5$ MB

Size of memory = 32 MB Ans.

Q. Calculate the no. of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

Sol: X = ? & Size of 1 locatn = 4-byte = 2^2

size of memory = 16 GB. = 16×2^{30} bytes.

Size of Memory = $2^{(X=n)} * 4$-bytes

$2^4 * 2^{30} = 2^X * 2^2 \Rightarrow 2^{32} = 2^X$

n = X = 32 bits Ans

Q. ~~Calculate~~ Consider a system with byte-addressable memory, 32-bits logical address, 4 Killobyte page size & page table entries of 4 bytes each. The size of the page table in system in megabytes is 4 MB.

Sol: Given data, $n = 32$ bits (logical address)

Page size = 4KB

Page table entry size = 4 bytes each.

size of page table = ?

Step(1) Page table size = No. of entries in page table * Page table entry size

Step(2) Number of Enteries in Page table —

⇓

No. of pages the process is divided = Process size / page size

Step(3) Process size = No. of bits in logical address = 2^n B

⇓

Process size = 2^n B = 2^{32} B = 4GB

Now,

No. of pages the process is divided = 4GB / 4KB

$= 4 * 2^{30} / 4 \times 2^{10}$

$= 2^{20}$ pages

∴ No. of Entries in page table = 2^{20} entries

Hence, page table size = $2^{20} * 4$-bytes = 4 MB Ans.

Q. In a virtual memory system, size of virtual memory address is 32-bits, size of physical address is 30-bit, page size is 4Kbyte & size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is max. no. of bits that can used for storing protection & other information in each page table entry?

(a) 2 (b) 10 (c) 12 (d) 14

Sol: Size of Main Memory

No. of bits in physical address = 30 bits, Thus

Size of main memory = 2^{30} B = 1GB.

No. of Frame in Main Memory

No. of frame in main memory = Size of memory / Frame size (Page size)

= 1GB / 4KB

No. of frame in main Memory = 2^{18} ~~bits~~

No. of Bits used for ~~Storing~~ other info.

Max. No. of bits that can be used for storing protect & other information = Page table entry size (virtual addr) − No. of bits in frame

→ 32 bits − 18 bits

⇒ 14 bits Ans.

Q. In a paging scheme, virtual m/m address space is 4 KB & page table entry size is 8 Bytes. What should be optimal page size?

Sol: Given,

Virtual address space = 4KB = process size

Page table entry size = 8 bytes.

Optimal page size = $\sqrt{(2 * \text{process size} * \text{Page table entry size})}$

$= (2 * 4KB * 8B)^{\frac{1}{2}} = (2^3 * 2^3 * 2^{10} B)^{\frac{1}{2}}$

$= 2^8$ bytes

Optimal page size = 256 bytes Ans:

Q. Consider a single level-1 paging scheme with a TLB. Assume no page fault occurs. It takes 20ns to search the TLB & 100ns to access the physical memory. If TLB hit ratio 80%, the effective memory access time is _____ msec.

Given, No. of levels of page table = 1

TLB access time = 20ns

Main m/m access Time = 100 ns

TLB Hit Ratio = 80% = 0.8

Sol: Calculating TLB Miss Ratio

TLB Miss Ratio = 1 − TLB Hit Ratio

TLB Miss Ratio = 1 − 0.8 = 0.2

EAT = Hit Ratio * (Access Time of TLB + Access Time of M/M) + Miss Ratio * (Access Time of TLB + (L+1) * Access time of M/M)

(L = level)

$= 0.8 * \{20 + 100\} + 0.2 * \{20 + (1+1) * 100\}$

$= 0.8 * \{120\} + 0.2 * \{220\}$

$= 96 ns + 44 ns$

$\Rightarrow$ 140 ns = EAT Ans:

Effective M/M Access Time (EMA) = Hit * M/M Access Time + Miss * [(L+1) * M/M Access Time]

Avg. Instruction execution time = CPU Time + P * S + (1−P) * EMA

P : Page fault Rate

S : Service page fault

Given data: Size of virtual M/M = 32-b, Page size = 4KB

Total No. of page Table entry = 128 || using 4-way set ass.

TLB Tag = ?

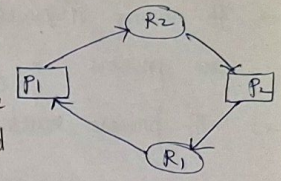
Thus, offset bits = Pagesize = 4KB = $2^2 \times 2^{10} B$ = 12 bits.

Now, No. of bits used for indexing = 32 − 12 = 20 bits.

No. of sets = 128/4 = 32 (requires 5 bits).

Total Tag bits = 20 − 5 = 15 bits / Ans

Deadlock



- The execution of two or more processes is blocked because each process holds some resources & waits for another resource held by some other process.
- Process P_1 holds resource R_1 & waits for Resource R_2 which is held by process P_2.
- Process P_2 holds resource R_2 & waits for resource R_1 which is held by process P_1.

Conditions for Deadlock

1.] Mutual Exclusion –

→ There must exist at least one resource in the system which can be used by only one process at a time. ✌

→ If there exists no such resource, then deadlock will never occur.

→ Ex:- Printer.

2.] Hold & Wait –

There must exist a process which holds some resource & waits for another resource held by some other process.

3.] No preemption –

Once the resource has been allocated to the process,

it can not be preempted.

→ It means resources can not be snatched forcefully from one process & given to the other process.

→ The process must release the resource voluntarily.

4.] Circular Wait –

All the processes must wait for the resources in a cyclic manner where the last process waits for the resources held by the first process.

Note :
- All these 4 cond^n must hold similtaneously for the occurance of deadlock.
- If any these cond^n fail, then the system can be ensured deadlock free.

Deadlock Handling

- Deadlock Prevention (violates 4 necessary conditions)
- Deadlock Avoidence (Banker's Algo.)
- Deadlock detect^n & Recovery
- Deadlock Ignorance.

→ Mutual Exclusion : To violate this cond^n, all the system resources must be such that they can be used in a Sharable mode

→ Hold & wait, process request the resources. Once it has acquired all the resources, only then it can start its execution.

→ No preemption → forcefully preemption ~~is never occurs~~.

→ Circular Wait, A natual no. of is assigned to every resource. Each Process is allowed to request for the resources either in only increasing or only in decreasing order of the resources number.

In case increasing order is followed, if a process requires a lesser no. resources then it must release all the resources having larger number if vice versa.

However, this process may cause starvation but will never lead to deadlock. (starvation = lesser priotizat^n)

Deadlock Handling

Q. A system contains three programs & each requires three tape units for its operation. The minimum no. of tape units which the system must have such that deadlock never arise is _____

Sol: Program = 3 & tape unit = 3 each

$P(T-1)+1 \leq R$

$n \cdot (m-1)+1 \leq R$

Where, n = No. of process / program

T/m = Resources requests made by process / Tape

R = no. of resources

$\Rightarrow 3 \cdot (3-1)+1 \leq R$

$\Rightarrow 7 \leq R$

$\Rightarrow R \geq 7$

Min. no of resource required to avoid deadlock = 7.

Q. Consider a system having m resources of the same type. These resources are shared by 3 processes A, B & C which have peak demands of 3, 4 & 6 respectively. For what value of m, deadlock will not occur? (a) 13 (b) 4 (c) 7 (d) 10

Sol: Total of Max needs $< (m + 3)$ (No. of resources + No. of process)

$(3+4+6) < m+3 \Rightarrow m > 10$. Hence, 'm' should be eleven or above.

So, only 13 qualify the criteria

Q. In a certain OS, deadlock prevention is attempted using the following scheme. Each process is assigned a unique timestamp & is restarted with the same timestamp if killed. Let 'Ph' be the process holding a resource R, 'Pr' be a process requesting for the same resource R & T(Ph) & T(Pr) be their time-stamps respectively. The decision to wait or preempt one of the processes is based on the following algo.

if T(Pr) < T(Ph)
then Kill Pr
else wait.

Which one of the following is TRUE?

(A) The scheme is deadlock-free, but not starvation-free

(B) The scheme is not deadlock free, but starvation-free

(C) The scheme is neither deadlock-free nor starvation-free

(D) The scheme is both deadlock & starvation-free.

Q. A computer system has 6 - tape drives with n - processes computing for them. Each process needs 3 tape drives. The max. value of 'n' for which the system is guaranteed to be deadlock free —

(A) 2 (B) 3 (C) 4 (D) 1

Sol. Max resources required is 3.

$\therefore (3-1) * n + 1 = 6 \Rightarrow n = \lfloor 5/2 \rfloor = 2$

$\Rightarrow \boxed{n = 2}$ Ans.

Q. If there are 6-units of resources R in the system & each process in the system requires 2 units of resources R, then how many processes can be present at max. so that no deadlock will occur?

Sol. $\boxed{n * (m-1) + 1 \geq m}$

n = No. of process = ?

m = No. of resources requests = 2

m = No. of resources = 6

$n * (1) + 1 \geq 6 \Rightarrow \boxed{n = 5}$ Ans.

Q. A system 9 tape drives. The current allocation & max. requirement of tape drives for 3 processes are shown below:

P	Current Allocation	Max. Requirement	need	Available
P_1	3	7	4	2
P_2	1	6	5	3+2=5
P_3	3	5	2	

Which of the best describes the current state of the system?

* Need = Max. Requirement − Current Allocation

* ① Available = Shares tape drives − (current Allocation)

= 9 − (3+1+3) = 2 (P_3 satisfied)

② Available = 3+2 = 5 (P_2 satisfied)

③ Available = 5+1 = 6 (P_1 satisfied)

So, Safe Sequence ⇒ $P_3 \rightarrow P_2 \rightarrow P_1$ (Not deadlock).

Hence, Answer : Safe, Not deadlocked.

* Sum of Max Need ≤ No. of process + Total no. of resources.

Q. For the reference string, how many more page faults occur with FIFO, LRU & optimal page replacement policy?
No. of frames = 3.

Trick

FIFO – First in First out (Left side) → Not stack based Hence
LRU – Least Recently Used (Far from left)
Optimal – Far from right (Stack based Algo.)
* Belady's Anomaly occurs only in FIFO but not always i.e. some times occurs.

* Belady's Anomaly is an abnormal condition which states that if page frames(↑) is increase then page fault(↑) is also increase.

(a) FIFO:

7	0	1	2	0	3	0	4	2	3	0
7	7	7	2	2	2	2	4	4	4	0
	0	0	0	0	3	3	3	2	2	2
		1	1	1	1	0	0	0	3	3
F	F	F	F	H	F	F	F	F	F	F

Page fault = 10
Hit = 1

LRU:

7	0	1	2	0	3	0	4	2	3	0
7	7	7	2	2	2	2	4	4	4	0
	0	0	0	0	0	0	0	0	3	3
		1	1	1	3	3	3	2	2	2
F	F	F	F	H	F	H	F	F	F	F

page faults = 9
Hit = 2

Optimal:

7	0	1	2	0	3	0	4	2	3	0
7	7	7	2	2	2	2	4	2	2	2
	0	0	0	0	0	0	4	4	4	0
		1	1	1	3	3	3	3	3	3
F	F	F	F	H	F	H	F	H	H	H

page fault = 7
Hit = 4

Q. The given program

```
include main()
{
int i;
for (i=0; i<10; i++)
if (i%2 == 0)
fork()
return 0;
}
```

Find the total child Process?

i = 0, 2, 4, 6, 8
Fork().
Fork will be called 5 times
⇒ n = 5
∴ No. of child process = $2^n - 1$
No. of child process = 31
Ans:

Trick

No. of child process = $2^n - 1$
No. of parent process = 1
Total process = 2^n = child process + Parent process

Fork()

→ Fork system call use for creates a new process, which is called child process, which runs concurrently with process.
→ After a new child process created, both processes will execute the next instruct^n following the fork() system call.
→ A child process uses the same pc (program Counter), same CPU registers, same open files which use in the parent process.
→ It takes no parameters & returns an integer value like -1,

→ Below are different values returned by fork():

① Negative Value : Creation of a child process was unsuccessful.

② Zero : Returned to the newly created child process.

③ Positive Value: Returned to parent or caller. The value contains process ID of newly created child process.

* Parent process always one [1].

Q. How many process are created by the program?

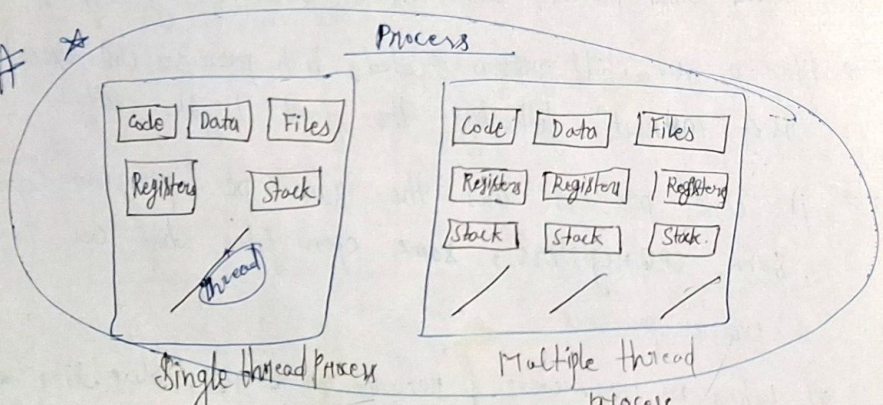
```
int main()
{
int i;
for (i=0; i<4; i++)
fork();
return 0;
}
```

i = 0, 1, 2, 3

Fork will be called 4 times.

Hence, n = 4

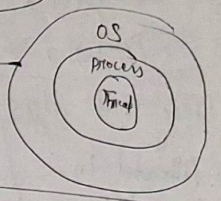
Total process = $2^n = 2^4 = 16$ Ans.



Single thread Process

Multiple thread process.

* On per-thread basis, the OS does not maintain virtual memory state (True stmt).

* User level thread	Kernal level Thread 
→ User thread are implemented by user processes.	→ Kernel threads are implemented by OS.
→ implementation is easy	→ Implementation is complicated due to overhead of mgmt.
→ Context switch time is less	→ Context switch time is more.
→ context switch requires no h/w support	→ H/w support is needed.
→ If one user level thread perform blocking operation then entire process will be blocked	→ If one kernel thread perform blocking operation then another thread can continue execution because it based on the distributed System.
→ Ex:- Java thread, POSIX threads.	→ Ex:- Window solaris.
→ faster	→ Slower because Mgmt overhead.

Create unuse memory space →
- Internal M/M fragmentation creates unuse space in memory
- External M/M fragmentation creates unuse space in disk of memory

Q. Consider 6 Memory Partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, & 250 KB. These partitions needs to be allocated to 4 processes of sizes 357 KB, 210 KB, 468 KB & 491 KB in that order. Perform allocath using :—

① First Fit Algo.
↓
Process Allocated in First partition

② Best Fit Algo. ③ Worst Fit Algo.

Partition Allocated:

357 KB	210 KB	468 KB	491 KB
P_1	P_2	P_3	P_4

M/M:

200 KB	400 KB	600 KB	500 KB	300 KB	250 KB
	P_1	P_2	P_3		

& P_4 is not allocated memory.

② Best Fit Algo: Process allocated in the near size partition first.

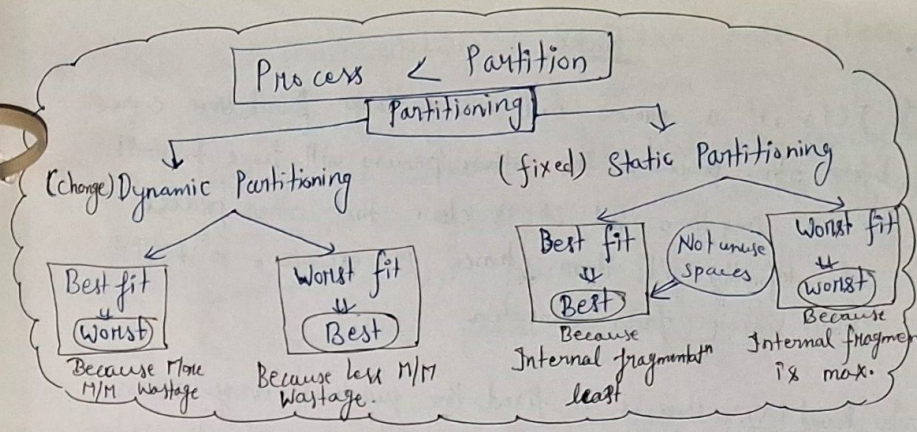
M/M:

200 KB	400 KB	600 KB	500 KB	300 KB	250 KB
	P_1	P_4	P_3		P_2

③ Worst fit Algo: Process Allocated in Largest Partition first

200 KB	400 KB	600 KB	500 KB	300 KB	250 KB
		P_1	P_2		

* P_3 & P_4 is not allocated M/M.



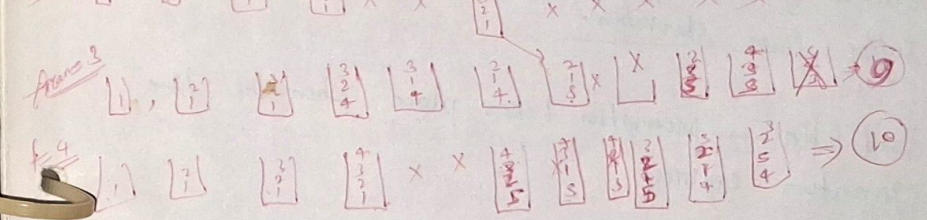
Q. A counting semaphore was initialized to 7. Then 20 P (wait) operations & x V (signal) operations were completed on this semaphore. If the final value of semaphore is 5, then the value of x will be 18.

Sol: $7 - 20 + x = 5 \Rightarrow \boxed{x = 18}$ Ans.

wait ⇒ -20
signal ⇒ +x

5 logical pages order is:

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



Tricks

① In FCFS, if a process with a very large Burst time comes before other process, the other process will have to wait for a long time but it is clear that other process will definitely get their chance to execute, so it will not suffer from starvation.

② In Round Robin, there is a fixed time quantum of every process will get their chance to be executed, So no starvation is here

③ * In Priority based scheduling if higher priority process keep on coming then low priority process will suffer from starvation.

④ * In SJF, if process with shortt process time keep on coming continuously then process with higher burst time will do wait & suffer from starvation.

⑤ * In SRTF, process with shortest burst time will execute first because of this process with high burst time may suffer from starvation.

⑥ Round Robin → preemption takes place when the time quantum expires.

⑦ First in First Out → No preemptn, the process once started completes before the other process takes over.

⑧ * Multi level Queue scheduling — Preemption takes place when a process of higher priority arrives.

⑨ Muti Level Queue scheduling with Feedback — preemption take place when a proces of higher priority arrives or when the quantum of high priority queue expires & we need to move the process to low priority queue.

Scheduling Algo.	CPU overhead (↓) cpu mgmt	Throughput (↑)	TAT (↓) submission of time to complettn time	RT (Response time) (↓)
FIFO	Low	low	High	low
SJF better	Medium	High	Medium	Medium
Priority	Medium	Low	High	High
Round-Robin	High	Medium	Medium	High

* A process has 5 logical pages & accesses them in this order: 1, 2, 3, 4, 12, 5, 1, 2, 3, 4, 5. Which stmt. is/are correct?

(a) The above referenced string suffers from Belady's anomaly when used with FIFO page replacement algo.

(b) The above referenced string generates 9 page faults with 3 frames & 10 page faults with 4 frames, using FIFO page replacement.

(c) The above referenced string suffers from Belady's anomaly when used with LRU page replacement algo.

(d) In LRU, the N most recently used frames are subset of N+1 most recently used frame. So if a page fault occurs with N+1 frame it will have occurred with N frame also.

* Consider a parent process that has forked a child process in the program

```
int a = 50;
int fd = fopen(...);
int ret = fork();
if ( ret > 0){
fclose (fd);
  a=60;
}
else if (ret == 0) {
printf ("a = %d\n", a);
read ( fd , something);
}
```

After the child process is forked, suppose that the child process is scheduled for the 1st ~~time~~ before the child process. Assume that the child process is scheduled for the 1st time only after the parent completes its execution. Now which of following stmt is/are TRUE?

(a) The value of the variable a as printed in the child process is 50

(b) The value of the variable a as printed in the child process is 60

(c) The attempt to read from the file descripton (fd) succeeds in the child.

(d) The attempt to read from the file descripton (fd) fails in the child.

Sol: The fork system call returns 0 to child & any integer > 0 to the parent.

(b) On fork the entire code is cloned in the child. If part is executed by the parent after fork & else part is executed by the child.

The parent makes a=60 & closed file descriptor. The child however has a=50, which is printed after executed. As the file descriptor (fd) is also copied use it even through the parent have closed it.

Important Quest^{n} of OS

Q. Consider the following 2 process synchronization solut^{n}:

Process 0:	Process 1:
Entry: loop while (turn^{0} == 1) // false	Entry: loop while (turn == 0); // true : So it executes in inner again & again
(CS) Critical section	Critical sect^{n}
Exit: turn = 1; // Enter the C.S.	Exit: turn = 0; // Not enter critical sect^{n}

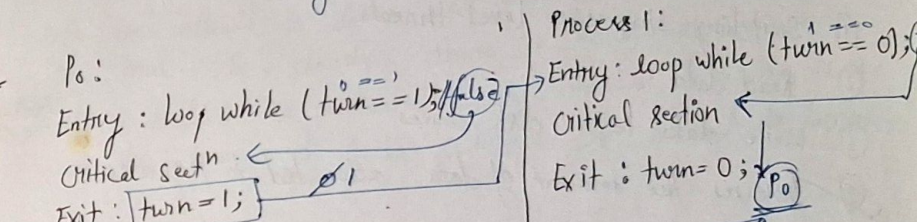
The shared variable turn is intialized to 0. // turn = 0;

Check whether it violates mutual exclusion requirement? Ans: No Any:

Sol: Mutual Exclusion : only 1 process can execute int its critical sect^{n} at any time.

Q. P$_0$:
Entry : loop while (turn^{0} == 1); // false
Critical sect^{n}
Exit : turn = 1;

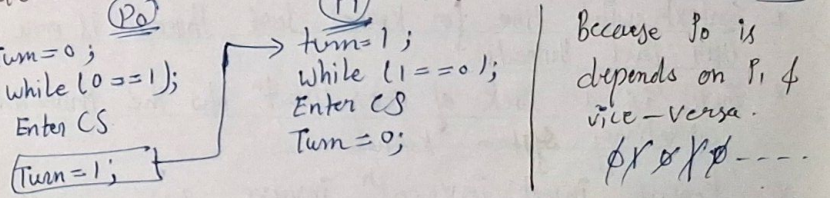
Process 1:
Entry : loop while (turn^{1} == 0); // true
Critical section
Exit : turn = 0;



Check whether it violates bounded waiting requirement? No Ans:

Ans: Bounded Waiting : No process will wait indefinitely.
i.e. there is no violates bounded waiting requirement.

Q. check whether this solut^{n} violates progress requirement? Yes Ans.



Ans: STrict Alteration ⟶ No progress.

for above quest^n: check whether this is correct 2 process synchronizat^n soLut^n — NO — Ans:

Ans: Synchronizat^n solut^n satisfied:
① Mutual Exclusion (✓) ② Bounded waiting (✓) ③ strict Alterat^n (No process ✗)

Hence, the strict alterat^n is not satisfied therfore this is not Synchronizat^n sol^n.

Q: Which of the following connot be done without entering into to ~~Kernel mode~~ M/M ?
① Switching b/w user level threads
② Read data
③ Write data to a disk drives
④ Increases the amount of data allocated to process

Ans: We need to access memory when we increase the amount of data allocated to a process that can only performed in Kernal mode.

* In user level threads if one thread Caccses a page fault, the proces blocks, which is Not in true Kernal level threads.
* Context switch time for kernel level threads is more than user level threads.
* There is a lack of coordinat^n b/w the threads & operating system Kernal.
* Kernal ~~int~~ invocat^n involve system calls therfore, it is considerly slow

Synchronizat^n construct

Q.

Process X
```
while (true)
{
  var P = true;
  while (varQ == true) // True
  {
  /* CS */
  var = false;
  }}
```
↓ Enter CS

Process Y
```
while (true)
{
  varQ = true;
  while (varP == True) // True
  {
  /* CS */
  varQ = false;
  }}
```
↓ enter CS

Check whether the sol^n ~~is~~ exist the ~~critical~~ Mutual Exclusion. No

Ans: When both processes try to enter critical sect^n simultaneously, both are allowed to do so since both shared variable varP & varQ are true. So, clearly, there is No mutual Exclusion.

Also, deadlock is prevented because mutual exclusion is one of the four cond^n to be satisfied for deadlock ~~to happen~~.

Deadlock free

Q Consider a system, where processes need to acquire & hold multiple locks at a time. All processes are mandated to use the funct^n acquire locks, instead of acquiring locks independently.

```
int acquire_locks (struct lock * la[], int n) {
for (int i=0; i<n; i++) {
while (! lock (la[i])); // if lock not available wait
}
return 1;
}
```

Which of the following would prevent a deadlock.

Ans: The above code is already free from deadlock.

OS - Most Expected Questions

* The page size in a system is increased while keeping everything else (including the total size of m/m) the same. Which of the following is/are True as a result of this increase in page size

Ans → Page table size decreases as we have fewer entries.
→ Internal fragmentation of M/M increases
→ TLB hit rate increase because of more coverage of M/M page-frames & internal fragmentation increases as more space is wasted in a page.

* Consider two processes A & B that uses binary semaphore to implement synchronization mechanism on three resources associated with binary semaphores a, b & c. The process A accesses resources as follows: P(a); P(b); P(c);
Which of the following access order by process B will not lead to deadlock?

(i) P(a); P(b); P(c);
(ii) P(a); P(c); P(b);
(iii) P(c); P(b); P(a);
(iv) P(b); P(a); P(c);

* Virtual Memory → Spatial locality
* Shared Memory → Mutual Exclusion
* Look-ahead buffer → Temporal locality
* Look-aside buffer → Address translation
* Critical region → Mutual Exclusion
* Signal/wait → Hoare's monitor
* Working Set → Principle of locality → Cache Memory
* Deadlock → Circular wait
* Buddy System → Memory allocation
* Interpretation → Run-Time type Specification
* Pointer Type → Garbage collection
* Virtual Memory → Segmentation

* "link editor" is a program that Acts as link b/w compiler and user program. [link editor —Program→ b/w compiler & user]

* Locality of reference implies that the page reference being made by a process is likely to be to one of the pages used in the last few page references.

* Thrashing → Implies excessive page I/O

* Dirty bits for a page in a page table, helps avoid unnecessary writes on paging device.

* EMAT = p*s + (1-p)*(m)
p: page fault
s = page fault service time
m = main M/M access time.

* Avg. Memory Access Time = [(% of page miss) * (time of service a page fault) + (% of page Hit) * (memory access time)] / 100

* Length of virtual Address = No. of PTE * Page size (only bits)

* Dirty → Write back policy
* R/W → Page Protection
* Reference → Page replacement ~~policy~~ policy
* Valid → Page initialization

page fault in reverse order = 2n − page frame

~~* No. of pages = No. of page table entry = size of page / entry table~~ NFE.

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

one block = $\frac{2^{\text{pointer}}}{\text{No. of bits}}$

* Gang Scheduling → Thread scheduling
* Rate Monotonic Scheduling → Real-Time scheduling
* Fair Share Scheduling → Guaranteed Scheduling.
* Disk scheduling → SCAN
* ~~Batch~~ Batch processing → FIFO (Queue)
* Time sharing → Round Robin
* Interrupt Processing → LIFO (Stack).
* Thread → CPU
* Virtual Address space → Memory.
* File system → Disk
* Signal → Interrupt.

* Max possible size = [[(Address pointed by doubly indirected block)^2 + (Add. pointed by single add.) + (Address pointed by single direct address)] * [block size]

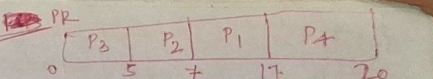
*

AT = 0 ACT = ?

Process	BT	Priority
P_1	10	3
P_2	2	2
P_3	5	1
P_4	3	4

(priority 1 is treated as high priority)

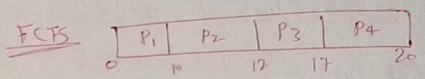
PR



Avg. Complet" Time = ~~17~~ 17 + 7 + 5 + 20 / 4

ACT = 12.750 ($\frac{49}{4}$)

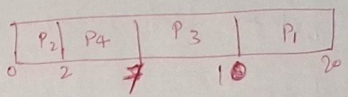
FCFS



ACT = $\frac{10 + 12 + 17 + 20}{4}$ $\frac{59}{4}$

ACT = 14.75

SJF



ACT = $\frac{20 + 2 + 10 + 7}{4}$

ACT = $\frac{39}{4}$ = 9.75

FCFS > PR > SJF ! Ans.

* ~~A real track movement point = 192~~.

* Suppose that the head of a moving-head disk with 192 tracks, numbered 0 to 191, is currently serving a request at track 80 & has just finished a request at track 62. The queue of requests is kept in FIFO order: 119, 58, 114, 28, 111, 55, 103, 30, 75. What is the total no. of tracks traversed by head movements needed to satisfy these request for the FCFS disk-scheduling algo.?

Sol: (119−80) + (119−58) + (114−58) + (114−28) + (111−28) + (111−55) + (103−55) + (75−30) = 547 Ans.

* A computer whose processes have 1024 pages in their address spaces keeps its page tables in memory. The overhead required for reading a word from the page table is 5 ns. To reduce this overhead, the computer has a TLB, which holds 32 (virtual page, physical page frame) pairs

& can do a lookup in TLB. What hit rate is needed to reduce the mean overhead to 2ns?

Ans: The Effective Memory access time = Time for VA to PA translation + Actual page access

Here, the overhead means time for VA to PA translation

∴ Mean overhead = Hit × (TLB lookup time) + Miss (TLB lookup + MAT)

⇒ $2 = h \times 1 + (1-h)(1+5)$

⇒ $\boxed{h = 0.80}$ ans

* Consider a system with 8-bit virtual & physical addresses, & 16 byte pages. A process in this system has 4 logical pages, which are mapped to 3 physical pages as follows:

page Table

Page	Frame
0	6
1	3
2	11
5	*

TLB

Index	Frame
0	6
2	11

Which of the following is/are TRUE?

(A) CPU access to virtual address 7 leads to a TLB hit

(B) " " " " " 20 " " " " "

(C) " " " " " 70 " " " " trap to OS

(D) " " " " " 80 " " " " page fault.

Sol: Virtual address 7 = 0000 (page No.) + 0111 (offset) = logical page 0 hit

Virtual address 20 = 0001 0100 = logical page 1. TLB miss

" " 70 = 0100 0110 = logical page 4. TLB miss

MMU access page table & discovers it is an invalid entry. MMU raises a trap to the OS. Virtual address 80 = 0101 0000 = logical page 5. TLB miss

MMU accesses page table & discovers page not present.

MMU raises a page fault to the OS.

* Consider the following code in which item=0, A & B are shared variables. Pick the correct choice from the given options.

Process A

```
int A=1;
while (1){
if (B==0){
if (item==0){
item = item+1;
}
}
A=0;
}
```

Process B

```
int B = 1
while (1){
if (A==0){
if (item==0){
item = item+1;
}
}
B=0;
}
```

(A) The Above code ensure mutual exclusion on shared variable item

(B) The " " does not ensure " " " "

(C) The above code ensures bounded waiting

(D) None of the above.

Since, the OF stage of I_2 is at t_{12}, the ID stage of I_3 is also at t_{12} because for a given instruction to enter a particular stage the preceding instruction has to enter the next stage.

Q. The expression for a machine which is having m-bit instructions that can contain zero-address, one address, two-address instruction. Let the memory be having 2^n words. x, s, t denoting no. of zero, one, two instruction respectively.

Ans: $2^m = x + s*2^n + t*2^{2n}$

$t*2^{2n}$ → 2-address instruct^n

$s*2^n$ → One-address instruct^n

x → zero address instruct^n.

I_0 : DIV R_2, R_0, R_1 (The operat^n is $R_2 \leftarrow R_0/R_1$)

I_1 : MUL R_5, R_3, R_4 (" " " $R_5 \leftarrow R_3 * R_4$)

I_2 : SUB R_2, R_5, R_2 (" " " $R_2 \leftarrow R_5 - R_2$

I_3 : ADD R_5, R_1, R_6 (" " " $R_5 \leftarrow R_1 + R_6$)

If there is no operand forwarding in the pipeline, what is the No. of clock cycles needed to execute the above sequence of instructions?

t_1	t_2	t_3	t_4	t_5 t_6	t_7 t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}
I_0:	ID	OF	PO	PO PO	PO WO									
	IF	ID	OF		PO	PO	PO	WO						
		IF	ID						OF	PO	WO			
			IF						ID			OF	PO	WO

Between I_0 & I_1 there is no dependency. So they OF stage of happens at the same time when PO stage of I_0 is going on at t_4. And once PO stage of I_0 is completed at t_9 the PO stage of I_1 starts from t_5.

But b/w I_1 & I_2 there is dependency so OF stage of I_2 has wait till WO stage of I_1 is over. because there is no operand forwarding. Similary there is dependency b/w I_2 & I_3 the OF stage of I_3 can start only after WO stage of I_2.

So total no. of clock cycle is needed (17)

Read operations are 60%. that means write operations are 40%.

$T_{avg} = h_1 t_1 + (1-h_1) * t_2 + t_1$ [If nothing is mentioned, we have to go with the default case ie. hierarchical access]

$= 0.9 \times 20 + 0.1 * (140 + 20)$

$= 34$ ns [It is the avg. time required for read operation]

In write through updation both the m/m & cache are updated simultaneously. If we consider the write operation is complete when data is written to both cache & m/m. Because of that we take the max. of these two, which is m/m access time as the time for write operation.

So, Tavg. for write = 140 ns.

$T_{avg}w = 0.6 * 34 + 0.4 * 140$ m/m

(Read time, write)

$T_{avg}w = 76.4$ ns Ans.

* A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) & write operation (WO) stages. The IF, ID, OF, & WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD & SUB instructions, 3 clock cycles for MUL instruction & 4 clock cycles for DIV instruction respectively.

Alternate Method

New CPI = (1 + stall frequency * stall cycle) clock cycle/ins

$= (1 + 0.2 * 3) = 1 + 0.6 = 1.6$ Ans.

* A computer has a 256 KByte, 4-way set associative, write back data cache with block size of 64 Bytes. The processor sends 32 bit addresses to cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit. The size of tag directory is

16	10	6
tag	Set No.	Bo

Ans: CS = 256 KB, block size = 64 B.

So, No of block in the cache = 256 KB / 64 B = 4K.

It is a 4-way set associative cache. Each set has 4 blocks.

So, No. of sets in cache = 4K/4 = 1K = 2^{10}

Tag = 32 − 10 − 6 = 16 Ans.

Now, Given additional data, 2 valid bits & 1 modified bit.

So, size of each tag entry = 16 + 2 + 1 = 19 bits.

Size of cache tag directory = (No of tag entries) · (size of each tag entry)

= 4K * 19 bits Ans.

* Consider a system with cache access time 20 ns & main memory access time 140 ns. If 60% operations are read operations & hit ratio is 90%. What is the effective access time if write through updation technique is used.

Sol: Given data.

Cache access time = 20 ns

M/M access time = 140 ns

hit ratio = 90% = 0.9

* Consider a microprogrammed control unit which supports 128 instructions, each of which on an avg. takes 10 micro-operations. The system supports 16 flag conditions & two groups of control signals which are a total 50 control signals. Group-1 has to generate one or none of 30 control signal & group-2 can have at most 20 from the remaining control signals. What are the no. of bits required to represent the flag conditions, control signal respectively in the control word?

Ans:

f	G_1 (30)	G_2 (20)
log(16)	log(30)	At most -20
(4)	(5)	20 Ans:

→ Since there are 16 flags, we need log(16) = 4 bits flag conditions Ans:

→ total 50 control signal (one/more → 30 → G_1; almost → 20 → G_2)

→ If has to generate one or none of the 30 signals of Group-1 So, we can use vertical microprogramming for these and use log(30) = 5 bits for G1.

→ At most 20 of the G2 can be active. So for them we need to use horizontal microprogramming there & we need 20 bits.

→ So, in total for the control signal = 5+20 = 25 bits Ans

* Consider a pipeline system that overlaps all kinds of instructions except branch instructions. Each branch instruction introduces 3 stall cycles. If there exists 20% branch instructions what will be new CPI

Sol:

$$\frac{80}{100}(1) + \frac{20}{100}(3) = 0.8 + 0.6$$

$$= 1.4 \text{ Ans.}$$

Cap ## comparators = line No. (Associative).

Line No * BO = CS

K → comparators.

K-way | K=1 | K=N

Set-Associative | Direct mapping | Complete Associative

CO Expected Questions

(a) List 1

1. Implied Mode
2. Indirect addressing mode
3. Auto-increment addressing mode

List 2

① MUL R1, @(R2) // $R_1 \leftarrow R_1 * M[R_2]$

② SUB R1, +(R2) // $R_2 \leftarrow R_2 + 1$

③ INCA (increment accumulator)

* Consider a disk with each sector of 512 bytes, 1000 tracks per surface, 50 sectors per track, five-double sided platters & avg. seek time of 5ms. If T is the capacity of a track in bytes & S is the capacity of each surface in bytes, then (T,S)

Sol: T = Size of a track in bytes = No. of sectors per track * size of each sector

= 50 * 512 = 25KB.

S = No. of tracks per surface * no of sectors per track * size of each sector

= 1000 * 50 * 512 bytes = 25,000 K bytes.

* Choose correct stmt / regarding RISC & CISC processors from the below:
• CISC are generally not highly pipelined.
• RISC have hard-wired control units • CISC has microprogrammed
• RISC processors are pipelined to achieve avg. clocks / instruct^n around 1

Ans:

* Transfer data = $\frac{\text{Track storage capacity} * \text{No. of tracks/Rev.}}{\text{Rotation in 1 sec.}}$

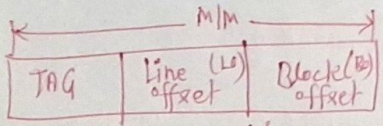
* Rotation in 1 sec = $\frac{60}{\text{Rotation speed}}$

* DMA I/O → Disk
* Interrupt I/O → Printer

* cache → High speed RAM
* Condition Code Register → ALU

* I/O redirection implies connect" 2 programs through a pipe
* Formatting of floppy disk refers to writting identificat" information on all track & sectors.
* characteristics of RISC processors are
  ① Relatively few instruct" ② More registers
  ③ Relatively few addressing Mode
  ④ Hardwired rather than micro-programmed control.
* The swap space in the disk used for saving process data.
* Block transfer & Polling interrupts, DMA transfer mode & interrupt handling mechanism will enable the highest I/O bandwidth.
* Signal processing system is a most likely candidate example of a pipe & filter architecture.

Direct Mapping

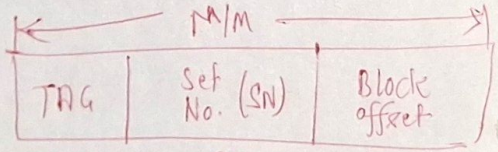


Line offset (LO) = $\frac{CS}{BS}$

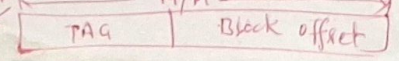
TAG = M/M − (LO + BO)

TAG = M/M − BO

n-way Set Associative [BO = BS]



Set No. = $\frac{CS}{BS * n\text{-set}}$ {2-way set, 4-way set



* Based & Relative addressing modes are suitable for program "relocat" at run time.
* k stage pipeline can process n tasks in T_k time

$$T_k = [(k + (n-1)]\tau$$

where, $\tau = T_m + d$

T_m = max. delay (

d = delay

n = process time

* Relative addressing → indexing of array elements with same instruction
* Relative Addressing can't be faster than absolute addressing as absolute addressing must be calculated from relative addr.
* Register renaming done in pipelined processors to handle certain kinds of hazards.
* Indirect Addressing → Passing array as parameter
* Indexed " → Array implementat"
* Base Reg. " → Writing relocated code.
* Absolute addrsing mode, the address of operand is inside the instruction
* Horizontal µp
  - don't require use of signal decoders
  - Uses one bit for each control signal
  - Results in larger sized microinstructions than vertical µp.
* The data transfer b/w memory & I/O device using interrupt driven I/O is faster than programmed I/O. Because programmed I/O technique require constant monitoring on peripheral by CPU.
* Conditional CALL or JUMP instructions only test flags, they do not affect any of the flag value.
* Avg latency (R) = $\frac{1}{2}$ * Rotation time.

COA

* Avg. Access Time (t_{avg}) = η * Secondary Memory access time

* In case of Hit ratio, H

$$t_{avg} = H * t_{a_1} + (1-H) t_{a_2}$$

Where, t_{a_2} & t_{a_1} = M/M Access time

* The block K of the M/M maps to the sets (K mod c) of the cache where, c = No. of sets.

* More than one word are put in one cache block to exploit the spatial locality of reference in a program.

* V.M. increase the degree of multiprogramming & context switching overhead.

* Avg. Access time of system ignoring the search time within the cache,

$$t_{avg} = t_1 h_1 + (1-h_1) h_2 t_2 + (1-h_1)(1-h_2) t_m$$

Where, t_1, t_2 ⇒ Access time of Level1 cache, Level2 cache

t_m ⇒ main memory access time

h_1 & h_2 ⇒ Hit rates of Level1 & Level2 caches.

* No. of bits needed for cache indexing = (cache size − Block Size)

* No. of block in a cache = capacity / block size [BS = BS]

* Tag Memory Size = No. of sets * No. of lines in a set * No. of tag bits;

* Indirect addressing → Pointers [int temp = *x]

* Immediate Addressing → Constants

* Auto decrement Addressing → Loops [while [*A++]]