Operating System:

Teaching Schedule:

I. Introduction and Background

II. Process Management
- 40% Imp
  - → CPU Scheduling
  - → Synchronization
- → Concurrent Programming
- → Deadlocks
- → Threads

III. Memory Management
- → RAM Chip Implementation
- → Loading, Linking and Address Binding
- 40% Imp
  - → Techniques
    - → Paging
    - → Multi level paging
    - → Inverted paging
    - → Segmentation
    - → Segmented paging
  - → Virtual Memory

IV. File Systems

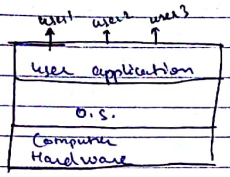
Textbooks:

1. O.S. by Galvin
2. Modern O.S. by A.S. Tenenbaum
3. O.S. by W. Stallings

Faculty ⇒ Balakrishna

Introduction and Background:

→ Operating System is an interface between user and computer hardware.




```
main()
{
  intr;
  printf("Hello");
}
```

→ Internally printf calls write() system call in order to communicate with the monitor.

* System Call: It is a request made by the user program to the O.S. in order to get any kind of service.

→ O.S. is also called as resource allocator because it is responsible of allocating the resources of a computer.

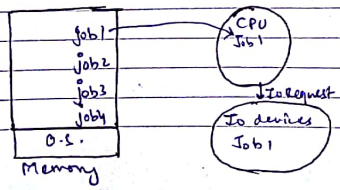
Resources
- Hw Type — Eg. devices, memories.
- Sw Type — Eg. files

* Goals of O.S.:

→ The primary goal is convenience (Easy to use)
→ The secondary goal is efficiency.

* Types of O.S.:

(1) The Batch O.S.:



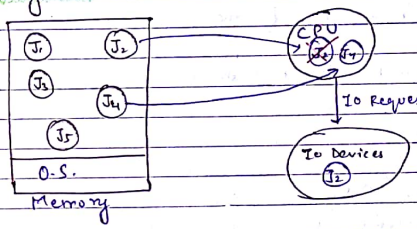
Time
- CPU Time
- I/O Time

→ If the job is completely completely, then only another job will be scheduled onto the CPU.
→ Increased CPU idleness.
→ Decreased throughput of the system.

* Throughput: No. of jobs completed for unit time is known as throughput of the system.

E.g. IBM OS/2.

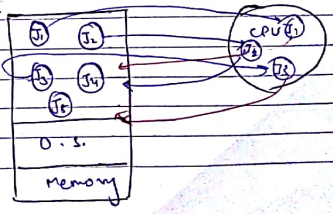
(2) Multi Programming O.S.:



→ If the job is leaving the CPU to perform IO operation, then another job which is ready for execution will be scheduled on to the CPU.
→ Increased CPU utilization.
→ Increased throughput of the system.

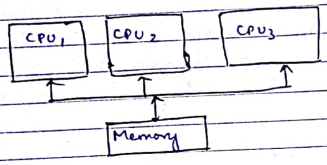
E.g. Window's, UNIX, LINUX

(3) Multi tasking O.S.



→ Multitasking is an extension of multiprogramming O.S.
→ The jobs will be executed in the time sharing mode.

(4) Multiprocessor O.S.:

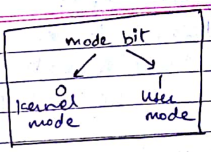


(i) Increased throughput of the system.
(ii) Reliability.
↳ Fault tolerant systems
(iii) Economical
E.g. UNIX.

(5) Real Time O.S.:
→ The systems which are strict deadly time bound.
→ Categorised as: Hard RTOS and Soft RTOS.
↳ Eg. Satellite systems, Missile system
↳ Eg. Banking sector

Eg: Vx Works, RTOS.

Dual mode Operation:



→ In the hardware level, instructions are executed in the 2 different modes:
① User mode / non-privileged mode
② Kernel mode / privileged mode / System mode / Supervisory mode
→ The dual mode operation is used in order to provide protection and security from errant users.
→ It is purely the decision of the O.S. in which particular mode, the instruction has to be executed.
→ Generally, the privileged instructions are executed in the kernel mode and non-privileged instructions executed in the user mode.
→ The mode bit is used to identify, in which particular mode, the current instruction is executing.
→ In the boot time, system always starts in kernel mode.
→ The O.S. always runs in kernel mode.

NOTE
↳ Mode switching takes very less time compared to the process switching.

* Privileged instructions:
① I/o operation
② context switching
③ Disabling interrupts
④ Changing the memory map
⑤ Clearing the memory map.
⑥ Set the time of clock.

* Non Privileged Instructions:

① Reading the status of the processor.
② Reading the time of clock.
③ Sending the final print output to the printer.

Layered Approach:

Process Mgmt.	L_1
Memory Mgmt.	L_2
File Mgmt.	L_3
Device Mgmt.	L_4
Protection & security	L_5

O.S.

→ The design and implementation of the O.S. will be done in multiple layers.
→ The main advantage of layered approach is modularity, abstraction and debugging.

→ It means that if any specific layer is updated, then, it will not have any affect or impact on other layers.

FORK() SYSTEM CALL IMPLEMENTATION

```
main()
{ int pid;
  pid = fork();
  if (pid < 0)
  [{ printf("child process creation failed");
    }
  else if (pid == 0)
  [{ printf("child process");
    }
  else
  [{ printf("parent process");
    }
}
```

→ Fork() is a system call used to create the child process.
→ Fork() returns negative value if the child process creation is unsuccessful.
→ Fork() returns value '0' to the newly created child process.
→ Fork() returns positive value to the parent process (i.e., process id of child process).
→ When the child process is created by using fork() system call, the new memory location will be allocated to the child process.
→ The child and parent process will have same virtual address but physical addresses will be different.

Eg. ①

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

P

```
main()
printf("Hello");
}
```

C_1

1 child
2 times Hello.

Eg. ②

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

P

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

C_1

```
main()
{
  printf("Hello");
}
```

C_2

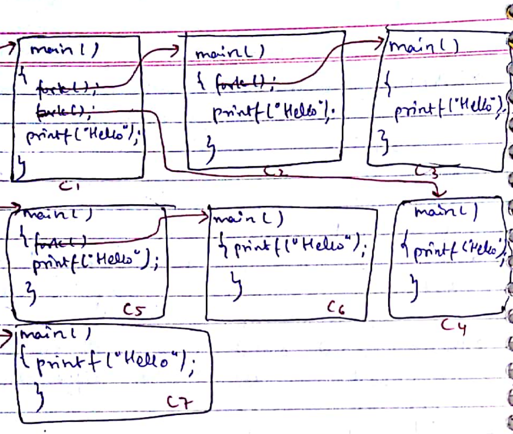
```
main()
{printf("Hello");
}
```

C_3

3 children
4 times Hello.

Program is passive & static.

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



7 children.
8 times Hello.

NOTE
↳ If the program is having "n" forks calls, then, it will create $2^n - 1$ child processes.

WB. Pg. 25 Q.13.

```
if (fork()==0)
{ a=a+5;
  printf(a, &a);
}
else
{ a=a-5;
  printf(a, &a);
}
```

child → a = 25
parent → a = 15
let a = 20.
10 = u+10
v≠y
(d)

Process Management

* Process Concepts:

Def: The program under execution is called as process.
- ① It should reside in the main memory.
- ② It is occupied the CPU to execute the instructions.
- ③ Active and dynamic

→ Process will have various attributes.
1. Process id
2. Process state
3. Priority
4. Program counter
5. General Purpose Registers
6. List of open files
7. List of open devices
8. Protection information

Context ⇓ stored PCB

(i) Process id : It is the unique identification number which is assigned by the O.S. at the time of process creation.

(ii) Process State : contains current state information of a process where it is residing.

(iii) Priority : It is a parameter which is assigned by the O.S. at the time of process creation.

(iv) Program Counter: contains the address of the next instruction to be executed.

⇒ All the attributes of the process is called as context of the process.

→ The context of the process will be stored in the ~~poor~~ PCB (Process Control Block)

* PCB.

Pid	P.S
P.C.	Priority
U.R	L.O.F
L.O.D	Interrim info

→ Every process will have its own PCB.
→ PCBs of the processes will be stored in the main m/m.
→ PCBs of the processes will be implemented by using double linked list.

States of a Process:
(1) New
(2) Ready
(3) Run
(4) Wait or Block
(5) Termination or Completion
(6) Suspend Ready
(7) Suspend block or Suspend wait

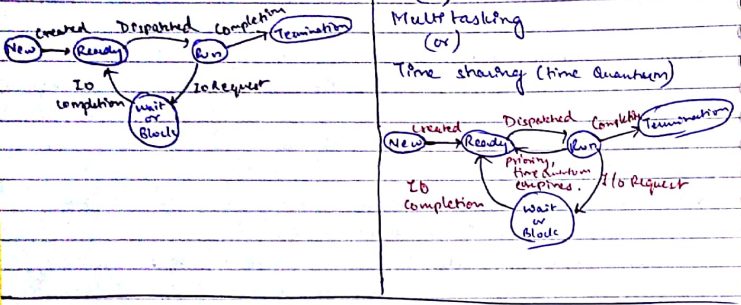
Operations:
(1) Creation
(2) Scheduling
(3) Dispatching
(4) Execution
(5) Killing or Termination
(6) suspending
(7) ~~Rota~~ Resuming

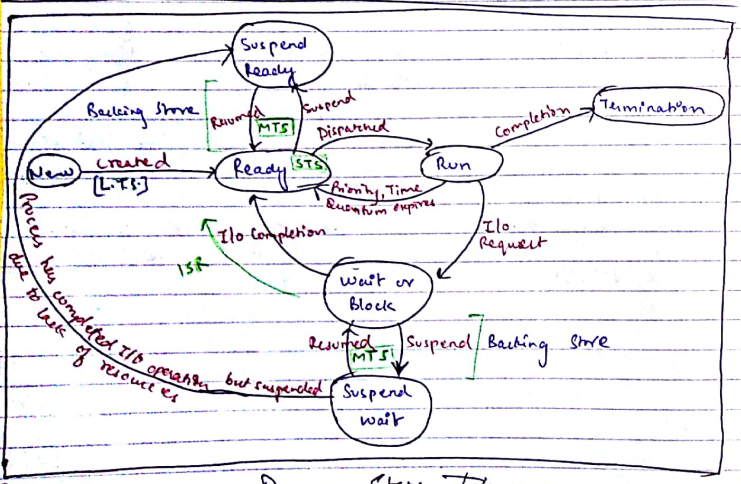
* ~~Operations~~:

Process State Diagram:

→ Initially Process will be in the new state, it means the process is under creation or process is being created.
→ In the Ready state, there will be multiple number of processes, and one of the process will be selected and that will be dispatched on to the running state.
→ In the Running state, process has occupied the CPU, and executing the instructions of the process and performing CPU time.
→ In the Running state, there will be only one process at any point of time.
→ If the running process requires IO operation, it will be moved out to wait / blocked state.
→ In the wait state also there will be multiple number of processes, it means multiple processes will perform IO operation simultaneously.
→ When the process is in the ready, running and block states, it is residing in the main memory.
→ If the resources are not sufficient to manage the processes in the ready state, then some of the processes will be suspended and they will be moved on to suspend ready state.
→ When the process is in the suspend ready, then it is residing in the secondary memory. [Backing store]

Multiprogramming:



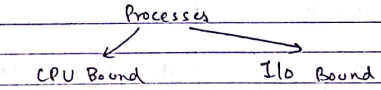


Process State Diagram

Degree of Multiprogramming:

→ The number of processes present in the main memory at any point of time is called as degree of multiprogramming.

⇒ Processes are classified as follows:



* CPU Bound Processes: Processes which require more amount of CPU time is called as CPU bound processes.

→ These processes will spend more time in the running state.

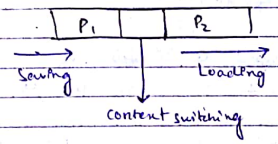
* I/O Bound Processes: The processes which require more amount of I/O time are called as I/O Bound processes.

→ These processes will spend more time in the wait/block state.

→ Each and every time, when a process is moving from one state to other state, the context of the process will change.

⇒

* Context Switching.



→ Saving the context of one process and loading the context of another process is called as context switching.

→ If the context of the process is more, then context switching time will also increase, which is undesirable.

NOTE
↳ If the context switching is disabled, then the process is not allowed for preemption.

→ Context Switching time is considered as a overhead for the system.

NOTE
↳ In some special cases, if there is only one process, still it is called as context switching.
E.g. Round robin scheduling with 1 process.

⇒ In the O.S., ~~there~~ there are 3 different schedulers:

(1) Long Term Scheduler (L.T.S.) (or) Job Scheduler:
→ LTS is responsible of creating and bringing the new processes into the system.

(2) Short Term Scheduler (S.T.S) (or) CPU Scheduler:
→ STS is responsible of selecting one of the process in the ready state for scheduling onto the running state.

(3) Mid Term Scheduler (MTS) (or) Medium Term Scheduler:
→ MTS is responsible of suspending and resuming the processes.
→ The job done by mid term scheduler is called as swapping.

* Dispatcher: Dispatcher is responsible of loading the selected job on to the CPU.
→ It is also responsible of performing context switching.

* The LTS should select good combination of both CPU bound and IO bound processes in order to get good throughput for the system.

⇒ Long Term Scheduler controls degree of multiprogramming.

Q. Consider a system which has "n" CPU processors. Then, what is the minimum and maximum no. of processes that may present in the Ready, Running and Blocked State.

	min	max
Ready	0	Any → depends on max^m degree of multiprogramming of the system
Run	0	n
Blocks	0	Any

→ Process is having various different times.

(1) Arrival Time [A.T.]: The time when the process is arrived into ready state.

(2) Burst Time [B.T.]: The time required by the process for its execution.

(3) Completion Time [C.T.]: The time when the process is completing its total execution.

(4) Turn around Time [T.A.T.]: The time difference between completion time and arrival time.

T.A.T. = C.T. − A.T.

(5) Waiting Time [W.T.]: The time difference between T.A.T. and B.T.

W.T. = T.A.T. − B.T.

(6) Response Time [R.T.]: The time difference between first response and arrival time.

CPU SCHEDULING:

Where: Ready State
Who: Short Term Scheduler
When: (i) Run → Termination
Run → Wait
Run → Ready
(ii) Wait → Ready
(iii) New → Ready

Goal: ① To maximize the CPU utilization and throughput of the system.
② To minimize the average T.A.T., avg W.T. & R.T. of the processes.

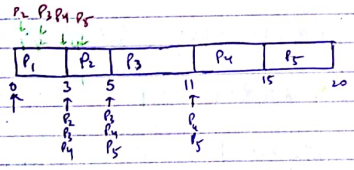
#FCFS [First Come First Serve]:

Criteria: Arrival Time
Mode: Non-preemptive

Q.

P.No	AT	BT
1	0	3
2	1	2
3	2	6
4	3	4
5	4	5

Gantt Chart:



Avg TAT?
Avg W.T.?

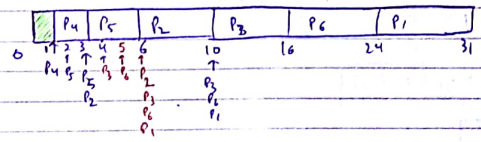
$P_1 \Rightarrow TAT = 3$ $WT = 0$
$P_2 \Rightarrow TAT = 4sec$ $WT = 2sec$
$P_3 \Rightarrow TAT = 9$ $WT = 3$
$P_4 \Rightarrow TAT = 12$ $WT = 8$
$P_5 \Rightarrow TAT = 16$ $WT = 11$

Avg TAT = 8.8
Avg WT = 4.8

Q.

P.No	A.T.	B.T.	T.A.T.	W.T.
1	6	7	25	18
2	3	4	7	3
3	4	6	12	6
4	1	2	2	0
5	2	3	4	1
6	5	8	19	11
			69	39

Avg:
TAT = 11.5
WT = 6.5

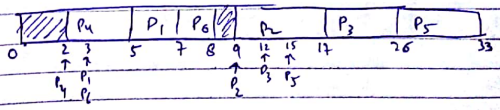


NOTE
→ If the arrival times of the processes are matching, then we schedule the process which has lowest process id.

Q.

P.No	A.T	B.T	TAT	WT
1	3	2	4	2
2	9	8	8	0
3	12	9	14	5
4	2	3	3	0
5	15	7	18	11
6	8	1	5	4
			52	22

Avg TAT = 8.67
Avg WT = 3.67



Q5

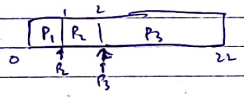
P.No	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	1	21	20	19
3	2	1	22	20	19

Avg WT = 38/3 = 12.67

Q5.

PNo	AT	BT	TAT	WT
1	0	1	1	0
2	1	1	1	0
3	2	20	20	0

Avg WT = 0



* Convoy Effect: In FCFS, if the first process is having CPU bound process and followed by many I/O bound processes, then it will have major effect on average waiting time of the processes.
This effect is called as Convoy Effect.

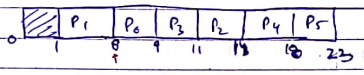
Shortest Job First (SJF)

Criteria : Burst Time
Mode : Non Preemptive SJF

Q5.

P.No	AT	BT	TAT	WT
1	1	7	7	0
2	2	3	12	9
3	3	2	8	6
4	4	4	16	10
5	5	5	18	13
6	6	1	3	2

Avg TAT = 62/6 = 10.33
Avg WT = 40/6 = 6.67

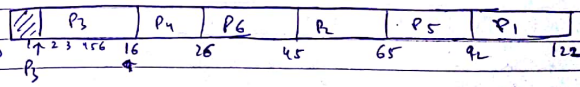


NOTE
↳ If the burst times of the processes are matching then schedule the process which has lowest A.T.

Q5.

P.No	AT	BT	TAT	WT
1	2	30	120	90
2	3	20	62	42
3	1	15	15	0
4	4	10	22	12
5	5	27	87	60
6	6	19	39	20
			345	224

Avg TAT = 57.5
Avg WT = 37.33



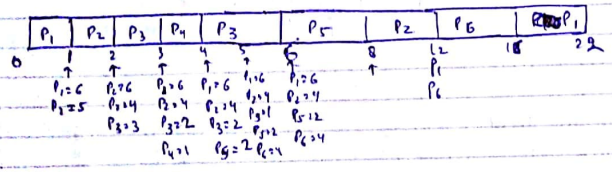
Shortest Remaining Time First (SRTF):

Criteria: Burst Time
Mode: Pre emptive

Q.

P.No.	A.T.	B.T.	TAT.	WT
1	0	7	22	15
2	1	5	11	6
3	2	3	4	1
4	3	1	1	0
5	4	2	4	2
6	5	4	11	7

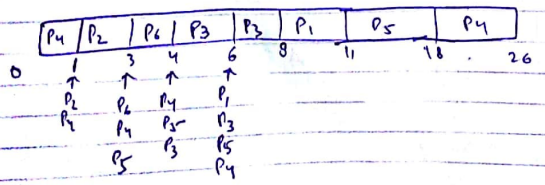
TAT = 53/6 = 8.83
WT = 31/6 = 5.16



Q.

P.NO	AT	BT	TAT	WT
1	6	3	5	2
2	1	2	2	0
3	4	4	4	0
4	0	9	26	17
5	3	7	15	8
6	2	1	2	1

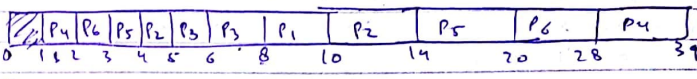
Avg TAT = 54/6 = 9
Avg WT = 28/6 = 4.67



Q.

PNo	AT	BT	TAT	WT
1	6	2	4	2
2	4	5	10	5
3	5	3	3	0
4	1	12	38	26
5	3	7	17	10
6	2	9	26	17

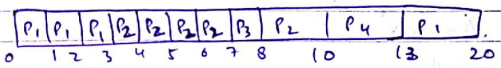
Avg TAT = 98/6 = 16.33
Avg WT = 60/6 = 10



2016
Q.

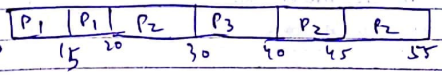
PNo	AT	BT	TAT	WT
1	0	10	20	10
2	3	6	7	1
3	7	1	1	0
4	8	3	5	2

Avg
TAT = 33/4 = 8.25
WT = 13/4 = 3.25

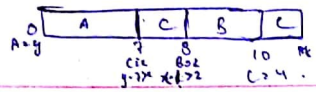


2007
Q.

PNo	AT	BT	TAT	WT
1	0	20		
2	15	25	40	15
3	30	10		
4	45	15		



Total w.t. of P2 using SRTF?



Q. Consider 3 processes A, B, C to be scheduled as per the SRTF algorithm. The process 'A' is known to be scheduled first and when 'A' has been running for '7' units of time, process 'C' has arrived. The process 'C' has run for 1 unit of time, then the process 'B' has arrived and completed running in '2' units of time. Then, what could be the min^m burst times of processes 'A' & 'C'?

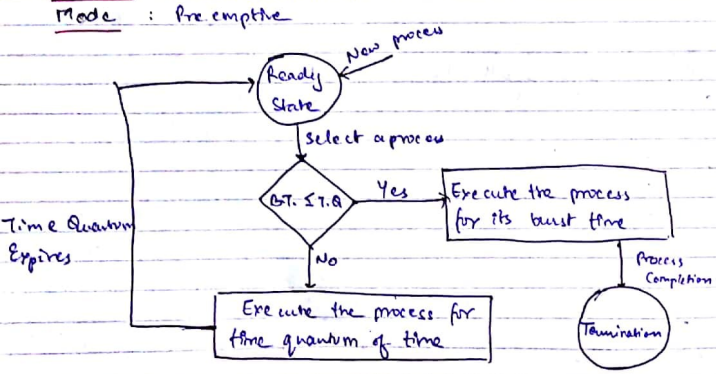
(A) 11 and 3
(B) 11 and 4
(C) 12 and 3
(D) 12 and 4

A = 12
C = 4

A → 7 + 5 = 12
C → 1 + 3 = 4
B → 2

Round Robin Scheduling:

Criteria : Time Quantum (Time slice), Arr Arrival Time
Mode : Pre emptive



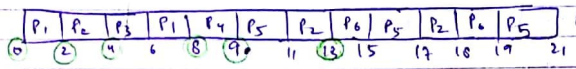
TQ = 2

Q.

P.No.	A.T	B.T.	TAT	W.T	R.T.
1	0	4	8	4	0
2	1	5	17	12	1
3	2	2	4	2	2
4	3	1	6	5	5
5	4	6	17	11	5
6	6	3	13	10	7

Avg:
TAT = 65/6 = 10.83
WT = 44/6 = 7.33
RT = 20/6 = 3.33

Ready Queue: P1, P2, P3, P1, P4, P5, P2, P6, P5, P2, P6, P5



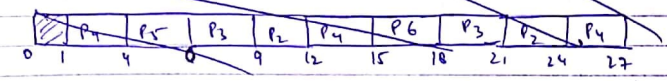
TQ = 3

Q.

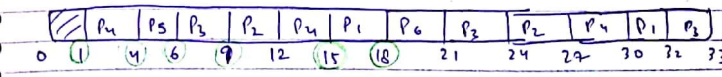
P.No.	A.T.	B.T.	TAT	WT	RT
1	5	5	27	22	10
2	4	6	23	17	5
3	3	7	30	23	3
4	1	9	29	20	0
5	2	2	4	2	2
6	6	3	15	12	12

Avg
TAT = 128/6 = 21.33
WT = 96/6 = 16
RT = 32/6 = 5.33

Ready Queue: P4, P5, P3, P2, P4, P6, P3, P2, P4, P3



Ready Queue: P4, P5, P3, P2, P4, P1, P6, P5, P2, P4, P1, P3

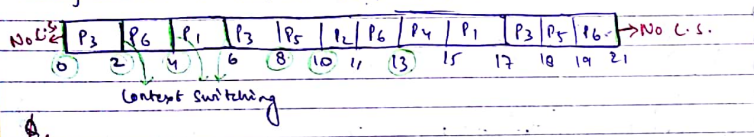


TQ = 2

P.No.	AT	BT	TAT	WT	RT
1	2	4	15	11	2
2	4	1	7	6	6
3	0	5	18	13	0
4	5	2	10	8	8
5	3	3	16	13	5
6	1	6	20	14	1

Avg.
TAT = 86/6 = 14.33
WT = 65/6 = 10.83
RT = 22/6 = 3.67

Ready Queue: P3, P6, P1, P5, P2, P6, P4, P1, P3, P5, P6



NOTE

↳ In the Round Robin, if the time quantum is less, then the no. of context switches will increase and response time will be less.

⇒ If the time quantum is large, then, the number of context switches will decrease and response time will be more.

⇒ If the time quantum is greater than all the burst time, then Round Robin behaves similar to FCFS scheduling.

WB. Pg82

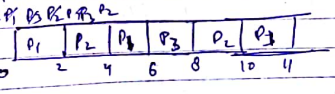
Q.31.

P.No.	AT	BT
P1	0	5
P2	1	7
P3	3	4

FCFS → P1, P2, P3

TQ = 2 → P1, P2, P1, P3, P2, P3, P2

Round Robin:



P1 P3 P2

Q. Consider a system which has 4 processes, P1, P2, P3 & P4, all arriving in the ready queue in the same order at time 0. The B.T. requirement of these processes 4, 1, 8, 1. Then what is the completion time of process P1, assuming RR scheduling with TQ = 1 ?

P	AT	BT
1	0	4
2	0	1
3	0	8
4	0	1

CT of P1 = ?

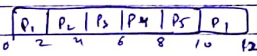
(a) 7 (c) 9
(b) 8 (d) 10

P1 P2 P3 P4 P1 P3

Q. Consider 'n' processes sharing the CPU in RR scheduling. The context switching time is 's' units, then what must be the time quantum 'q' such that, the no. of context switches are reduced, but at the same time each process is guaranteed to get its turn/job at the CPU after every 't' seconds of time?

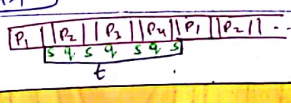
(a) $q = \frac{t - ns}{n+1}$
(c) $q = \frac{t - ns}{n-1}$
(b) $q = \frac{t + ns}{n-1}$
(d) $q = \frac{t + ns}{n+1}$

AT assume 0 if not given
BT assume ∞ if not given



$t = q(n-1) + ns$

$q = \frac{t - ns}{n-1}$



#Longest Job First:

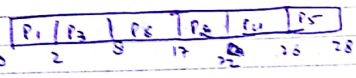
Criteria : Burst Time

Mode : Non Preemptive

Q.

PNo.	AT.	BT.	TAT	WT
1	0	2	2	0
2	1	5	21	16
3	2	6	6	0
4	3	4	23	19
5	4	2	24	22
6	5	9	12	3

Avg
TAT = 88/6 = 14.67
WT = 60/6 = 10



NOTE

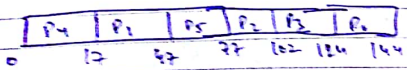
↳ If the burst time of the processes are matching then schedule the process which has the lowest arrival time.

Q.

PNo	AT	BT	TAT	WT
1	17	30	30	0
2	15	25	87	62
3	23	22	101	79
4	0	17	17	0
5	20	30	57	27
6	5	20	139	119

Avg:
TAT = 431/6 = 71.83
WT = 288/6 = 47.83

P4 P1 P5 P2 P3 P6



Longest Remaining Time First (LRTF)

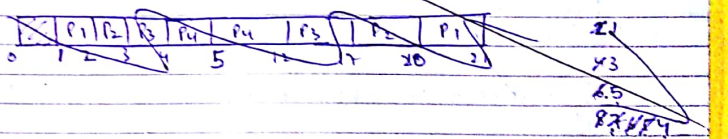
Criteria : Burst Time

Mode : Preemptive

Q.

PNo	A.T.	BT.	TAT	WT
1	1	2	20	18
2	2	4	16	14
3	3	6	14	8
4	4	8	8	0

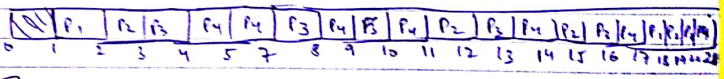
Avg
TAT = 60/4 = 15
WT = 40/4 = 10



Q.

PNo	AT	BT	TAT	WT
1	1	2	17	15
2	2	4	17	13
3	3	6	17	11
4	4	8	17	9

Avg
TAT = 68/4 = 17
WT = 48/4 = 12



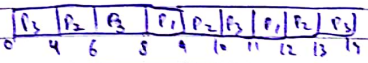
Read

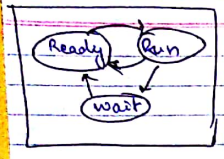
Q.

PNo	AT	BT	TAT
1	0	2	12
2	0	4	13
3	0	8	14

Avg TAT = 39/3 = 13

Avg TAT using LRTF?





Life Cycle

CPU time
I/O time
CPU Time
I/O time
⋮
Complete

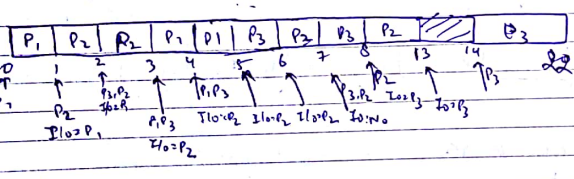
① CPU ✓
② I/O ✓
③ CPU, I/O ✓
④ CPU, I/O, I/O ✓
⑤ I/O, I/O, I/O ✓
⑥ CPU, CPU, I/O ✗

Q.

P No.	AT	Execution Time: I/O Time	CPU Time	I/O Time	C.T.
1	0	1₀	2₁₀	2₁₀	5
2	1	3₁₀	4₈ 5₀	5₀	13
3	2	3₁	6	8	22

What is completion time of processes P_1, P_2, P_3 using SRTF algo?

NOTE ① Process first spends CPU Time followed by I/O Time and followed by CPU time again.
② I/O of the processes can be overlapped as much as possible.



C.T.

P_1

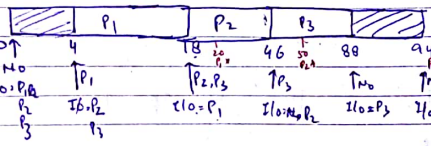
P_2

P_3

CT? Using SRTF ?

Q.

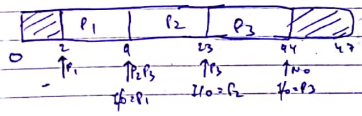
P No.	AT	Execution Time: I/O Time	CPU Time	I/O Time	CT
1	0	4₀	14₀	2₀	20
2	0	8₄	28₀	4	50
3	0	12₈	42	6	94



(a) 18, 46, 88
(b) 18, 46, 94
(c) 20, 50, 94
(d) 18, 50, 94

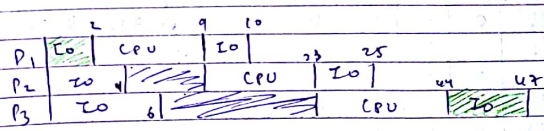
Q.28. (NCB Pg 81)

P No	AT	Execution Time: I/O	CPU	I/O
P_1	0	2₀	7₀	1
P_2	0	4₂	14	2
P_3	0	6₄	21	3



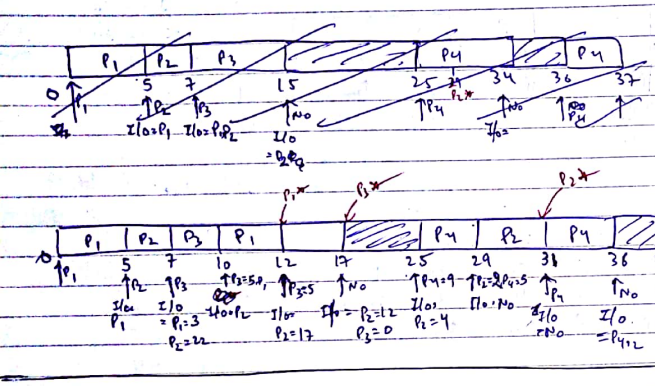
(b)

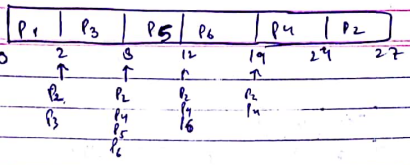
CPU idle time = $\frac{5}{42} \times 100 = 10.638\%$



Q.

P.No.	AT.	Execution Time: CPU time	I/O time	CPU time	CT?
1	0	5_0	5_3	2	12
2	3	2_0	22_{14}	2	31
3	7	8_0	0	0	17
4	25	9	2	1	39



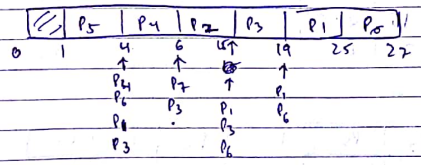


NOTE
↳ If the priorities of the processes are matching then schedule the process which has lowest arrival time.

Q.

Priority	P.No	AT	BT	TAT	WT
5	4	P_1	4	6	21	15
High ← 7		P_2	6	9	9	0
6		P_3	3	4	16	12
6		P_4	2	2	4	2
1		P_5	1	3	3	0
3		P_6	2	2	25	23

Avg
TAT = 78/6 = 13
WT = 52/6 = 8.66



Priority Based Scheduling:

Criteria: Priority
Mode: Non Preemptive

Eg.

Priority	PNo.	A.T.	B.T.	TAT	WT
2	P1	0	2	2	0
1	P2	1	3	26	23
3	P3	2	6	8	2
4	P4	3	5	21	16
High ← 6	P5	4	4	8	4
5	P6	5	7	14	7

Avg
TAT = 79/6 = 13.83
WT =

Preemptive Priority:

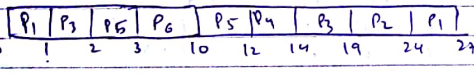
Criteria : Priority

Mode : Preemptive

Q.

P.No	AT	BT	Priority	TAT	WT
P_1	0	4	2	27	23
P_2	1	5	3	23	18
P_3	1	6	4	18	12
P_4	2	2	5	12	10
P_5	2	3	6	10	7
P_6	3	7	7 ← high	7	0

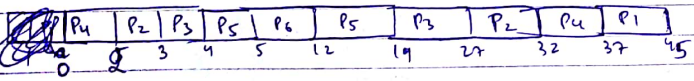
Avg
TAT = 97/6 = 16.16
WT = 70/6 = 11.67



Q.

PNo	AT	BT	Priority	TAT	WT
P_1	1	8	2	44	36
P_2	2	6	3	30	24
P_3	3	9	5	24	15
P_4	0	7	2	32	30
P_5	4	8	6	35	27
P_6	5	7	7 high	7	0

Avg
TAT = 157/6 = 26.16
WT = 109/6 = 18.66



WB Pg87

Q. 29.

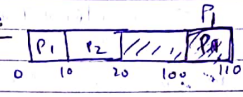
	CPU	IO
P_1	10ms	90ms
P_2	10ms	90ms

Pg87

Q. 29

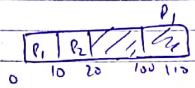
PNo	AT	CPU time	IO time
P_1	0	10ms	90ms
P_2	0	10ms	90ms

(a) FCFS:



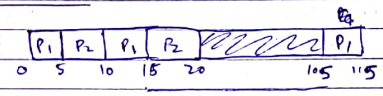
= CPU utilization = 20/110

(b) SRTF:



(c) Priority:

(d) RR TQ = 5ms:



(d)

Highest Response Ratio Next

Criteria : Response Ratio
Mode : Non Preemptive

$$\text{Response Ratio} = \frac{w+s}{s}$$

w → Waiting Time
s → Service / Burst Time

NOTE
↳ The above algorithm favours the shorter jobs and limits the waiting time of longer jobs.

Q.

P.No.	AT	B.T.	TAT	WT
P_1	0	3	3	0
P_2	2	6	10	12
P_3	4	4		
P_4	6	5		
P_5	8	2		

P_1	P_2	P_3	P_5	P_4
0	3	9	13	15	20

$P_3 = \frac{5+4}{4} = 2.25$

$P_4 = \frac{3+5}{5} = 1.6$

$P_5 = \frac{3}{5} = 0.6$

$P_4 = \frac{7+5}{5} = 2.4$

$P_5 = \frac{5+2}{2} = 3.5$

Q.

P.No.	AT	BT	TAT	WT
P_1	1	2	2	0
P_2	4	6	6	0
P_3	5	3	6	3
P_4	6	5	13	8
P_5	8	1	3	2

	P_1		P_2	P_5	P_3	P_4
0	1	3	4	10	11	14	19

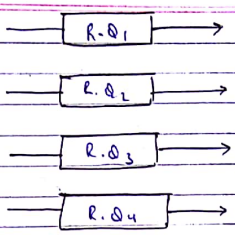
$P_2 = \frac{6+3}{3} = 2.47$

$P_4 = \frac{9}{5} = 1.8$

$P_5 = 3$

$P_3 \quad P_4$
$= \frac{6+3}{3} \quad = \frac{5+5}{5} = 2$
$= 3$

Multi Level Queue Scheduling :



→ Depending on the priority of the process in which particular ready queue, the process has to be placed will be decided.
→ The high priority processes will be placed in the top level ready queue and the low priority processes will be placed in the bottom level ready queue.
→ Only after the completion of all the processes from the top level ready queue, the further level ready queue processes will be scheduled.
→ If this is the strategy followed, then the processes which are placed in the bottom level ready queue will suffer from starvation.

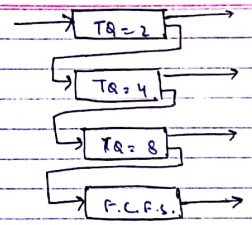
* Starvation : The indefinite waiting of a process is called as starvation.

NOTE
↳ To avoid the problem of starvation, the concept of ageing will be used.

* Ageing : If the waiting time of a process increases, then, the priority of the process will be increased.

→ If the age of a process increases by increasing the priority, the processes will definitely get a chance to execute and the starvation problem will be avoided.

Multi Level Feed Back Queue (M.L.F.B.Q)



→ In the above algorithm, every process will definitely get a chance to execute for some amount of time, but still there may be a possibility for the process to go into starvation.

Q.8. BT = 40

Queue	Remaining B.T.	TQ
1	38	2
2	31	7
3	19	12
4	2	17
5	0	22

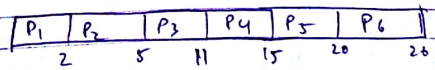
Interruptions = 4
Queue = 5

Q.	P No.	A.T.	B.T.
	1	0	2
	2	1	3
	3	2	6
	4	3	4
	5	4	5
	6	5	6

~~Throughput = 26 / 6 = 4.33~~

Throughput = $\frac{6}{26} = 0.23$

Throughput using FCFS scheduling?

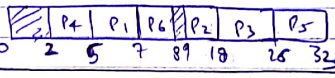


* Throughput: = # of processes / Max (C.T.) − min (A.T.)

Q.

P No.	A.T.	B.T.
1	3	2
2	9	9
3	12	8
4	2	3
5	15	7
6	3	1

Throughput = $\frac{6}{33-2} = \frac{6}{31} = 0.193$



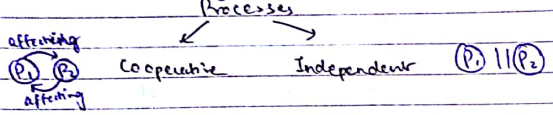
***

	Algorithm	Starvation
1.	FCFS	NO
2.	NP → SJF	YES → Here shorter processes will finish & leave system ↳ Thus, system is never full
3.	SRTF	YES
4.	RR	NO
5.	NP → LJF	YES
6.	LRTF	NO → New longer processes don't leave system (system is finite)
7.	NP → Priority	YES
8.	Pre → Priority	YES
9.	HRRN	NO → RP : $\frac{W+S}{S}$ ↑ (S constant)
10.	ML.Q.S.	YES
11.	M.L.F.B.Q	YES

* Majority of the O.S. practically implement Round Robin scheduling.

Synchronization: [Tenenbaum]

→ Processes are categorized into 2 types.



→ The execution of one process affects or affected by other process, then, these processes are said to be cooperative processes, otherwise, they are said to be independent processes.

* Understanding Synchronization:

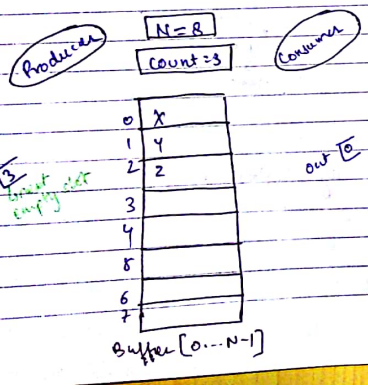
(1) The problems arises NOT having synchronization between the processes.

(2) The conditions to be followed to achieve synchronization.

(3) The solutions [Wrong and Right]

ST6P1: Problems arising due to NOT having synchronisation between the processes.

* THE PRODUCER-CONSUMER PROBLEM:




```
int count=0;
void producer(void)
{
  int temp;
  while (TRUE)
  {
    produce_item (item p);   → Check if buffer is full.
    while (count == N);      → If the condition is true, it will keep looping until condition becomes false.
      Buffer [in]=Itemp;
      in= (in+1) mod N;
      count = count +1;
  }
}
```

```
I. load Rp, m[count]
II. INCR Rp
III. store m[count], Rp
```

```
void consumer (void)
{
  int Item c;
  while (TRUE)
  {
    while (count == 0);
    Item C = Buffer[out];
    out = (out+1) mod N;
    count = count -1;
    process_item(item C);
  }
}
```

```
I. load Rc, m[count]
II. DECR Rc
III. store m[count], Rc
```

→ in is a variable used by the producer to identify the next empty slot in the buffer.
→ out is a variable used by the consumer to identify from where it has to consume the item.
→ count is a varible used by producer and consumer to identify the number of items present in the buffer at any point of time.

* Shared Resources:
(1) Count variable
(2) Buffer.

* Two conditions to be followed:
① When the buffer is full, producer is not allowed to produce the item into buffer.
② When the buffer is empty, consumer is not allowed to consume the item from the buffer.

* Universal Assumption:
→ The running process can get pre empted at any point of time after completion of the current instruction.

Analysis:
Itemp: 'A'
Item c: 'X'

count = 3 ≠ 4
in = 3 ≠ 4
out = 0 ≠ 1

0	
1	4
2	2
3	A
4	
5	
6	
7	

P → I Rp 3 4
P → II
C → I
C → II Rc 3 2
C → III
P → III

First Problem:
① Inconsistency: The processes are not properly synchronised while sharing the common variable "count", Hence, it is leading to the problem of inconsistency.

* Printer Spooler Daemon:

N = 8

Processes P1, P2, P3 ...

Printer

0	a.doc
1	b.doc
2	c.doc
3	
4	
5	
6	
7	

in 3
out 0

Spooler Directory [SD]

Respective process registers
Enter File:
I. load Ri m[in]
II. store SD[Ri], F-N
III. INCR Ri Filename
IV. store m[in], Ri

→ in is a variable used by all the processes to identify the next empty slot in the spooler directory.
→ out is a variable used by the printer to identify from where it has to print the document.

* Shared Resources:
① Spooler Directory (SD)
② 'in' variable.

* Analysis:

R.Q	P1	P2
xyz.doc		pqr.doc

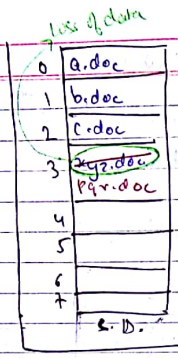
P1 → I
P1 → II
P1 → III
P2 → I
P2 → II

R1 3 4
R2 4 3

0 a.doc
1 b.doc
2 c.doc
3 xyz.doc
4 pqr.doc
5
6
7

Second Problem:

② Loss of Data: The processes are not properly synchronized while sharing the common variable "in". Hence, it is leading to the problem of loss of data.



Third Problem:

③ Deadlock: If the processes are not properly synchronized, while sharing the common variables or common resources, then, it is also risky for the occurence of deadlock.

#Definitions:

① Critical Section: The portion of program text, where shared variables or shared resources will be placed.

E.g. count = count+1, (or) count = count -1;

② Non-critical Section: The portion of program text where the independent code of the processes will be placed.

E.g. in = (in+1) mod N;

③ Race Condition: The final value of any variable depends on execution sequence of the processes. This type of condition is called as RACE condition.

E.g.

count = 4	count = 2
P→I	C→I
P→II	C→II
C→I	P→I
C→II	P→II
C→III	P→III
P→III	C→III

XXX

NOTE

↳ To avoid the RACE condition only one process should be allowed into critical section at any point of time.

25/09/2017: STEP-2

#THE CONDITIONS TO BE FOLLOWED TO ACHIEVE SYNCHRONISATION:

① Mutual Exclusion: No 2 processes may be simultaneously present inside the critical section at any point of time.

→ Only one process is allowed into critical section at any point of time.

② Progress: No process running outside the critical section should block the other interested process from entering into the critical section when critical section is free.

→ If there is only one process trying to enter into critical section, then it should be definitely allowed into critical section.

→ If two or more processes are trying to enter into critical section then one process should be definitely allowed into critical section.

③ Bounded Waiting: No process should have to wait forever to enter into critical section

→ There should be a bound on getting chance to enter into critical section.

→ Some process is indefinitely waiting to enter into critical section because critical section is always busy by some other processes. This situation should not arise.

→ If the bounded waiting is not satisfied, then, it is possible for starvation to occur.

(4) No assumptions related to hardware and processor speed:
→ Architectural neutral.

STEP-3. THE SOLUTIONS:

I. Software Type:
(a) Lock variables
(b) Strict alteration or Dekker's algorithm
(c) Peterson's Algorithm

II. Hardware Type:
(a) TSL, Instruction Set
↳ Test and set Lock

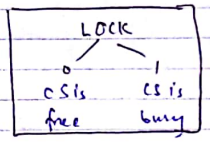
III. O.S. Type:
(a) Counting Semaphore
(b) Binary Semaphore

IV. Programming Language Compiler Support Type:
(a) Monitors

I. SOFTWARE TYPE SOLUTIONS:

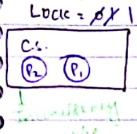
(a) Lock Variables:

Entry Section:
I. Load R1, m[lock]
II. CMP Ri, #0
III. JNZ to Step (I)
IV. Store m[lock], #1
V. C.S.
VI. Store m[lock], #0



* Analysis:

Lock = 0 / 1



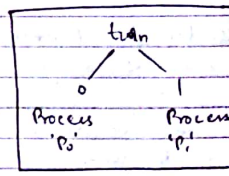
$P_1 \rightarrow I$ R_1 0
$P_1 \rightarrow II$
~~$P_1 \rightarrow III$~~
~~$P_1 \rightarrow I$~~
$P_2 \rightarrow II$ R_2 0
$P_2 \rightarrow III$
$P_2 \rightarrow IV$
$P_2 \rightarrow V$
$P_1 \rightarrow IV$
$P_1 \rightarrow V$

Mutual ~~Exte~~ ~~Excl~~ Exclusion is NOT satisfied

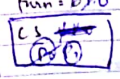
* We have proved that both the processes are entering into critical section at the same time, hence, mutual exclusion is not satisfied and solution is bound to be incorrect.

(b) Strict Alternation or Decker's Algorithm:
(process takes "turn" to enter into C.S.)

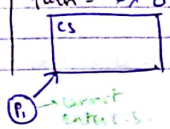
Process 'P0' code	Process 'P1' code
while (true)	while (true)
{	{
Non-CSC;	Non-CSC;
while (turn != 0);	while (turn != 1);
C.S.	C.S.
turn = 1;	turn = 0;
}	}



Analysis:
turn = 0 / 0



turn = 0 / 0



① Mutual Exclusion is satisfied
② Progress is not satisfied
→ We have proved that progress is not satisfied and solution is bound to be incorrect.

*Important Points:
① Preemption is just a temporary stop and process will come back and continue the remaining execution.
② If there is any possibility of a solution becoming wrong by taking the pre emption, then, consider the pre emption.
③ If any solution is having deadlock, then, the progress is not satisfied.

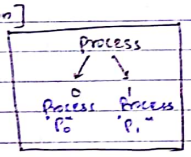
Q.Page 98-
D.15. $S_1 = 1$ $S_2 = 1$

② Mutual exclusion but not progress

CS
P_2

(c) Peterson's Algorithm: [2 process solution]

```
#define N 2
#define TRUE 1
#define FALSE 0
int turn;
int interested[n];
void Enter_Region(int process)
{ 1. int other;
  2. other = 1- process;
  3. interested[process] = TRUE;
  4. turn = process;
  5. while (turn == process && interested[other] == TRUE);
}
```



*If ... leave ... then ... to ...

C.S.

```
void Leave_Region(int process)
{
  interested[process] = FALSE;
}
```

Initially:
interested[0] = FALSE;
interested[1] = FALSE.

*turn is a shared variable used by both the processes P_0 & P_1.

Analysis 1:
Initially:
interested[0] = ~~FALSE~~ TRUE
interested[1] = ~~FALSE~~ TRUE

turn = ~~0~~ 1

Process 'Pi'	Process 'Pj'
CS	CS
P_2	P_2

# Analysis 2:
interested[0] = ~~FALSE~~ TRUE
interested[1] = ~~FALSE~~ TRUE

turn = process && interested[other] == TRUE

turn = X 0

Process 'P0'	Process 'P1'
CS	CS
P_1	P_1

① Mutual Exclusion is satisfied.
② Progress is satisfied
③ Bounded waiting is satisfied.
⇒ We have proved that all the 3 conditions are satisfied in the above algorithm. The solution is bound to be correct and it is properly providing Synchronization to the processes to access common critical section.

Q. Assume that both the processes P_0 and P_1 are trying to enter into critical section at the same time. Then, which process will enter into critical section first.

(a) The process which executes statement 2 first.
(b) The process which executes statement 3 first
(c) " " " " 4 "
(d) we can not say.

Steps:

$P_0 \rightarrow$ ~~3~~ 3 then $P_1 \rightarrow 3$

~~$P_1 \rightarrow 3$~~

$P_1 \rightarrow 4$ then $P_0 \rightarrow 4$

II. HARDWARE TYPE SOLUTIONS

TSL INSTRUCTION SET [Test and Set Lock]

⇒ TSL Register flag: copies the ~~values~~ current value of flag into register and stores the value '1' into flag in a single atomic cycle without any preemption.

Entry Section:

I. TSL Ri, m[flag]
II. CMP Ri, #0
III. JNZ to step (I)
IV. C.S.
V. Store m[flag], #0

*Analysis:

flag = ~~0~~ 1

CS

$P_1 \rightarrow$ I — R_1 0
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III
$P_1 \rightarrow$ IV
$P_2 \rightarrow$ I — R_2 1
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III

(1) Mutual Exclusion is satisfied.

(1) Mutual Exclusion is satisfied.
(2) Progress is satisfied.
(3) Bounded Waiting is NOT satisfied.

Analysis: (Bounded Waiting)

flag = 0;

CS (P1)

P1, P3, P4, P5

$P_1 \rightarrow$ I
$P_1 \rightarrow$ II — R_1 0
$P_1 \rightarrow$ III
$P_1 \rightarrow$ IV
$P_2 \rightarrow$ I
$P_2 \rightarrow$ II — R_2 1
$P_2 \rightarrow$ IV

Round Robin:

flag = ~~0~~ 1, 0 ↳ B.W. not satisfied.

CS

RQ: P1 P3 P4 P5 P1 P3 P4 P5 P1 P3

NOTE

↳ If some process is in the critical section, then all the other processes which are trying to enter critical section will be regularly checking I, II, III instructions.

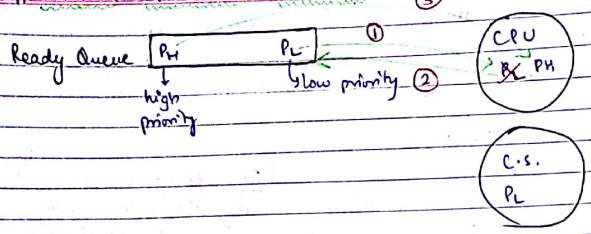
→ These processes are busy in checking these instructions and waiting to enter into critical section. This is called as busy waiting and these processes are wasting the CPU cycles [time].

→ Busy waiting is also called as spin lock.

→ To avoid busy waiting, semaphore will be used.

Solution	Mutual Exclusion	Progress	Bounded waiting
(1) Lock Variable	✗	✓	✗
(2) Strict Alteration (or) Deckee's algorithm	✓	✗	✓
(3) Peterson's Algorithm	✓	✓	✓
(4) TSL instruction set	✓	✓	✗

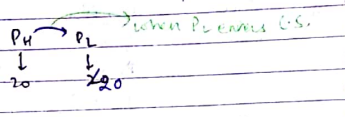
Priority Inversion Problem:



Live Lock
$P_L \rightarrow$ Ready
$P_H \rightarrow$ Run

Solution:

→ a. Priority Inheritance



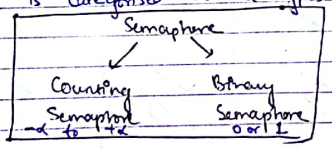
→ If a lower priority process is executing the critical section and a higher priority process arrives into the CPU.
→ the low priority process inherits the priority of the higher priority process.
→ Thus, the process which arrives first executes the critical section first, i.e., P_L.

26/09/22:

II Operating System Type Solutions:

Semaphore:

→ Semaphore is an integer variable used by various processes in a mutual exclusive manner to achieve synchronization.
→ Improper usage of semaphore will also give wrong results.
→ Semaphore is categorised into 2 types:



→ The two different operations will be performed on the semaphore variable by the processes.
① Down(); or wait(); or p();
② Up(); or signal(); or V(); or Release();

Counting Semaphore:

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

```
Up (Semaphore S)        → In any case the process is not suspended.
{
  S.valvalue = s.value + 1;
  if (s.value ≤ 0)
  {
    Select a process from
    the suspended list
    and wakeup();
  }
}
```

→ After performing down operation, if the process is getting suspended, then, it is called as unsuccessful down operation.
→ If it is unsuccessful down operation, then process will not continue further execution.
→ After performing down operation if the process is not getting suspended, then, it is called as successful down operation.
→ If it is successful down operation, then only process will continue further execution in the code.
→ The positive value of the counting semaphore represents the number of successful down operations we can perform.
→ The down operation on the counting semaphore is successful only if the initial value of the semaphore is

$S \geq 1$

→ There is no unsuccessful up operation and up operation is always successful.
→ The process performing up operation will definitely continue further execution.

Q. Consider a system where initial value of the counting semaphore is S = +17, then, various semaphore operations like 23P, 18V, 16P, 14V, 1P are performed. Then what is the final value of the counting semaphore.

+17 → 23P = ~~+6~~ − 6 | +12 → 16P = −4 | +10 → 1P → 9
~~+6 → 18V = +12~~ | −4 → 14V = 10

(a) 7 (b) 8 (c) 9 (d) 10

⇒ S = ~~+2~~, +1, 0, ~~−1~~, −2

P_1	P_2	P_3	P_4
P	P	P	P
✓	✓	X	X

Suspended list

NOTE
↳ The negative value of the counting semaphore indicates the number of processes blocked [in the suspended list]

2016
Q. Consider a counting semaphore variable 'S'. The various semaphore operations like 20P, 12V are performed during execution. Then what it the largest initial value of 'S' so that 1 process will be blocked.

~~+9 → 20P = −1~~ x + 9 + 20P + 12V = −1 ⇒ x − 20 + 12 = −1
~~x = 8~~ x = 7 ⇒ x = 7

Binary Semaphore:

```
Down (Semaphore S)
{
  If (S.value == 1)
    S.value = 0;
  else
  {
    Block the process and
    place its PCB in the
    suspended list();
  }
}
```

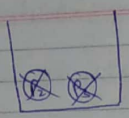
```
Up (Semaphore S)
{
  If (Suspended list() is Empty)
    S.value = 1;
  else
  {
    Select a process from the
    suspended list and wakeup();
  }
}
```

→ After performing down operation, if the process if getting suspended, then, it is called as unsuccessful down operation
→ If it is unsuccessful down operation, then, the process will not continue further execution.
→ After performing down operation, if the process is not getting suspended, then, it is called ~~as~~ as successful down operation.
→ If it is successful down operation, then only process will continue further execution.
→ Down operation on the binary semaphore will be successful, only if the initial value of the semaphore is 1. S = 1

→ There is no unsuccessful up operation and up operation is always successful.
→ The process performing up operation will definitely continue further execution.

S = X, 0, X, 1

P_1	P_2	P_3	P_4	P_5	P_6	P_7
P	P	P	V	V	V	V
✓	X	X	✓	✓	✓	✓



W.B Pg 91

Q.22: Mutex = 1 ; 9p(), 4v(), 6p(), 8v(), 3p(), 2v()

}

Mutex = X, X, 0, X, 0

NOTE

↳ * Applicable for both counting and binary semaphore.

1. ⇒ Every semaphore variable will have its own suspended list.
2. ⇒ The down and up operations are atomic.
3. ⇒ If more than one process is in the suspended list, then, every time, when we perform one up operation, one process will wake up from the suspended list and this will be based on FCFS.
4. ⇒ If two or more processes are in the suspended list and there is no other process to wake up these processes, then these processes are said to be involved in the deadlock.

Q. Each process P_i; i = 1 to 9 executes the following code:

```
while (true)
{ p(mutex);
  CS
  v(mutex);
}
```

```
while (true)
{ v(mutex);
  C.S.
  v(mutex);
}
```

→ Process P_{10} executes this code.

process can execute any no. of times

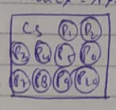
What can be the maximum no. of processes that may be present inside the critical section at any point of time?

NOTE Initial value of binary semaphore mutex = 1.

(a) 2 (b) 3 (c) 9 (d) 10 (e) 1

*Analysis:

mutex = X, 0, X, 0, X, 0, X, 0, X, 0, X, 0, X, 0, X, 0, X, 0, 1



* If P_{10} after every process it makes the value of mutex = 1.

* So, all process are able to enter C.S.

Q. P_{10};

mutex = X, 0

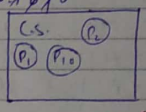
while (true)

{ v(mutex);

C.S.

p(mutex);

}



* Maximum 3 processes can enter C.S.

(a) 2 (b) 3 (c) 9 (d) 10 (e) 1

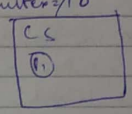
Q. P_1-P_9

```
while (true)
{ p(mutex)
  C.S.
  v(mutex)
}
```

P_{10}

```
while (true)
{ p(mutex)
  C.S.
  v(mutex)
}
```

mutex = 1, 0



- MF satisfied
- Progress satisfied
- Bounded wait satisfied

(a) 2 (b) 3 (c) 9 (d) 10 (e) 1

2003

Q. Consider two concurrent processes 'P' and 'Q' executing their respective codes:

Process 'P' code	Process 'Q' code
while (true)	while (true)
{	{
w: P(T) | P(S)	y: P(S)
print ('0');	print ('1');
print ('0');	print ('1');
x: V(S) | V(T)	z: V(T) | V(S)
}	}

what should be the binary semaphore operations on w, x, y, z respectively and what initial values of binary semaphores 'S' and 'T' should be taken in order to get print always as: 00110011 00.....

(a) w = P(T), x = V(T), y = P(S), z = V(S); S = T = 1;
(b) w = P(T), x = V(T), y = P(S), z = V(S); S = 1; T = 0;
(c) w = P(T), x = V(S), y = P(S), z = V(T); S = T = 1;
(d) w = P(T), x = V(S), y = P(S), z = V(T), S = 0, T = 1;

S = ~~1~~ 0 X 0
T = X ~~0~~ 1

Q. Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1, where n is odd?

(a) w = P(S); x = V(S); y = P(T); z = V(T); S = T = 1
(b) w = P(S); x = V(T); y = P(T); z = V(S); S = T = 1
(c) w = P(S); x = V(S); y = P(S); z = V(S); S = 1
(d) w = V(S); x = V(T); y = P(S); z = P(T); S = T = 1

S = X ~~0~~ 1 — 011110 ✓

(d)

W8Pg90

Q.17.

	P1	P2
Sx = ~~1~~ 0 X ~~0~~ 0 X 0	while true do {	while true do {
Sy = ~~1~~ 0 ~~0~~ 0	L1: ~~P(Sy)~~ | P(Sx) | P(Sx) | P(Sx) | P(Sx)	L3: P(Sx) | P(Sy) | P(Sy) | P(Sx)
	L2: P(Sy) | P(Sx) | P(Sx) | P(Sy)	L4: P(Sy) | P(Sx) P(Sy) | P(Sy)
	x = x + 1;	y = y + 1;
	y = y - 1;	x = y - 1;
	V(Sx)	V(Sy);
	V(Sy)	V(Sx);
	}	}

* Both processes should not be in suspended state at the same time

Q16 S0 = X ~~0~~ X ~~0~~ 1 X 0 X 1 — (a) 000 (b) 0
S1 = ~~0~~ 0 X ~~0~~ 1 ~~0~~ X 0
S2 = 0 X X 0 X X 0

Analysis:

Case 1: P0 → P1 → P0 → P2 → P0
S0 = X ~~0~~ X ~~0~~ X 0
S1 = 0 X ~~0~~ X 1 — 000
S2 = 0 X X 0 1

Case 2: P0 → P1 → P2 → P0
S0 = X ~~0~~ X X 0
S1 = 0 X ~~0~~ 1 — 00
S2 = 0 X 0 1

At least twice (Process wake up)

Case 3:

WB:
Chapter 3

Q.1. TQ = Q
C.S. = T
I/O = R

Efficiency = Useful time spent by CPU / Total time spent by CPU ∴

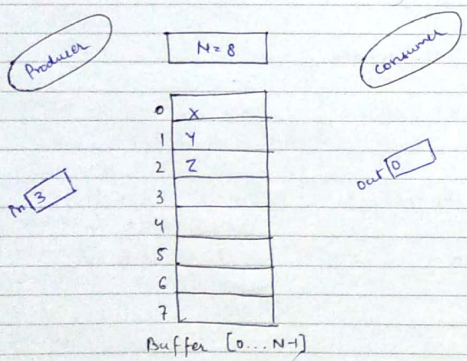
$\frac{Q}{Q+T} \in \frac{Q}{Q+T+R}$ → I/O is unpredictable so ignore it

* I/O
  - ① I/O may occur alone
  - ② I/O may occur alongwith CPU
  - ③ Multiple I/Os may occur simultaneously.

27/09/2017

Classical Problems of IPC :
↳ Inter Process Communication.

I. Producer - Consumer with Semaphore :




```
Semaphore mutex = 1;
Semaphore Empty = N;
Semaphore Full = 0;
void producer (void)
{
  int Item p;
  while (true)
  {
    produce_item (item p)
    down (Empty);
    down (mutex);
    Buffer [in] = Item p;
    in = (in+1) mod N;
    up (mutex);
    up (Full);
  }
}
```

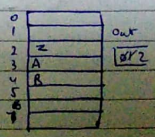
```
void consumer (void)
{
  int Item c;
  while (true)
  {
    down (Full);
    down (mutex);
    Item c = Buffer[out];
    out = (out+1) mod N;
    up (mutex);
    up (Empty);
    process_item (item c);
  }
}
```

→ Mutex is a binary semaphore used by producer and consumer to access the buffer in a mutual exclusive manner.
→ Empty is a counting semaphore variable representing the number of slots empty in the buffer at any point of time.
→ Full is a counting semaphore variable representing the number of slots full in the buffer at any point of time.

Analysis:

mutex = V V V V V V 1
Empty = 8 7 6 5 4 5
Full = 8 7 8 9 8 3
8 8 8 8

Item p = A, B
Item c = X, Y



Q. What happens if we interchange down(Empty), down(mutex) in the producer code.

(a) No problem, the solution still works correct.
(b) Both producer and consumer will access the buffer at the same time.
(c) Some of the items produced by the producer will be lost.
(d) It is possible for deadlock.

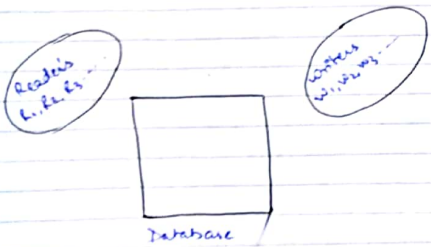
* Buffer is full:
  Mutex = 0
  Empty = 0, -1
  Full = 7

Q. What happens if we interchange up(mutex), up(full) in the producer code? [Same options as above]

(a) No problem, the solution still works correct.

* Mutex = 0 & 1
  Empty = 0
  Full = 8 7

READERS-WRITERS PROBLEM:




```
int rc = 0;
semaphore mutex = 1;
semaphore db = 1;
void Reader(void)
{
  while(true)
  {
    down(mutex);
    rc = rc+1;
    if (rc==1) down(db);
    up(mutex);
    D.B
    down(mutex);
    rc = rc-1;
    if (rc==0) up(db);
    up(mutex);
  }
}
```

```
void writer(void)
{
  while(true)
  {
    down(db);
    D.B
    up(db);
  }
}
```

4 conditions to be followed:

1. R → W ✗
2. R → R ✓
3. W → R ✗
4. W → W ✗

→ rc is the reader's count, represents the number of readers present in the database at any point of time.
→ mutex is a binary semaphore used by the readers in a mutual exclusive manner.
→ db is a binary semaphore used by readers and writers in a mutual exclusive manner.

Analysis (1):

rc = 0 / 2

mutex = 1 / 0 / 0 / 1

db = 1 / 0

DB (R1) (R2) — ① ✓ consumer ② ✓ cond.

Analysis (2):

rc = 0 / 1

mutex = 1

db = 1 / 0

D.B. (W1)

Q. What happens if we interchange down(mutex), rc = rc+1 in the readers code?

(a) No problem, the solution still works correct.

(b) multiple readers are NOT allowed into database at the same time.

(c) Both reader and writer will enter into database at the same time.

(d) It is possible for deadlock.

rc = 0 / 1

mutex = 1 / 0

db = 1 / 0

DB (R1) (R2) (W1)

rc = 0 / 1 / 2

mutex = 1 / 0

db = 1 / 0

Q. What happens if we interchange "if" condition and "up(mutex)" in the reader's code? (Same options as above)

rc = 0 / 1

mutex = 1 / 0 / 1

db = 1

(c) Both reader and writer will enter into database at the same time.

Q. What happens if we interchange "down(mutex)", "rc = rc - 1" in the reader's code?

rc = 0 / 0 / 1 / 0 / 1

mutex = 1 / 0 / 1

db = 1 / 0 / 1 / 0

(d) It is possible for deadlock

DB (R1)

rc = 0 / 1

mutex = 1 / 0 / 0

db = 1 / 0

WB Page 91

Q.19. L1: down(mutex);

if (w==1)

{

up(mutex);

goto L1

}

else

{ R = R+1;

up(mutex);

}

D.B.

down(mutex);

R = R-1;

up(mutex);

}

R = 0 / 1

w = 0 / 1

mutex = 1 / 0 / 1 / 0 / 1

L2: down(mutex);

if (W==1 or R>=1)

{

up(mutex);

goto L2;

}

w = 1;

up(mutex);

D.B.

down(mutex);

w = 0;

up(mutex);

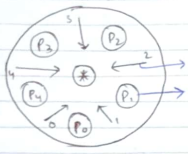
}

(d) up(mutex), up(mutex), w==1 or R>=1

Q.20. (d)

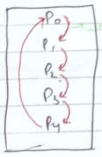
Dining Philosopher's:

N = 5




```
void philosopher (int i)        // philosopher number
{
  while (true)
  {
    thinking ();
    take_fork (i);          // take left fork
    take_fork ((i+1) % N);  // take right fork
    eat ();
    put_fork (i);           // put left fork back
    put_fork ((i+1) % N);   // put right fork back
  }
}
```

→ If all the philosophers are hungry at the same time, then everybody will take their left fork first and when they try to attempt to take the right fork, nobody will get the right fork and all the philosophers will wait on each other and that leads to deadlock.




```
#define N 5
#define THINKING 0
#define HUNGRY   1
#define EATING   2
#define LEFT  (i+N-1)%N     // left philosopher
#define RIGHT (i+1) %N      // right philosopher
semaphore mutex = 1;
semaphore S[N];             // all Si's are initialised to 0
int state[N];               // an array to keep track of everyone's state
void philosopher (int i)    // philosopher number
{
  while (true)
  { Thinking ();
    take_forks (i);
    eat ();
    put_forks (i);
  }
}
take_forks (int i)
{ down (mutex);
  state[i] = HUNGRY;
  test (i);
  up (mutex);
  down (S[i]);
}
put_forks (int i)
{ down (mutex);
  state[i] = THINKING;
  test (LEFT);
  test (RIGHT);
  up (mutex);
}
```

```
void test (int i)
{
  if (state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING)
  {
    state[i] = EATING;
    up (S[i]);
  }
}
```

→ mutex is a binary semaphore used by the philosophers in a mutual exclusive manner.
→ S[N] is an array of binary semaphore, initially all are assigned to 0.
→ State[N] is an integer array used to keep track of every philosopher's state. Initially, all the philosophers will be in the THINKING state.

Analysis:
mutex = 1

P_0 = T	S[0] = 0
P_1 = T	S[1] = 0
P_2 = ~~T~~ H E	S[2] = ~~0~~ 1 0
P_3 = T	S[3] = 0
P_4 = T	S[4] = 0

Q. Assume that P_1 and P_3 are in the eating state, then the philosopher P_2 is also trying to go into eating state, then which statement in the above code is controlling the philosopher.
(a) down(mutex);
(b) If condition
(c) test condition
(d) down(S[?]);

P_0 = T	S[0] = 0
P_1 = E	S[1] = ~~0~~ 1 0
P_2 = T	S[2] = 0
P_3 = E	S[3] = ~~0~~ 1 0
P_4 = T	S[4] = 0

mutex = 1

Q. what happens if we interchange up(mutex), down(S[i]) in the take_forks() function.
(a) No problem, solution still works correct.
(b) More than 2 philosophers can go into eating state at the same time
(c) Possible for deadlock
(d) None of the above.

mutex = ~~1~~ 0 0

P_0 = T	S[0] = 0
P_1 = ~~T~~ H E	S[1] = ~~0~~ 1 0
P_2 = T	S[2] = 0
P_3 = ~~T~~ H E	S[3] = 0
P_4 = T	S[4] = 0

Q. 28/09/2017
GATE Let p[0] ... p[4] be the processes and m[0] ... m[4] be the binary semaphore mutexes all initialized to '1'. Each process 'Pi' executes the following code:

```
wait (m[i])
wait (m[(i+1) mod 4]);
C.S.
signal (m[i]);
signal (m[(i+1) mod 4];
```

m[0] = ~~1~~ 0 ~~1~~ 0 ~~0~~ 1 0
m[1] = ~~1~~ 0 ~~1~~ 0
m[2] = 1 1
m[3] = ~~1~~ 0 1
m[4] = ~~1~~ 0 ~~1~~ 0

Consider the following statements:
I. M.E. is satisfied.
II. M.E. is NOT satisfied
III. It is possible for deadlock
Which of the following are TRUE?
(a) only I
(b) only II
(c) only I, III
(d) only II, III.

Q. Consider the following code used by the processes to access the common C.S.:

```
Shared boolean lock = FALSE;
       boolean key;
       do
       { key = TRUE;
         while (key == TRUE)
         swap(lock, key);
         C.S.
         lock = FALSE;
       }while (TRUE);
```

swap function atomically swaps two variables by using call by reference.

Consider the following statements.

I. M.E. is satisfied.

II. M.E. is NOT satisfied

III. It is possible for deadlock.

Which of the above are True?

(a) only I
(b) only II
(c) only I, III
(d) only II, III.

~~lock = FALSE T~~ ~~T F~~ lock = FALSE T
~~P_1 → key = T~~ ~~FALSE FALSE~~ P_1 → key = T F
~~P_2 → key = T~~ ~~T~~ P_2 → key = T
~~P_3 → key = T~~

lock = ~~FALSE~~ T		
P_1	P_2	P_3
key = T F	key = T	

IV. Programming Language Compiler Support Type:

Monitors:

→ Monitors is programming language compiles support type of solution to achieve synchronisation.

→ Monitors is collection of variables, ~~con~~ condition variables and procedures combined together in a special kind of module or package.

→ The processes running outside the monitor cannot directly access the internal variables of the monitor but however, they can call procedures of the monitor.

→ Monitors has an important property that only one process can be active inside the monitor at any point of time

Syntax:

Keyword → Name of monitor

```
monitor Example
{
  variables;
  condition variables;
  procedure P1
  {
    =
  }
  procedure P2
  {
    =
  }
}
```

Condition Variables:

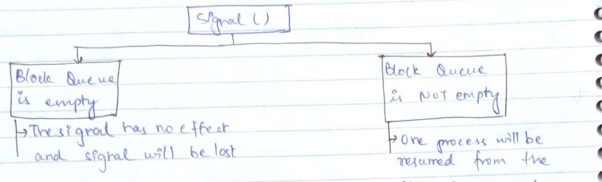
→ condition x,y;

The two different operations will be performed on the condition variables of the monitor.

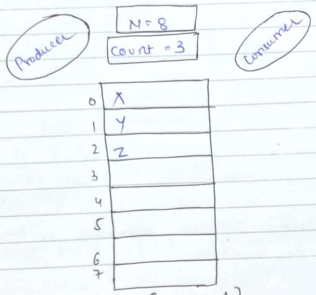
① wait();
② signal();

wait(): x.wait(); (or) wait(x);
→ The process performing wait() operation on any condition variable will be suspended and suspended process will be placed in the Block Queue of respective condition variable.

signal(): x.signal(); (or) signal(x);



Producer Consumer with monitors:




```
Monitor producer-consumer
{
  int count = 0
  condition Empty, Full;
  procedure Enter-item.
  {
    if (count == N) wait (FULL);
    enter (item);
    count = count + 1;
    if (count == 1) signal (Empty);
  }
  procedure Remove-item
  {
    if (count == 0) wait (Empty);
    Remove (item);
    count = count - 1;
    if (count == N-1) signal (full);
  }
}
procedure producer
{ int item p;
  while (TRUE)
  { produce-item (item p);
    producer-consumer. Enter-item;
  }
}
procedure consumer
{ int item C;
  while (TRUE)
  { produces-consumes. Remove-item;
    process-item (item C);
  }
}
```

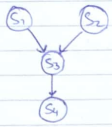
NOTE

↳ Even though we are using count variable, there will not be any inconsistency while updating the count variable because inside the monitor, only one process can be active at any point of time.

CONCURRENT PROGRAMMING:

S_1: a = b + c; Read set = {b, c, e, f, a, d, g, h};
S_2: d = e * f;
S_3: g = a / d; Write set = {a, d, g, h};
S_4: h = g x i;

Precedence Graph:



→ Any 2 statements "S_i" and "S_j" can be executed concurrently or parallelly if they are following the below conditions:

1. $R(S_i) \cap W(S_j) = \phi$
2. $W(S_i) \cap R(S_j) = \phi$
3. $W(S_i) \cap W(S_j) = \phi$

NOTE

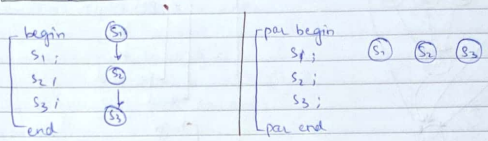
↳ The real concurrent programming is possible only on the multi-processor systems.

Concurrent: [3 meanings]:

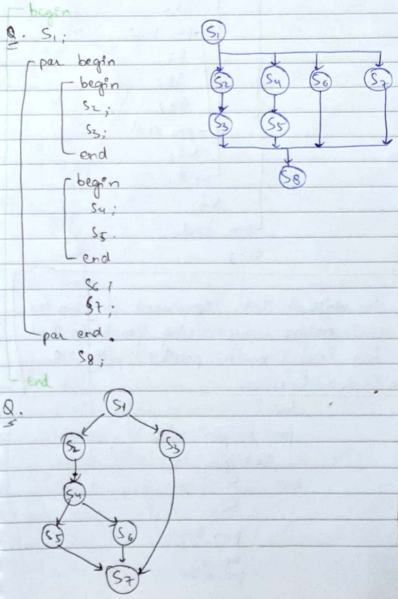
↳ They can execute concurrently or parallelly.
↳ They don't have any dependency.
↳ Anyone can start first. *

→ The concurrent programming will be written by using:

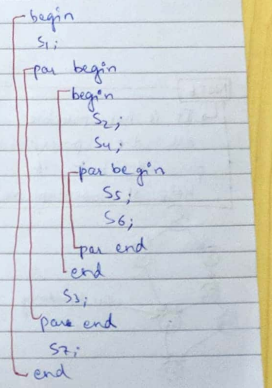
par begin - par end → parallel
(or)
co-begin - co end → concurrent



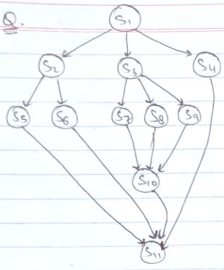
Q.



Q.



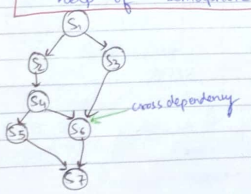
Q.



```
S1;
par begin
  begin
    S2;
    par begin
      S5;
      S6;
    par end
  end
  begin
    S3;
    par begin
      S7;
      S8;
      S9;
    par end
    S10;
  end,
  S4;
par end
S11;
```

NOTE

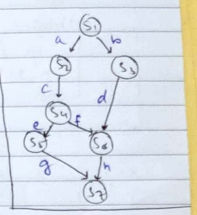
↳ It is not possible to write the concurrent program for all the precedence graphs by using par begin and par end but it is very much possible with the help of semaphore operations.



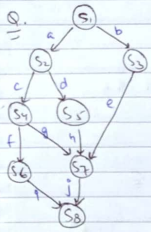

```
S1;
par begin
  S2 begin
    S2;
    S4;
  par begin
  S5; S6;
  par end
```

*All binary semaphores variables a,b,c,d,e,f,g,h are initialized to 0.

```
par begin
  begin S1, par begin V(a), V(b), par end, end;
  begin P(a), S2, V(c), end;
  begin P(b), S3, V(d), end;
  begin P(c), S4, par begin V(e), V(f), par end, end;
  begin P(e), S5, V(g), end;
  begin P(d), P(f), S6, V(h), end;
  begin P(g), P(h), S7, end;
par end
```



Q.



```
par begin
  begin S1, par begin V(a), V(b) par end, end;
  begin P(a), S2, ~~P(c)~~ par begin V(c), V(d), par end, end;
  begin P(b), S3, V(e), end;
  begin P(c), S4, par begin V(f), V(g) par end, end;
  begin P(d), S5, V(h), end;
  begin P(f), S6, V(i), end;
  begin P(g), P(h), P(e), S7, V(j), end;
  begin P(i), P(j), S8, end;
par end
```

Q. Consider the following concurrent program:

```
int x = 0 , y = 0;
par begin
  begin
    x = 1;
    y = y + x;
  end
  begin
    y = 2;
    x = x + 3;
  end
par end
```

What can be the final values of 'x' and 'y' after completion of above concurrent program?

I. x = 1 , y = 2
II. x = 1 , y = 3
III. x = 4 , y = 6

Which of the ~~following~~ above are possible?

(a) only I, II (c) only I, III
(b) only II, III (d) All I, II, III

x = 1 y = 2
y = 0 + 1 = 1 x = 0 + 3 = 3

x	y
4	6
1	3

W.B Pg87

Q.5. x = ∅ y = ∅ | x = ∅ y = ∅ | x = ∅ y = ∅

x	y
6	4
6	10
6	5

x = ∅ y = ∅ | x = ∅ y = ∅ | x = ∅ y = ∅

x	y
6	4
6	4
1	5

(b)

Pg92

Q.25: x = ∅ ✗ 2 S = ✗ ✗ ✗ + 2

W X Y Z

W X

P(1) W, X	Y, Z P(2)
I. Load R_i, M[x]	I. Load R_i, M[x]
II. INCR R_i	II. DECR R_i
III. Store M[x], R_i	III. Store M[x], R_i
V(s)	V(s)

(d)

S = 2

x = ∅ ✗ ✗ ✗ + 2

Analysis:

W → I Rw ∅ 1
W → II
W → III
Y → I
Y → II R_Y ✗ − 2
Y → III
Z → I
Z → II R_Z ✗ − 4
Z → III
W → III 4
X → I R_X ✗ 2
X → II
X → III

W X	Y Z
I. Load R_i, M[x]	I. Load R_i, M[x]
II. INCR R_i	II. DECR R_i (by 2)
III. Store M[x], R_i	III. Store M[x], R_i

Q. Consider the following concurrent program:

```
int count = 0
void tally()
{
  int i;
  for (i = 1; i <= 5; i++)
    count = count + 1;
}
main()
{ par begin
    tally() → P1
    tally() → P2
  par end
}
```

What can be the minimum final value of count after completion of both tally() functions:

NOTE count = count + 1, will execute in 3 instructions like load, INCR & store.

I. Load R_i, m[count]
II. INCR R_i
III. Store m[count], R_i

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

count = ✗ = 0

tally 1 tally 2

∅ 1

count = ~~0~~ 5 + 5

tally 1 → I
tally 1 → II
tally 2 → I
tally 2 → II
tally 2 → III
tally 1 → III

5 times

tally1 ~~0~~ + 5

tally2 ~~0~~ 5

count = 0

tally 1 | tally 2
0 | 0

P_1 P_2

1) ① ③
2)
3) ⑤
4)
5)

1) ②
2)
3)
4)
5) ④ ⑥

count = ~~0~~ X 2
count = ~~0~~ + ~~0~~ 2

P_1: 1st iteration → I
P_1: ~~1~~ 1st iteration → II
P_2: 1, 2, 3, 4 iterations
P_1: 1st iteration → III
P_2: 5th iteration → I
5th iteration → II
P_1: 2, 3, 4, 5 iterations
P_2: 5th iteration → III

R_1 ~~0~~ 1

R_2 ~~0~~ + 2

DEADLOCKS:

→ Two or more processes are waiting on some event to happen, which never happens, then those processes are said to be involved in the deadlock.

* Basics of deadlocks:

* Processes: P_1 P_2

* Resources: R_1 R_2

* Resources with instances: R_1 → single instance of resource R_1; R_1 •

R_2 •• → 2 instances of R_2

* Requesting Edge:

P_1 → Process P_1 is requesting for one instance of resource R_1

P_2 → Process P_2 is requesting for 2 instances of resources R_2

* Allocation Edge:

P_1 ← R_1 ⇒ 1 instance of resource "R_1" allocated to process "P_1"

R_2 ← P_2 ⇒ 2 instances of resource R_2 allocated to process "P_2".

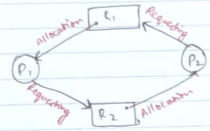
NOTE
↳ The resource request and resource allocation will be represented by resource allocation graph.

Resource Allocation Graph (R.A.G): $G(V, E)$

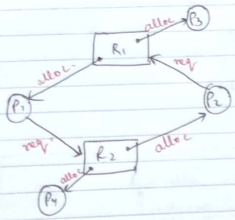
Processes, resources — Allocation, requesting

* The Resource Request / Release lifecycle:

1. The process will make a request for the resource.
2. O.S. clearly validates request of the processor.
3. If the request made by the process is valid, then the O.S. will check for the availability of resource.
4. If the resource is freely available, then, it will be allocated to the process, otherwise, process has to wait.
5. If all the resources required for the execution are allocated, then, the process will go into execution.
6. Once the execution of the process is completed, then, the process will release all the resources.



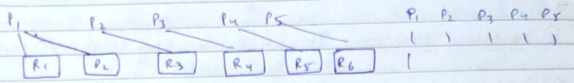
⇒ Deadlock exists in the R.A.G.



⇒ No deadlock exists in the R.A.G.

Q.1. Consider a system which has 'n' processes and sinöle tape drives and each process requires 2 tape drives, ~~then~~ to complete their execution. Then what is the maximum value of 'n' which ensures deadlock free execution.

(a) 2 (b) 3 (c) 4 ✓(d) 5



P_1 P_2 P_3 P_4 P_5
1 1 1 1 1
1

Q.2. Consider a system, which has 3 processes and each process requires 2 resources to complete their execution. Then what is the minimum no. of resources required to ensure deadlock free environment.

(a) 2 (b) 3 ✓(c) 4 (d) 5

Q.3. Consider a system, which has 'n' processes and 6 resources. Each process requires 3 resources to complete their execution. Then, what is the max^m value of 'n', which ensures deadlock free execution

(a) 2 (b) 3 (c) 4 (d) 5

P_1 P_2

Q.4. Consider a system which has 3 processes P_1, P_2 & P_3. The peak demand of each process is 5, 9, 13 respectively for the resource R. Then what is the minimum number of resources required to ensure deadlock free execution.

(a) 22 (b) 23 (c) 24 ✓(d) 25

P_1 P_2 P_3 ⇒ 4 + 8 + 12 + 1 = 25

WB. Pg. 98

Q.23. P_1 P_2 P_3 P_4 — 20 resources.

4 3 7

3 2 6 (9) (9)

→ In deadlocks, we will use 2 conditions.

① necessary condition: The deadlock may be possible or may not be possible.

② Sufficient condition: The deadlock never be possible.

→ The least upper bound which satisfies the condition is called as sufficient condition.

WB Pg 97

Q.13. m resources, n processes

demand of $P_i \rightarrow S_i$

$P_1 \cdots P_n$

$S_i \quad S_i$

$S=5 \quad P=4$ ~~$m=20$~~

$m=17$ (c)

(a) $\forall i, S_i < m$

$m=10$

$P_1 = 9$

$P_2 = 9$

$P_{100} = 9$

(b) $\forall i, S_i < n$ ✗

(c) $\sum_{i=1}^{n} S_i - n + 1 = m$

$\sum_{i=1}^{n} = m+n-1$

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

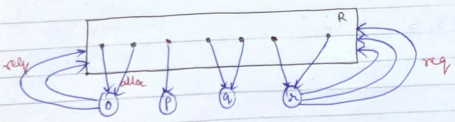
Pg 98.

Q.10. n processes

$P_i \rightarrow r_i$ of R $1 \le i \le n$

$P \rightarrow r_i^{\circ}$

y_i°



$x_0 = 2 \quad y_0 = 2$

$x_p = 1 \quad y_p = 0$

$x_q = 2 \quad y_q = 0$

$x_r = 2 \quad y_r = 3$

(b)

CONCEPTS OF DEADLOCKS:

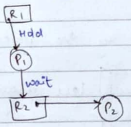
1. Deadlock characteristics
2. Deadlock Prevention
3. Deadlock Avoidance *
4. Deadlock Detection
5. Deadlock Recovery

* Deadlock Characteristics:

① Mutual Exclusion: The resource has to be allocated to only one process or it is freely available.

→ There should be a one to one relationship between the resource and the process.

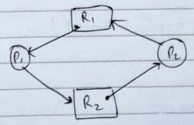
② Hold and wait: The process is holding the resource and waiting on some other resource simultaneously.

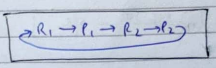


③ No Preemption: The resource has to be "voluntarily" released by the process after completion of the execution.

→ It is not allowed to forcefully preempt the resource from the process.

④ Circular Wait: The processes are circularly waiting on each other for the resources.



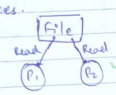


By dissatisfying any of the 4 conditions, we can prevent deadlock.

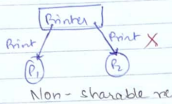
NOTE
↳ If all the 4 conditions are existing simultaneously in the system, then, definitely, there exists a deadlock.

* Deadlock Prevention:

① Mutual Exclusion: It is not possible to dissatisfy mutual exclusion always because of shareable and non-shareable resources.



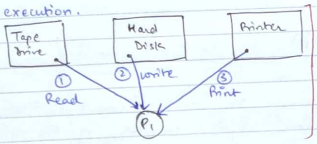
Shareable resource



Non-shareable resource

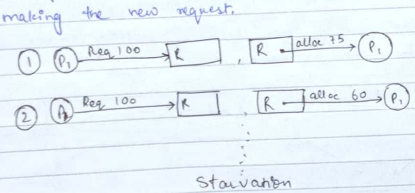
② Hold & Wait:

(i) Allocate all the resources required by the process before start of execution.



less resource utilization (or) low device utilization

(ii) The process should release all the existing resources before making the new request.



Starvation

③ No Preemption:

The process P_1 is requesting for resource R_1.

If the Resource R_1 is free, then, it will be allocated to process P_1.

The resource R_1 is not free, and it is allocated to some other process P_2.

If the process P_2 is in execution, then the process P_1 has to wait

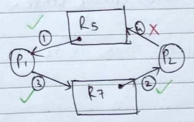
If the process P_2 NOT in the execution & it is waiting on some other resource R_2.
↳ Pre empt Resource R_1 from process P_2 and allocate it to process P_1.

④ Circular Wait:

→ The resources will be assigned with the unique numerical numbers.

→ The process can request for the resource only in the ~~inor~~ increasing (or decreasing) order of enumeration (numbering)

increasing order) $P_1 \rightarrow R_7$ ✓ | $P_1 \rightarrow R_5$ X | $P_1 \rightarrow R_9$ ✓



Total available
A B C
4 2 3
10 5 7

Resources = A, B, C
Process = P_0, P_1, P_2, P_3, P_4

Deadlock Avoidance:

→ Deadlock avoidance is implemented by using Banker's algorithm.

Safe seq ⇒ P_1, P_3, P_0, P_2, P_4

	Max Need	Current allocation	Current available	Remaining need
	A B C	A B C	A B C	A B C
P_0 →	7 5 3	0 1 0	3 3 2	7 4 3 → P_0
P_1 →	3 2 2	2 0 0	5 3 2	1 2 2 → P_1
P_2 →	9 0 2	3 0 2	7 4 3	6 0 0 → P_2
P_3 →	2 2 2	2 1 1	7 5 3	0 1 1 → P_3
P_4 →	4 3 3	0 0 2	10 5 5	4 3 1 → P_4
		7 2 5	10 5 7	

Remaining = Max Need − Current allocation Need

Current available = Total available − Total current allocation

* Safe sequence: The order in which we satisfy the remaining need of all the processes.

→ If we can satisfy remaining need of all the processes with the current available resources, then, the system is said to be in the safe state, otherwise system is said to be in the unsafe state.

→ If the system is in the unsafe state, then, it is possible for the deadlock.

→ Safe sequence may not be unique, there may be multiple safe sequences.

→ The deadlock avoidance is less restrictive than deadlock prevention.

→ The unsafe state purely depends on the behaviours of the processes.

P_1 ⇒ 332 + 200 = 532

P_2 ⇒ 532 + 211 = 743

P_0 ⇒ 743 + 010 = 753

→ Each & every time when the process is requesting for any resource, the bankers algorithm will be implemented to identify whether the system is in the safe state or unsafe state.

→ If the system is in the safe state, then, the request of the process will be granted and if the system is in the unsafe state then, the request of the process will be denied and the deadlock will be avoided.

Q.1 Q.2.

	Max	Allocated	Available	Remaining	
	A B C	A B C	A B C	A B C	
P_0	6 5 4	0 3 4	4 3 1	6 2 0	P_0
P_1	3 4 2	2 1 2		1 3 0	P_1
P_2	1 0 4	0 0 2		1 0 2	P_2
P_3	3 2 5	1 2 1		2 0 4	P_3

P_1 ⇒ 431 + 212 = 643

P_2 ⇒ 643

P_1 643 P_3

P_2 ⇒ 643 + 002 = 645

P_0 ⇒ 643 + 034 = 677

P_2 679 P_3 798

P_2 ⇒ 643 + 002 = 645

P_0 ⇒ 645 + 034 = 679

P_3 ⇒ 677 11

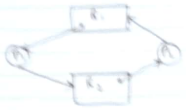
P_1 & P_3 ⇒ 643 + 121 = 764

P_0 ⇒ 764 + 034 = 798

P_2 ⇒ 798 + 002 = 800

* Deadlock Detection:

① The resources are of single instance type:



Run cycle detection algorithm.

(P1 → R1 → P2 → R2 → P1)

→ If all the resources are of single instance type, then cycle in the R.A.G. is necessary and sufficient condition for occuring of deadlock.

→ If all the resources are of NOT single instance type, then cycle in the R.A.G. is just a necessary condition but not sufficient condition for the occurrence of deadlock.

② Resources are of multiple instance type:

	Current allocation A B C	Current available A B C	Remaining Need A B C	
P_0	0 1 0	0 0 0	0 0 0	P_0
P_1	2 0 0	0 1 0	2 0 2	P_1
P_2	3 0 3	3 1 3	0 0 0	P_2
P_3	2 1 1	5 1 3	1 0 0	P_3
P_4	0 0 2	7 2 4	0 0 2	P_4
	7 2 6			

→ what happens if the process P_2 is requesting for additional resources of (0,0,1)

$P_2 \rightarrow (0,0,1)$
000 (old)
+ 001 (new)
001 (latest)

	Current alloc.	Current avail.	Remaining need.	
P_0	0 1 0	0 0 0	0 0 0	
P_1	2 0 0	0 1 0	2 0 2	
P_2	3 0 3		0 0 1	
P_3	2 1 1		1 0 0	
P_4	0 0 2		0 0 2	

$P_0 \rightarrow$ unsafe → Deadlock P_1, P_2, P_3, P_4

* Deadlock Recovery:

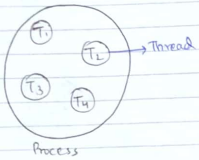
① Killing the process:
(a) Kill all the processes which are involved in the deadlock.
(b) Kill one after the other.
→ low priority processes.
→ % of process completion. (P_1 40% ✓, P_2 10%)
→ Number of resources the process is holding. (P_1 1 unit, P_2 2 units ✓)
→ Number of resources the process is requesting. (P_1 1R, P_2 2 units ✓)

② Resource Preemption: The resources will be preempted from the processes which are involved in the deadlock and the preempted resources will be allocated to some other processes, so that, there may be a possibility of recovering the system from deadlock.
→ If the above condition is followed, then, there may be a possibility for the process to go into starvation.

③ Ostrich Algorithm: Ignore the deadlock.

THREADS:

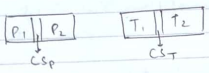
→ The light weight process.
- → No. of Instructions will be less
- → Context will be less



Advantages of Threads:

① Responsiveness: If the process is divided into multiple threads, then if one thread has completed the execution, then, the output will be responded immediately.
- This response will be faster compared to the response of the process.

② Faster Context Switches:

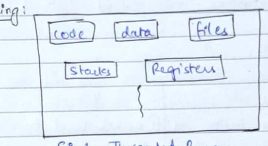


$CS_T << CS_P$

→ The context switching time, b/w the threads will be very less compared to the context switching time between the processes because threads will have less context compared to processes.

③ Effective utilization of multiprocessor system:
→ If the process is divided into multiple threads, then, different threads can be scheduled onto different processors, so, that process execution will be faster.

④ Resource Sharing:



Single Threaded Process

Multi threaded process.

→ The resources like code, data [global variables] and heap], files and memory will be shared among all the threads within the process but the stacks and registers can not be shared and every thread will have its own stacks and registers.

⑤ Enhanced throughput of the system: If the process is divided into multiple threads and if we consider one thread as one job, then, number of jobs completed per unit time will increase and throughput of the system will be enhanced.

⑥ Economical: Implementation of threads does not require any cost and there are various programming language APIs which support implementation of threads.
E.g. JAVA API.

→ The threads are further categorized into 2 types:
① User level threads
② Kernel level threads.

User Level Threads	Kernel Level Threads
① User level threads are implemented by user/programmer.	① Kernel level threads are implemented by the O.S.
② O.S. doesn't know about user level threads and it views user level thread as a process only. If means O.S. cannot recognize user level threads.	② Kernel level threads are recognized by operating system.
③ If one user level thread is performing (I/O) blocking system call, then, the entire process will go into block state.	③ If one kernel level thread is performing blocking system call, another thread will continue the execution.
④ User level threads are designed as dependent threads.	④ Kernel level threads are designed as independent threads.
⑤ Implementation of user level threads is easy.	⑤ Implementation of kernel level threads is complicated.
⑥ User level threads will have less context.	⑥ Kernel level threads will have more context.
⑦ No hardware support required for the user level threads.	⑦ Scheduling requires hardware support.

NOTE
↳ GANG scheduling is used in threads.

I/O Operation: I/O of the processes is categorised into 2 types:
① Synchronous I/O
② Asynchronous I/O

* Synchronous I/O: In synchronous I/O, the process performing I/O operation, will be placed in the block state till the I/O operation is completed.
→ Once I/O operation is completed ~~and~~, an Interrupt Service Routine (ISR) will be initiated, which brings the process from block state to ready state.

* Asynchronous I/O: In asynchronous I/O, while initiating I/O request the handler function will be registered.
→ The process is not placed in the block state and it continues to execute remaining code after initiating I/O request.
→ Once, the I/O operation is completed, the signal mechanism will be used to notify the process that the data is available and registered handler function will be asynchronously invoked.

Main mem
Primary mem
Physical mem
RAM

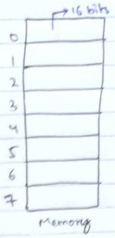
MEMORY MANAGEMENT: [William Stallings]

* functionality: Allocating and de allocating the memory to the processes.

* Goal: The efficient utilization of memory by minimizing the internal and external fragmentation.

→ Memory will be represented in 2 different ways:
① Byte addressable E.g. 512 KB
② Word addressable E.g. 512 KW

$2^2 = 4$	$2^8 = 256$
$2^3 = 8$	$2^9 = 512$
$2^4 = 16$	$2^{10} = 1024 \approx 1K$
$2^5 = 32$	$2^{20} = 1M$
$2^6 = 64$	$2^{30} = 1G$
$2^7 = 128$	$2^{40} = 1T$



Capacity of memory = No. of words * word size
= 8 words * 16 bits
= 8 * 2 B
= 16 Bytes.

Q. Consider a system which has 512 M words and each words is having size of 128 bits, then what is the capacity of the memory in Bytes:
$2^{29} * 2^4 \Rightarrow 2^{33} \Rightarrow 8$ GB

Q Consider a system which has 256 G words and each word is having 64 Bytes. Then what is the capacity of the memory in Bytes?
$2^{38} * 2^6 = 2^4 * 2^{40} \Rightarrow 16$ TB

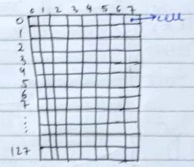
Q Consider a system where 32 binary bits are used to represent all the words of the mem and each word is having size of 32 bits, then what is the capacity of the mem in Bytes?
$2^{32} * 2^2 \Rightarrow 2^{34} \Rightarrow 16$ GB

RAM Chip Implementation:

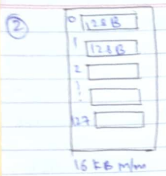
RAM Chip Size = 128 B
Organize the main memory capacity of 16 KB.

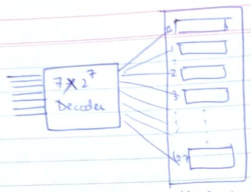
Q1. # of RAM chips required?
Q2. Draw the memory organisation map?
Q3. what is the size of the decoder required?

RAM chip size = 128 B



① No. of RAM chips required = $\frac{16\text{ KB}}{128\text{ B}} = \frac{2^{14}\ B}{2^7\ B} = 2^7 = 128$

② 

③ 

Q. RAM chip size = 256 X 1 bit
Organise main memory capacity of 32 MB.

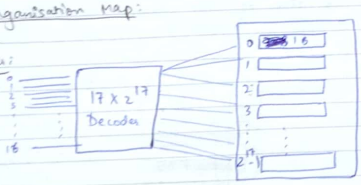
① No. of RAM chips required

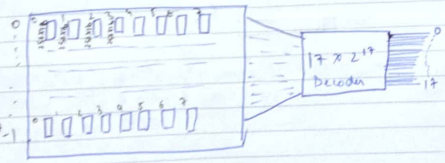
$= \frac{2^{25} \times 8}{2^8 \times 1} = 2^{17} \times 8$

$= 2^{20}$

② Memory organisation map:

③ Size of decoder:





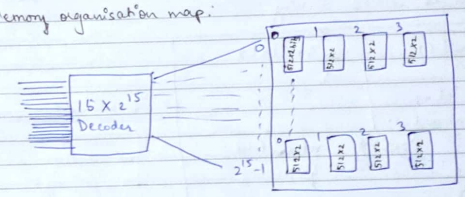
Q. RAM chip size = 512 X 2 bits
Organise main m/m capacity of 16 MB.

① No. of RAM chips:

$= \frac{2^4 \times 2^{20} \times 8}{2^9 \times 2} = 2^{17}$ chips

② Memory organisation map:

③



Q. The main memory unit with the capacity of 4MB is built using 1M X 1 bit DRAM chips. Each DRAM chip has 1K rows of cells with 1K cells in each row.
The time taken for single refresh operation is 100ns.
The time required to perform one refresh operation on all the cells in the memory unit is?

(a) 100ns
(b) $100 * 2^{10}$ ns
(c) $100 * 2^{20}$ ns
(d) 3200×2^{20} ns

Ram chip size = $1M * 1$ bit =
Memory = 4MB = $2^2 \times 2^{20} * 8$

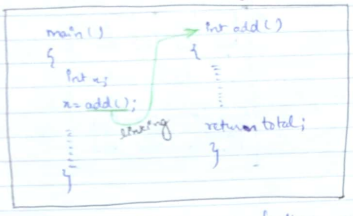
Ram chips = $\frac{2^{20} * 2^2 * 8}{2^{20}} = 2^5 = 32$ RAM chips.

$32 * 2^{20}$ cells

Total refresh time = $(32 * 2^{20} * 100)$ ns
$= 3200 * 2^{20}$ ns

Loading, Linking and Address Binding:

→ Bringing the program from secondary memory to main memory is called as loading.
→ Establishing the linking between all the modules or all the functions of the program in order to continue program execution is called linking.



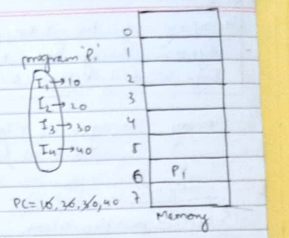
→ Loading and linking are further categorized into 2 types:
① Static ② Dynamic

* Static: Loading the entire program into memory before start of program execution is known as static loading.
→ Inefficient utilization of memory because whether it is required or not required, the entire program will be brought into memory.
→ Program execution will be faster.
→ If the static loading is used, accordingly, static linking will be applied.

* Dynamic: Loading the program into memory on demand is called as dynamic loading.
→ Efficient utilization of memory.
→ Program execution will be slower.
→ If dynamic loading is used, accordingly, dynamic linking will be applied.

Address Binding:

→ Association of program instructions and data, to the actual physical memory locations is called as address binary.
→ Address binding is categorized into 3 types:
① Compile Time A.B.
② Load Time A.B.
③ Execution Time/Dynamic A.B.



* Compile Time Address Binding:
→ If the compiler is responsible of performing address binding then, it is called as compile time A.B.
→ This type of address binding will be done, before loading the program into memory.
→ Compiler requires to interact with the O.S. memory manager to perform compile time A.B.

* Load Time Address Binding:
→ This type of address binding will be done after loading the program into memory.
→ This type of address binding will be done by O.S. memory manager, i.e., Loader.

* Execution Time / Dynamic Address Binding:
→ The address binding will be postponed even after loading the program into memory.
→ The program will keep on changing the locations in the memory till the time of program execution.
→ This type of address binding will be done by the processor at the time of program execution.

NOTE

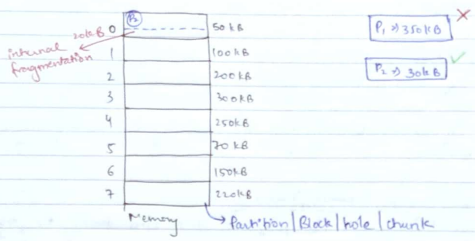
↳ Majority of the O.S. practically implement dynamic loading, dynamic linking and dynamic address binding.
E.g. Window, UNIX, Linux

Memory Management Techniques :

- Contiguous
  - → Fixed partition scheme
  - → Variable partition scheme
- Non-Contiguous
  - → Paging
  - → Multilevel paging
  - → Inverted paging
  - → Segmentation
  - → Segmented Paging

CONTIGUOUS :

* Fixed Partition Scheme :



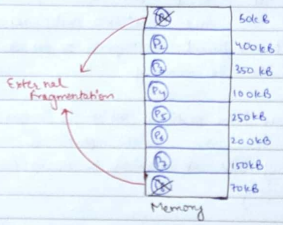
→ Partition | Block | hole | chunk

→ In the fixed partition scheme, memory will be divided into fixed number of partitions.
→ Fixed means, the number of partitions are fixed in the memory.
→ In every partition, only one process will be accomodated.
→ Degree of multiprogramming is restricted by no. of partitions in the memory.

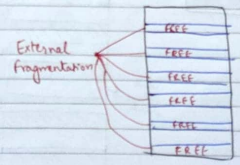
→ Maximum size of the process is restricted by maximum size of the partition.
→ Every partition is associated with the limit registers.

Limit Registers
- → Lower Limit : contains starting address of the partition.
- → Upper Limit : contains ending address of the partition.

* Variable Partition Scheme :



→ In the variable partition scheme, initially memory will be single continuous, free block.
→ Whenever a ~~proc~~ request by any process arrives, accordingly, partition will be made in the memory.
→ If the smaller process requests keep on coming, then, the larger partitions will be made into smaller partitions.



→ To avoid the problem of external fragmentation, the following techniques are used:

① Compaction: Moving all the processes towards the top or towards the bottom to make free available memory in a single contiguous place.

→ Compaction is undesirable to implement because it interrupts all the running processes in the memory.

② Implementing non-contiguous memory management techniques:

Partition Allocation Methods:

① First Fit: Allocate the process in a partition which is first sufficient partition from top of the memory.

② Best Fit: Allocate the process in a partition which is smallest sufficient among the free available partitions.

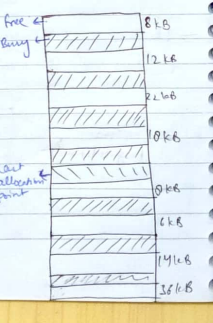
- To find out the smallest sufficient, it requires to search all the free partitions in the memory.

③ Worst Fit: Allocate the process in a partition which is largest sufficient among the free available partitions.

- To find out the largest sufficient, it requires to search all the free partitions in the memory.

④ Next Fit: Next fit also works like first fit but it will search for first sufficient partition from last allocation point.

Q. Consider the following memory map.



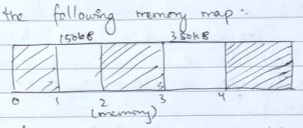
First Fit → 22 kB
Best Fit → 18 kB
Worst Fit → 26 kB
Next Fit → 36 kB

(A) 16 kB

Q. How many successive requests of 6 kB will be satisfied by using First Fit, assuming variable partition scheme.

(a) 17 (b) 18 (c) 19 (d) 20 (e) 21

Q. Consider the following memory map:



- The requests from the processes are 300 kB, 25 kB, 125 kB, 50 kB respectively.
- The above requests could be satisfied with? (Assume variable partition scheme is used).

(a) Best fit but NOT first fit
(b) First fit but NOT best fit
(c) Both first and best fit
(d) Neither first fit NOR best fit

Non-Contiguous Memory Management Technique:

* Paging:

(a) Logical address space (L.A.S.) or Virtual Address space (V.A.S.)
↳ Represented in the form of words or Bytes. E.g. 512 kW or 512 kB

(b) Logical address (L.A.) or Virtual Address (V.A.)
↳ Represented in the form of bits. E.g. 19 bits, 32 bits

(c) Physical Address Space (P.A.S.)
↳ Represented in the form of words / Bytes. E.g. 32 kW or 32 kB

(d) Physical Address (P.A.)
↳ Represented in the form of bits. E.g. 15 bits

E.g. ① L.A. = 28 bits
L.A.S. = 2^{28} W
= $2^8 * 2^{20}$ W
= 256 MW

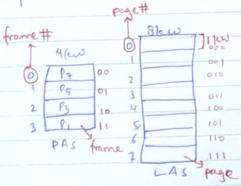
② L.A.S. = 32 kW
L.A. = 15 bits

③ P.A. = 46 bits
P.A.S. = $2^6 * 2^{40}$ Bytes
= 64 TB

④ PAS. = 12 GB
PA. = 37 bits

→ The technique of mapping CPU generated logical address to physical address is called as paging.
→ Paging is implemented in the hardware level.
→ If the logical address & the same as physical address, then no hardware support of paging is required.

① L.A. = 13 bits ; No. of frames = 4
L.A.S. = 8 kW
PA = 12 bits
PAS. = 4 kW
Page size = 1 kW
No. of pages = 8/1 = 8 pages



→ Logical address space will be divided into equal size pages.

No. of pages = L.A.S / Page size

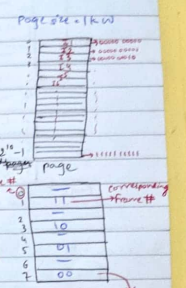
→ Physical address space will be divided into equal size frames.
→ Page size will always be same as frame size. Page size = Frame size

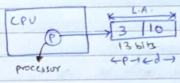
No. of frames = P.A.S. / Frame size

→ Some of the pages of L.A.S. will be brought into P.A.S.
→ Whenever the paging is applied, the page table has to be maintained.
→ No. of entries in the page table is the same as number of pages in the L.A.S.

No. of P.T.E.s = No. of pages in L.A.S.

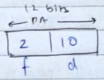
→ Page table is also called as Address Translation Table.
→ Page table entry definitely contains frame number.
→ Frame number is also called as Translation bit.





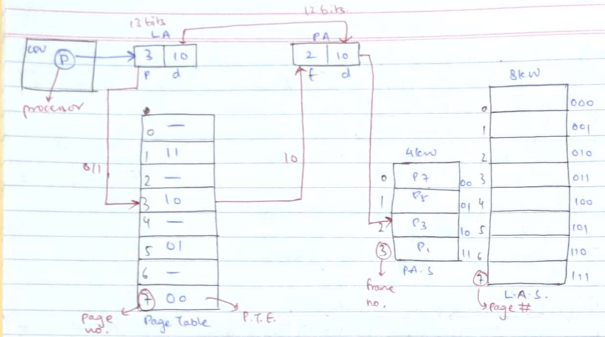
p = No. of bits required to represent pages of L.A.S. or page number.
d = No. of bits required to represent page size or ~~word no. of the page~~ or page offset.

Physical address:



f = No. of bits required to represent frames of P.A.S. or frame number.
d = No. of bits required to represent frame size or ~~from~~ word no. of the frame.
or frame offset

Windows Page Size = 4KB



e.g. LA: 15 bits.

011 | 00000 000101

P d

PA: 12 bits.

10 | 00000 00101

f d

Important Points:

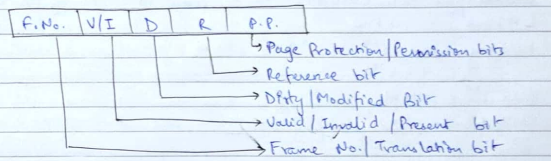
1. Whenever the process is created, paging will be applied on the process and page table will be created and base address of the page table will be stored in the PCB.
2. Paging is with respect to every process and every process will have its own page table.
3. Page tables of the processes will be stored in the main memory.
4. There is no external fragmentation in paging.
5. Internal fragmentation exists in the last page and internal fragmentation is considered as $\frac{P}{2}$, where 'P' is the page size.
6. Maintaining the page table is considered as overhead for the system.

NOTE:

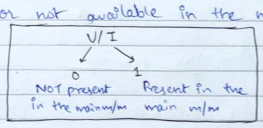
↳ Page table entry definitely contains frame number but sometimes alongwith a frame number, it may also contain additional bits like:

1. Valid / Invalid bit or present bit
2. Dirty bit / Modified bit
3. Reference bit
4. Page protection bits or permission bits.

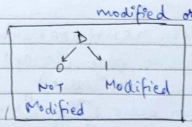
* P.T.E.



Valid / Invalid or Present bit: This bit is used to identify, whether the page is currently available or not available in the main memory.



Dirty bit or Modified bit: used to identify whether the page is modified or not modified by the processor.



No. of reference bits is just fixed (depends on availability)

Reference bits: These bits are used to identify how many times, page is refered by the processor as part of the execution.

Page Protection/Permission bits: These bits are used for the protection and security from unauthorized access

Q) Consider a system which has logical address = 27 bits and P.A = 21 bits and page size is 4 KW. Memory is word addressable. Then, calculate no. of pages and no. of frames.

LAS = 2^{27} = 128 MW

No. of pages = $\frac{128 \times 2^{20}}{4 \times 2^{10}} = 2^5 \times 2^{10} = 32$ K

PAS = 2^{21} = 2 MW

No. of frames = $\frac{2 \times 2^{20}}{4 \times 2^{10}} \Rightarrow 2^9 \Rightarrow 512$ frames.

② Consider a system which has no. of pages as 8K and page size = 16 KW and mem. is word addressable. P.A = 22 bits. Then calculate logical address and no. of frames.

PAS = 2^{35} = 4 MW

No. of frames = $\frac{8 \times 2^{10}}{2^5 \times 2^{10}} = 2^8 = 256$ frames.

L.A.S = $8 \times 2^{10} \times 16 \times 2^{10} = 2^{27}$ = 128 M W

L.A = 27 bits.

③ Consider a system which has LAS = 128 MW, PA = 24 bits, PAS is divided into 8 K frames, then, what is the page size and how many pages in the L.A.S.? [Memory is word addressable]

L.A = 27 bits. | P.A = 2^{24} W | frame size = $\frac{2^{24}}{2^{13}} = 2^{11}$ words = 2K W

No. of pages = 2^{16} = 64K pages

④ Consider a system, L.A = 32 bits; PAS = 64 MB, Memory is byte addressable and page size = 4 KB. Then, what is approximate size of page table in Bytes.

LAS = 2^{32} = 4 GB

No. of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$ pages

No. of frames = $\frac{2^{26}}{2^{12}} = 2^{14}$ frames

# of PTE = 2^{20} Size of each PTE = 14 bits ≈ 2 bytes

size of page table = 2×2^{20}

≈ 2 MB.

(a) 1 MB (b) 2 MB (c) 4 MB (d) 8 MB

Page Table size = No. of P.T.E. * P.T.E. size

⑤ Consider a system having a page table with 4K entries and L.A = 29 bits, then what is the physical address of the system has 512 frames.

No. of pages = 2^{17} | p | d | 12 | 17 |

LAS = 2^{29}

Page size = $\frac{2^{29}}{2^{12}} = 2^{17}$ W

P.A.S = $2^9 \times 2^{17}$ = 2^{26} W

f	d
9	17

PA = 26 bits.

W.B. pg 103

Q) LAS = 2^{16} B

1 word u/t bit

1 R bit

1 dirty

3 Protection

Page size = 2^9 B

No. of pages = 2^7 = 128

No. of frames = 128 = 7 bits

PTE size = 2 B = 16 bits.

Total bits = 6 + 7 = 13

Bits left for storing aging information = 3 bit = 16 - 13

2 marks
GATE 2016

Q. Consider a system which has logical address of 40 bits and page size = 16 kB and mem. is byte addressable. PTE size = 48 bits. Then what is page table size ——— in MB?

LAS = 2^{40} B

No. of pages = $\frac{2^{40}}{2^{14}} = 2^{26}$ pages

Page table size = $6 \times 2^{26} = 384 \times 2^{20} = 384$ MB

GATE 2015 1 marks

Q. Consider a system which has page size = 8kB and PA = 32 bits and memory is byte addressable and page table size = 24 MB. P.T.E. contains a valid bit, a dirty bit, 3 permission bits and translation bits. What is the logical address supported by the system in bits.

PAS = 2^{32} B

Page size = 2^{13}

No. of frames = 2^{19} frames = 19 translation bits.

PTE size = 24 bits = 3 B

PTE = $\frac{24 \times 2^{20}}{3} = 8$ M ⇒ No of pages = 2^{23}

LAS = $2^{23} \times 2^{13} = 2^{36}$

LA = 36 bits.

Q. Consider a systems which has LAS = PAS = S Bytes. Page size = P Bytes. PTE size = 'e' Bytes. Then, what is the optimal value of page size by minimizing the memory overhead of maintaining page table size and internal fragmentation in the paging. [mem. is byte addressable]

(a) $P = \sqrt{2Se}$
(b) $P = \sqrt{2Se^2}$
(c) $P = \sqrt{2S^2e}$
(d) $P = \sqrt{2(Se)^2}$

LAS = S Bytes
PAS = S Bytes
Page size = P Bytes
PTE size = 'e' Bytes
No. of frames = $\frac{S}{P}$

Page table size = $\frac{Se}{P}$
No. of pages = $\frac{S}{P}$
PTE size = $\frac{S}{P} \times e$

Memory overhead = Page Table size + Internal fragmentation in paging

$= \frac{Se}{P} + \frac{P}{2}$

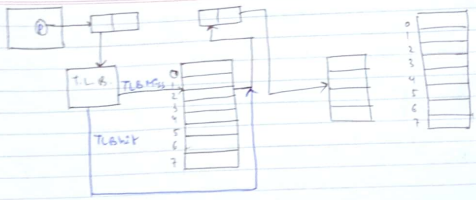
$\therefore \frac{d}{dP}\left(\frac{Se}{P} + \frac{P}{2}\right) = 0 \Rightarrow \frac{-Se}{P^2} + \frac{1}{2} = 0$

$\Rightarrow \frac{Se}{P^2} = \frac{1}{2} \Rightarrow P = \sqrt{2Se}$

Performance of Paging:

⇒ Main memory access time = m
Page tables are stored in the main memory.
Then, the formula for effective memory access time = 2m

⇒ The translation lookaside buffer (T.L.B) is added to improve the performance of paging.
⇒ T.L.B is a hardware device, which is implemented by using associative registers.
⇒ T.L.B access time will be very less compared to main memory access time.
⇒ T.L.B contains frequently referred page numbers and corresponding frame numbers.



⇒ TLB access time = 'c'
TLB hit ratio = 'x'
Then, the formula for effective memory access time

E.M.A.T. = x(c+m) + (1-x)(c+2m)

Q. Consider a system which has main memory access time = 100ns and TLB access time = 20ns, TLB hit ratio = 95%. Then what is the E.M.A.T with TLB and w/o TLB.

w/o TLB:
EMAT = 2×100 = 200 ns

with TLB:
EMAT = 0.95*(120) + (0.05)*(220)
= 114 + 11 = 125 ns

Q. How much hit ratio is required to reduce EMAT from 300ns w/o TLB to 250 ns with TLB. The TLB access time = 60ns.
Mem access time = 150ns

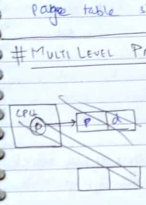
w/ TLB:
x*(210) + (1-x)(360) = 250 ⇒ 210x + 360 - 360x = 250
150x = 110 ⇒ x = 11/15 = 0.7333 = 73% = 73.33%

Q. Consider a system, which has L.A. = 32 bits and page size of LAS. = 4 kW. Pag. PTE size = 1 W. Then what is the page table size.

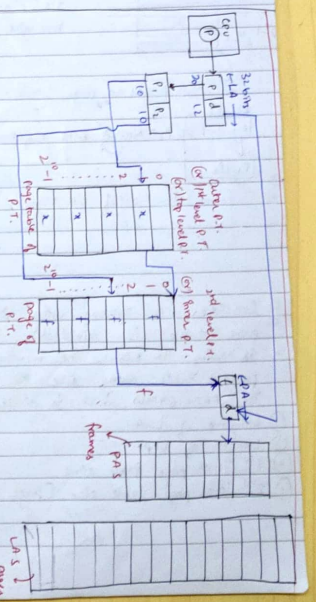
No. of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$ pages.

Page table size = $1 \times 2^{20} = 2^{20}$ word = 1MW.

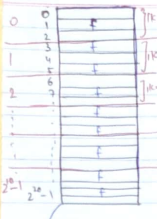
MULTI LEVEL PAGING:



n = Base address of page of P.T.
Page size of P.T. = 1 kW



Page table size = 2^{20} W = 1MW



① ⇒ To avoid the overhead of bringing large size page table into memory, the multi level paging will be implemented.

② ⇒ In multi level paging, paging will be applied on the page table and instead of bringing, the entire page table into memory the pages of page table will be brought into memory.

No. of pages on page table = $\frac{2^{20}\,W}{2^{10}\,W} = 2^{10}$ = 1K

P_1 = No. of bits required to represent pages of page table (or) page number of page table.

P_2 = No. of bits required to represent page size of page table (or) word number of page of P.T. (or) page offset of page table.

Q Consider a system using 2-level paging applicable. The page table is divided into 8K pages and each page is having 4K entries. The memory is word addressable and page table entry size in both the levels = 2 words. The PAS = 64MW, which is divided in 16K frames. Then calculate:

① Length of L.A.
② Length of P.A.
③ 1st level page table size
④ 2nd level page table size [page of page table]

No. of pages of page table = 8K = 2^{11} page

Size of page of P.T. = 4K = 2^{12}

P.T.E. size = 2W

PAS = 2^{26} W
# frames = 2^{14}
Frame size = $\frac{2^{26}}{2^{14}} = 2^{12}$ W

① Length of L.A. = 35 bits
② Length of P.A. = 26 bits
③ 1st level page table size = $2^{11} * 2 = 2^{12}$ W = 4KW
④ 2nd level page table size = $2^{12} * 2 = 8$KW

Q. Consider a system, with 2-level paging applicable. The page table is divided into 8K pages and each page is having 16K entries. Memory is byte addressable. P.T.E. size = 32 bits in both the levels. The PAS = 128MB, which is divided into 4kB frames. Then calculate:

① Length of L.A.
② Length of P.A.
③ 1st level P.T. size
④ 2nd level P.T. size [Page of P.T.]

PAS = 128MB Frame size = 2^{12} B
= 2^{27} B

# of frames = $\frac{2^{27}}{2^{12}} = 2^{15}$

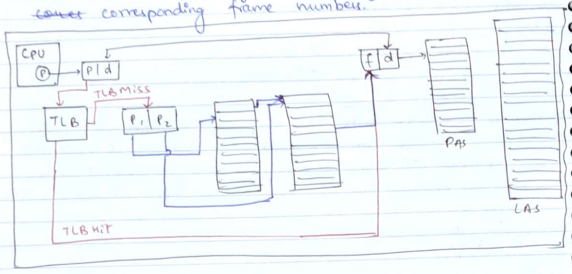
Length of L.A. = ~~42 bits~~ 39 bits

② Length of P.A. = 27 bits
③ 1st level P.T. size = $2^2 * 2^{13} = 2^{15}$ B = 32kB
④ 2nd level P.T. size = $2^2 * 2^{14} = 2^{16}$ B = 64kB

Performance of 2-level paging:

⇒ Main memory access time = 'm'
The page tables are stored in the main memory.
Then, the formula for effective memory access time = 3m

⇒ The T.L.B. is added to improve the performance of paging.
T.L.B. contains frequently refereed page numbers and ~~cover~~ corresponding frame numbers.



⇒ TLB access time = 'c' and TLB hit ratio = 'x'.
Then, the formula for Effective Memory Access Time:

$$E.M.A.T. = x(c+m) + (1-x)(c+3m)$$

⋮

$$E.M.A.T. = x(c+m) + (1-x)(c+(n+1)*m)$$

(n-level paging)

Important Points:

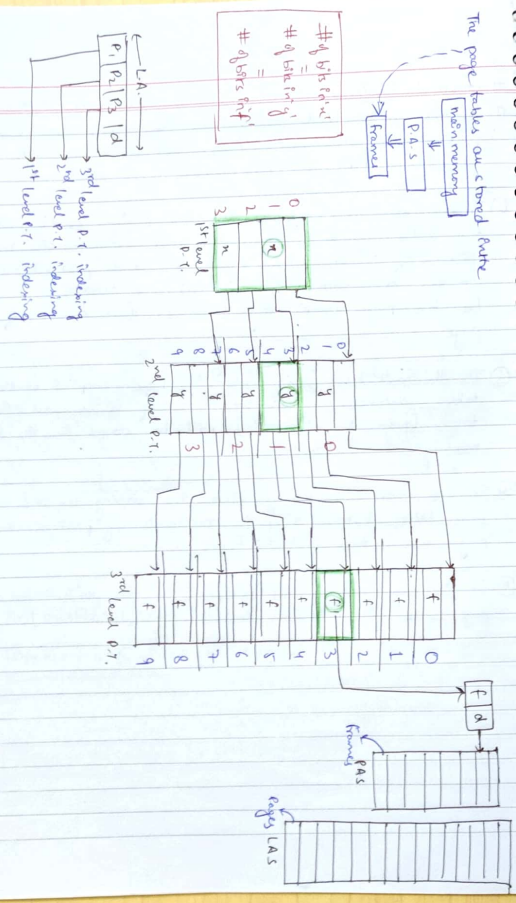
① In the multi level paging when the paging applied on the page tables, the last page table which we get is called as first level page table.

② In the multi level paging, when the paging applied on the page tables, the first level page table entry contains base address of 2nd level page table, 2nd level page table entry contains base address of 3rd level page table and so on. The final page table's entry contains frame number of actual page.

③ In the multi level paging, when the paging applied on the page tables, whatever may be the levels of paging, all the page tables [page of page table] will be stored in the main memory.

④ In the multi level paging, when the paging applied on the page tables, whatever may be the levels of paging, all the page table entries contain frame number.

⑤ If the page size is not mentioned in the problem, generally, page size will be same in all the places [levels].



W.B. Pg-102

Q.25. L.A. = 32 bit

Page size = 1024 B = 2^{10}

PTE size = 4 B = 32 bits

No. of 2^{nd} level ~~page~~ P.T. entries = $\frac{2^{22}}{2^{10}} = 2^{12}$

LA=32 bits

| 10 |

4B
4B
2^{12} entries
1 page

1st time paging:

P.T. size = $\left(\frac{2^{32}}{2^{10}}\right) * 4$ B

= $2^{22} * 2^2$ B = 2^{24} B

= 16 MB

P.T. size ≤ Page size

16MB ≰ 1kB

2nd time paging:

P.T. Size = $\left(\frac{2^{24}}{2^{10}}\right) * 4$ B

= $2^{14} * 2^2$ B = 2^{16} B

= 64 kB

64kB ≰ 1kB

3rd time paging:

P.T. size = $\left(\frac{2^{16}}{2^{10}}\right) * 4$ B

= $2^6 * 2^2$ = 2^8 B

= 256 B ← 1st level P.T. size

256B ≤ 1kB ✓

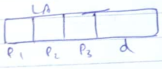
Ⓒ

3 levels of paging required.

2 marks.
GATE-2015

Q. Consider a system, L.A. = 46 bits ; P.A. = 32 bits and w/m is byte addressable and P.T. entry size = 32 bits. The O.S. uses 3 level paging for logical to physical address translation and 1st level P.T. size is exactly same as page size. Then what is page size.

(a) 2 KB
(b) 4 KB
(c) 8 KB
(d) 16 KB



- L.A. = 46 bits
- P.A. = 32 bits
- PTE size = 32 bits = 4 B

~~1st page size = ??~~

1st level page table size = P = Page size
No. of entries in 1st level P.T. = $\frac{P}{4}$
2nd level P.T. size = $\frac{P^2}{4}$

1st time paging = $\frac{2^{46}}{x} * 4B = \frac{2^{48}}{x}$ B = P.T. size

2nd time paging ⇒ P.T. size = $\left(\frac{\frac{2^{48}}{x}}{x}\right) * 4B = \frac{2^{50}}{x^2}$ B

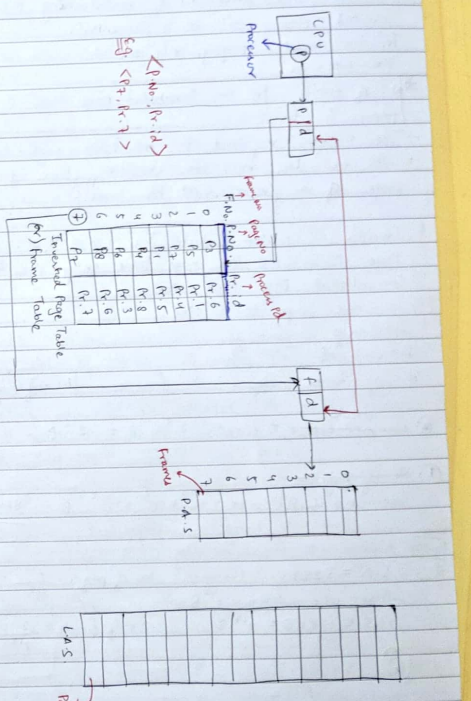
3rd time paging ⇒ P.T. size = $\left(\frac{\frac{2^{50}}{x^2}}{x}\right) * 4B = \frac{2^{52}}{x^3}$ B ← 1st level P.T. size

$\frac{2^{52}}{x^3} = x$ ⇒ $x^4 = 2^{52}$ ⇒ $x = (2^{13})^{\frac{1}{4}}$

$x = 2^{13}$

⇒ Page size = $xB = 2^{13} B = 8$ KB

Inverted Paging



⇒ To avoid the overhead of maintaining P.T. for every process, the inverted paging will be implemented.
In the inverted paging, only one P.T. will be maintained for all the processes.

* No. of entries in the inverted page table is same as the no. of frames in the P.A.S.

→ The memory required to maintain page tables of the process will be less but the searching time for the corresponding page of a process will be more.

Q. Consider a system which has L.A. = 34 bits and P.A. = 29 bits. Page size = 16 KB. The page table entry size is 8 B. Then calculate
① Conventional Page table size.
② Inverted page table size.

P.A.S = 2^{29} B
No. of frames = $\frac{2^{29}}{2^{14}} = 2^{15}$ frames.
No. of pages = $\frac{2^{34}}{2^{14}} = 2^{20}$

② Inverted ~~Conventional~~ P.T. size = $2^{15} * 8 = 2^{18}$ B = 256 KB

① ~~No. of pages =~~
Conventional P.T. size = $2^{20} * 2^3 = 2^{23}$ B = 8 MB

W.B:
Pg 102
Q 2.3. L.A. = 32 bits
Page size = 4 KB = 2^{12} B
PAS = 128 KB = 2^{17} B
No. of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$
No. of frames = $\frac{2^{17}}{2^{12}} = 2^5$

④

W.B
Page 100
Q.5. Page size = 4 KB
L.A. = 32 bits
P.A. = 30 bit
Overhead bits = 12
No. of frames = $\frac{2^{30}}{2^{12}} = 2^{18}$ frames.
No. of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$ pages. ∴ 20 bits → bits to represent page no.
Inverted PTE size = 20 + 12 = 32 bits = 4 B
Inverted P.T. size = $2^{18} * 4 = 2^{20}$ B

Segmentation

→ Paging does not follow, user's view of memory allocation.
→ To achieve user's view of memory allocation, the segmentation will be implemented.
→ In the segmentation, LAS will be divided into various segments
→ Segments of LA.S. will vary in the size.
→ Segments of L.A.S. will be brought into P.A.S.
S = No. of bits required to represent segments of L.A.S. (or) segment number
d = No. of bits required to represent segment size (or) word number of the segment (or) segment offset.

⇒ No. of entries in the segment table is same as number of segments in the L.A.S.

NOTE
→ Variable size segments are brought from L.A.S. to P.A.S., so, it is behaving similar to variable partition scheme. Hence, segmentation still suffers from external fragmentation.

Segmentation :

Eg: ① d=210 L=200, S=2
d<L
210<200 ✗
Referring beyond the segment or illegal address.

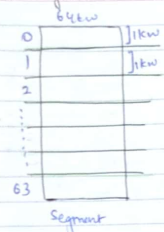
② d=190 L=200
d<L
190<200 ✓
Valid Request.
i) BA+d->P.A.
580+190=770

W.B Pg-102 : d
Q.24. 3 : 722
Physical address = 498 + 222
= 720

Segmented Paging :

Paging on segment, when segment size increases
x = Base address of page table of respective segment
page size of segment = 1 KB

Segment size = 2^{16} W = 64 kW



→ To avoid the overhead of bringing large size segment into memory, the segmented paging will be implemented.

→ In the segmented paging, paging will be applied on the segment and instead of bringing the entire segment into memory, the pages of segment will be brought into memory.

No. of pages on segment = $\frac{\text{segment size}}{\text{page size}} = \frac{64\text{kw}}{1\text{kw}} = 64$

⇒ No. of entries in the page table of segment is same as no. of pages on segment.

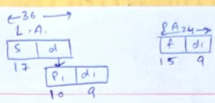
P_1 = No. of bits required to represent pages of segment (or) page number of segment

d_1 = No. of bits required to represent page size of segment (or) word no. of page of segment or page offset of segment.

* Page size of segment = Frame size of P.A.S.

Q① Consider a system using segmented paging architecture. The segment is divided into 1k pages and each page is having 512 entries. The segment number number requires 17 bits to represent all the segments of L.A.S. and frame no. requires 15 bits to represent all the frames of P.A.S. & memory is word addressable and PTE size = 2W. Then, we calculate

① Length of L.A.
② Length of P.A.
③ P.T. size of segment.



Page size = 512 W = 2^9 W No. of frames = 2^{15}

No. of pages = 2^{10}

No. of segments = 2^{17}

Segment size = $2^{10} * 2^9 = 2^{19}$ W

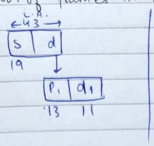
① Length of L.A = Segment size * No. of segments.
= $2^{19} * 2^{17} = 2^{36}$
= 36 bits

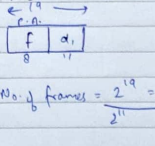
② Length of P.A = ~~No. of frames~~ * No. of frame bits + ~~No. of segs~~ Page size bits
= 15 + 9 = 24 bits.

③ P.T. size of segment ⇒ PT entries # * PTE size
= $2^{10} * 2$
= 2^{11} = 2 kW

Q. Consider a system using segmented paging architecture. The segment is divided into 8k pages and each page is having 2k entries. The segment no. requires 19 bits to represent all the segments of L.A.S. Memory is Byte addressable and P.A.S = 512kB. P.T.E size = 8 bits Then calculate:

① Length of L.A. = 43 bits
② Length of P.A. = 19 bits
③ Page Table size of segment = 8kB
④ No. of frames in the P.A.S = 2^8 = 256 frames.





No. of frames = $\frac{2^{19}}{2^{11}} = 2^8$

P.T. size = $2^{13} * 1$
= 8kB

WB Pg102

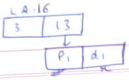
Q88. LAS = 2^{16} B

PAS = 2^{16} B

No. of segments = 2^3

Page size = x

No. of pages = $\frac{2^{13}}{x}$



PTE size = 2B

PT. size = $\frac{2^{13}}{x} * 2$

$= \frac{2^{14}}{x}$

PT. size = x

$2^{14} = x^2 \Rightarrow (2^7)^x = x^2 \Rightarrow x = 2^7$

$= 128$ B

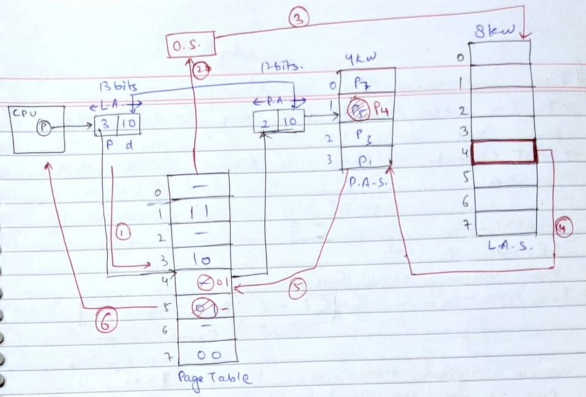
Virtual Memory:

→ Virtual memory gives an illusion to the programmer that the programs larger size than actual physical memory can be executed.

→ Virtual memory will be implemented by using demand paging (or) demand segmentation.

Demand Paging:

→ Loading the page into memory on demand [whenever page fault occurs] is called as demand paging.



Step-①: CPU is trying to refer the page in the main memory but the page is currently not available [This is called as page fault].

Step-②: Program execution will stop and signal will be sent to the O.S., regarding the page fault.

Step-③: O.S. will search for the required page in the L.A.S.

Step-④: • Required page will brought from L.A.S. to P.A.S.
• The page replacement algorithms will be used for the decision making of selective the page for the replacement.

Step-⑤: Respective page table entry will be updated accordingly.

Step-⑥: Signal will be sent to the CPU to continue program execution and the CPU will access the required page in the main memory and continue program execution.

Page fault service Time: (PFST) — practically in ms

→ The time taken to service the page fault is called as page fault service time (PFST).
→ PFST includes the time to perform all the above 6 steps.

PFST = 's'
PF rate = 'p'
Main Memory access time = 'm'
→ Then the formula for EMAT = p x s + (1-p) x m
s >> m

Q. Consider a system MMAT = 15ns, PageFST = 175 ns
Page Hit ratio = 75%. Then what is the EMAT?
PFrate = 0.25
EMAT = 0.25 x 175 + 0.75 x 25
= 70ns.

WB Pg 104
Q.20. PFrate = 1/k PFST = i+j MAT = i

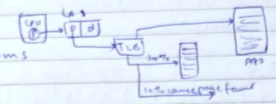
EMAT = 1/k x (i+j) + (k-1)/k x i ⇒ (i+j+ki-i)/k
⇒ (ki+j)/k x p + j/k

~~For k instructions~~
WB Pg 109
Q.23. PFST = 10ms MAT = 20ns
PF rate = 1/10^6

EMAT = 1/10^6 x 10x10^-3 + (10^6-1)/10^6 x 20x10^-9
= 10ns + 20ns = 30ns

WB Pg 106
Q.22 PFST = 8ms PFST_m = 20ms MMAT = 1ms
Modified = 2%.
EMAT = 2 ms
EMAT = p [0.7 x 20 + 0.3 x 8] + (1-p) [1]
2 = p [14 + 2.4] + 1 - p
2 = 16.4p + 1 - p ⇒ 15.4p = 1
p = 1/15.4 = 0.064

Pg 107
Q.21. PFST = 200ms MMAT = 10ms
TLB Hit ratio = 0.8



EMAT = 0.8(10) + 0.2(200) + 0.2x0.9(10)
= 8 + 40 + 1.8
= 13.8 ms

EMAT = 0.8(0+10) + 0.2(0+0.1 x 200 + 0.9 x (10))
= 13.8 ms

Page Replacement Algorithms:

Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

↳ Page numbers refered in the L.A.

NOTE

↳ No. of frames allocated to the process is decided by Instruction set architecture.

FIFO (First in First Out):

of frames = 4

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
			2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
		1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	✗	4	4	4	4	4	4	4	4	✗	7	7	7	
7	7	7	7	✗	3	3	3	3	3	3	3	✗	2	2	2	2	2	2	
F	F	F	F		F		F			F	F		F				F		

: 10 Faults

of frames = 3

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		1	1	1	✗	0	0	✗	3	3	3	✗	2	2	2	✗	1	1	
	0	0	0	✗	3	3	✗	2	2	2	2	✗	1	1	1	✗	0	0	
7	7	✗	2	2	2	✗	4	4	✗	0	0	0	0	0	0	7	7	7	
F	F	F	F		F	F	F	F	F			F	F			F	F	F	

: 15 Faults

R.S. → 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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

= 9 faults.

4 frames:

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

= 10 Faults.

* Belady's Aramoly:

→ By increasing, no. of frames to the process, no. of page fault should decrease but instead, they are increasing. This problem is called as Belady's Aramoly.

Optimal Page Replacement:

→ In the event of page fault replaces the page which is not used for longer duration of time in future.

4 frames →

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
			2	2	2	2	2	2	2	2	2	2	2	2	2	✗	7	7	7
		1	1	1	1	✗	4	4	4	4	4	4	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	✗	3	3	3	3	3	3	3	✗	1	1	1	1	1	1	1
F	F	F	F		F		F					F				F			

8 faults. =

LIKE FIFO

3 frames:

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		1	1	1	✗	3	3	3	3	3	3	✗	1	1	1	1	1	1	1
	0	0	0	0	0	✗	4	4	✗	0	0	0	0	0	0	0	0	0	0
7	7	✗	2	2	2	✗	2	2	2	2	2	✗	2	2	2	✗	7	7	7
F	F	F	F		F	F			F			F			F			F	

9 faults =

Least Recently Used (LRU) Page replacement:

→ In the event of page fault, we replace the page which is least recently used.

3 frames

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

= 12 faults

4 frames:

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

= 8 faults.

Most Recently Used (MRU) Page replacement:

→ In the event of page fault, replace the page which is most recently used.

4 frames:

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

= 12 faults

3 frames:

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

= 16 faults.

WB Pg107

Q.17. Reference string = 1, 2, 4, 5, 5, 5, 1, 2, 2, 2, 3, 3 (4 needed to write again)

Record no.	Page ff
0-99	0
100-199	1
200-299	2
300-399	3
400-499	4
500-599	5

1 X 4 5 1 2 3 = 7 faults. (a)

R.S → 1, 2, 4, 5, 1, 2, 3

↳ R.S does not contain same page twice.

NOTE

① R.S. ~~str~~ never contains continuous same page.

Eg 1, 2, 3, 4, 4, 3, 2, 1 (X)

1, 2, 3, 4, 3, 2, 1 ✓

② If there is only one frame allocated to the process, then, every page referred will be a page fault.

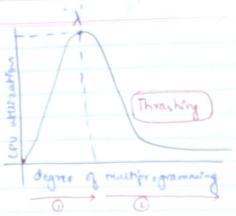
WB Pg106

Q.12. (c) m, n

minimum page faults = m (unique pages)

maximum page faults = n (total pages referred)

Thrashing:



Free Frames: 400

① # of processes: 100
Each process 'P' gets = 4 frames.

② # of processes: 400
Each process 'P' gets = 1 frame

→ In the initial, degree of multiprogramming upto the point of 'λ', the CPU utilization is very high and system resources are utilized perfectly.

→ If we further increase the degree of multiprogramming, the CPU utilization will drastically fall down and the system will spend more time only in the page replacement and the time taken to complete the execution of the process increases.
This situation in the system is called as Thrashing.

Causes of Thrashing:
① High degree of multiprogramming
② Lack of frames

L.T.S. = long term scheduler.

Recovery of Thrashing:
① Do not allow the system to go into thrashing and instruct the L.T.S. not to bring the processes into the memory after the point of 'λ'.
② If the system is already in thrashing, then instruct the mid term scheduler to suspend some of the processes, so that we can recover the system from thrashing.

Q Consider the system with the following parameters:
CPU utilization = 10% Paging disk = 90%
Then which of the following factors will improve the CPU utilization?
(a) Install bigger disk NO
(b) Install faster CPU NO
(c) Increase degree of multiprogramming NO
(d) ~~Dog~~ Decrease degree of multiprogramming. YES
(e) Install more main memory ? YES
(f) ~~Dog~~ Decrease page size NO (Assume frame size is constant)
(g) Increase page size LIKELY YES

Paging disk = Time spent on page replacement

02/11/2017

File Systems:

⇒ File: collection of logically related entities (records). File will have various attributes.

Attributes:

1. Name
2. Type
3. Size
4. location
5. Creation date
6. Last modified date
7. Permissions
8. Owner
9. Password

→ All the attributes of the file, is called as File context.
→ File context will be stored in the File Control Block (FCB).
→ Every file will have its own FCB.

* Types:

1. .doc
2. .txt
3. .dll
4. .exe
5. .obj
6. .class
7. .bat
8. .pdf
9. .png
10. .apk
11. .xls | .xlsx
12. .c | .cpp | .java | .html | .php | .xml | ...
13. .mp3 | .mp4
14. .avi | .mkv | .flv
15. .bmp

→ Various operations will be performed on files:

Operations:

1. Create
2. Open
3. write
4. Read
5. Edit
6. delete
7. close
8. save
9. copy
10. paste
11. cut
12. Undo | redo
13. print
14. Send
15. Save as
16. Rename

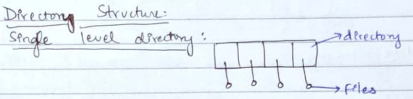
Access Methods:

1. Sequential Access
2. Random Access

→ For better classification of files, files will be stored in the directory.

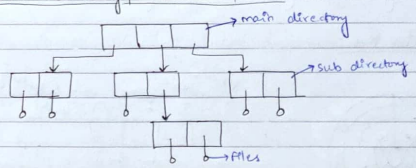
Directory Structure:

1. Single level directory:



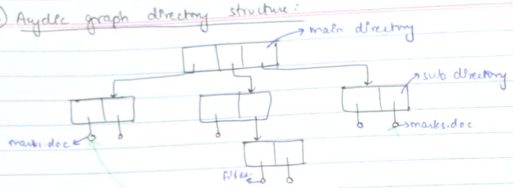
⇒ Implementation of the directory structure is easy/simple.
⇒ Two files cannot have the same name.
⇒ ~~Se~~ Searching time for the specific file will be more.

2. Multi level directory / Hierarchical / Tree:



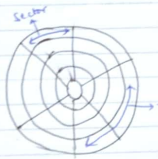
⇒ Implementation is difficult.
⇒ Better classification of the files as per the criteria.
⇒ Searching time for the specific file will be less.
⇒ If the same file exists in two different directories, then, if one file is updated, the other file has to be updated accordingly. Otherwise, there will be a inconsistency.

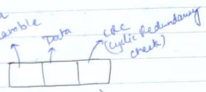
③ Acyclic graph directory structure:



⇒ The Implementation of the directory structure is difficult.
⇒ Better classification of the files as per the criteria.
⇒ The searching time for files will be less.
⇒ If the same file exists in two different directories, then, if one file is updated, then other file will be updated automatically by using file sharing concept.

Disk Structure:





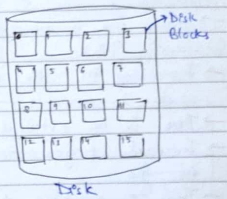
Q. Consider a disk which has 16 platters, each platter is having 2 surfaces. Each surface is divided into 1k tracks. Track is divided in 512 sectors and each sector can store 2kB data. Then calculate:
① Capacity of the disk
② How many bits are required to identify specific sector of a disk.

① Capacity = $(2^4 * 2 * 2^{10} * 2^9 * 2^{11})$ B
= 32 GB

② Bits for sector = 4 + 1 + 10 + 9
= 24 bits

Disk space allocation methods:
* Contiguous Allocation:

File	Starting D.B.A (Disk Block address)	Size (cont. # of disk blocks)
abc.doc	2	4
xyz.doc	9	5

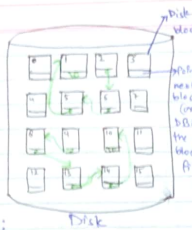


→ In contiguous allocation, whenever a file is created, the disk blocks will be allocated to it in a continuous manner.
→ Every file is associated with two parameters:
① Starting D.B.A ② Size
→ Increasing the file size may not be possible always.
→ It is suffering from external fragmentation.
→ Internal fragmentation may exist in the last disk block of the file.
→ It supports both sequential and random access of the file.
→ Starting D.B.A. + offset ⇒ Random access.

Eg: 9 + 4 = 13 → offset value for 5th block

Linked Allocation [Non Contiguous]:

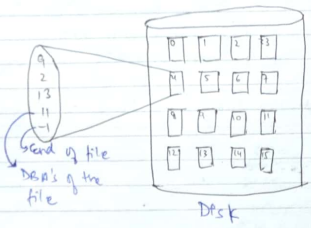
File	Starting DBA	Ending DBA
abc.doc	2	1
xyz.doc	9	10



→ In the linked allocation, disk blocks will be allocated in a non-contiguous manner.
→ Every file is associated with 2 parameters:
① Starting DBA ② Ending DBA
→ Increasing the file size is always possible if the free disk block is available.
→ There is no external fragmentation.
→ Internal fragmentation may exist in the last disk block of the file.
→ There is overhead of maintaining the pointer in every disk block.
→ If the pointer of any disk block is lost, file will be truncated.
→ It supports only sequential access of the file.

Indexed Allocation

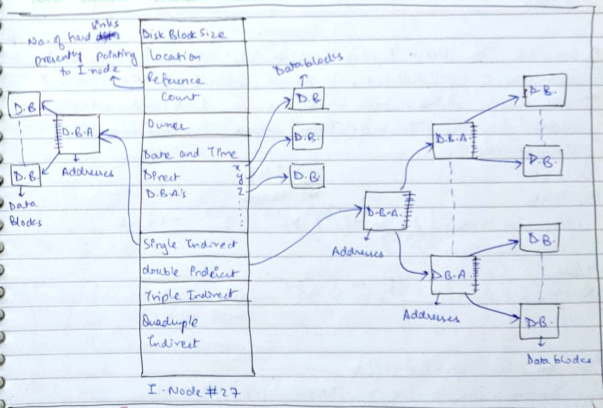
File	Index Node
abc.doc	4



→ In indexed allocation, every file is associated with its own index node.

→ Index node contains all the disk block addresses (DBAs) of the file.
→ There is no external fragmentation, but internal fragmentation may exist in the last disk block.
→ If the file is very very large, then, one disk block may not be sufficient to store all the disk block addresses.
→ If the file is very very small, then, it is waste of using entire one disk block, just to store the addresses.

UNIX-I-NODE Implementation:



Total size of the file system = [# of Direct DBAs + (DB size / DBA) + (DB size / DBA)^2 + (DB size / DBA)^3 + ...] × DB / Size

File	I-NODE
abc.doc	27

# of single indirect DBAs
# of double indirect DBAs
# of triple indirect DBAs

→ Unix follows I-NODE way of implementation. In order to allocate the disk space to the files.
→ Every file is associated with its own I-node.
→ Depending on the size of the file, file will be stored in one particular place, maybe in the direct disk block addresses, or maybe in the single indirect or maybe in the double ~~indirect~~ indirect or may in the triple indirect and so on.

DB size / DBA = # of DBAs, one disk block can contain

Q. Consider the UNIX I-node which has 10 direct disk block addresses, 1 single indirect, 1 double indirect, 1 triple indirect disk block addrs. Disk block size is 1 KB and disk block address requires 32 bits. Then what is maximum file size possible.

(a) 2 GB (c) 8 GB
(b) 4 GB (d) 16 GB

Max file size = $\left(\frac{2^{10}}{2^2}\right)^3 * 2^{10} = (2^8)^3 * 2^{10} = 2^{34} = 16$ GB

WB Pg 114
Q16. Max file size = $\left(\frac{2^9}{2^2}\right)^3 * 2^9 \Rightarrow 2^{30} = 1$ GB

Q12. DB size = 1024 B = 2^{10} B
DB size / DBA = 128 ⇒ 1024 / 128 = DBA size ⇒ 2^3 = DBA size
Max file size = $\left(\frac{DB\ size}{DBA}\right)^3 * DB\ size = \left(\frac{2^{10}}{2^3}\right)^3 * 2^{10}$
$= 2^{31} \Rightarrow 2$ GB (b)

WB Pg 113
Q.7. DB size = 1 KB = 2^{10} B
Max file size = 4 TB = 2^{42} B

$\left(\frac{DB\ size}{DBA}\right)^4 * DB\ size = 2^{42} \Rightarrow \left(\frac{DB}{DBA}\right)^4 = 2^{32}$

$DB\ (2^8)^4 = \left(\frac{DB\ size}{DBA}\right)^4$

$2^8 = \frac{DB\ size}{DBA} \Rightarrow \frac{2^{10}}{2^2}$ → DBA = 4 Bytes = 32 bits.

$2^8 = 2^{10}$ / 2^2

Disk Free Space Management:

Size of disk = 20 MB
Disk block size = 1 KB
DBA = 16 bits

# of disk blocks available on the given disk = Size of Disk / Disk block size

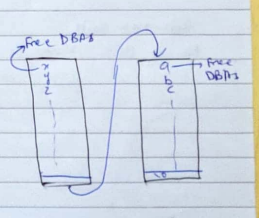
$= \frac{20\ MB}{1\ KB} = \frac{20 * 2^{20}}{2^{10}} = 20$ K

① Free list approach:

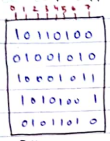
→ In the free list approach, some disk blocks are used just to store the free disk block addresses.

① # of DBAs possible to store in one disk block = DB size / DBA = 1 KB / 2 B = 512

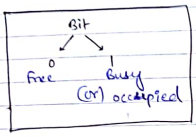
For 20 K ⇒ $\frac{20 * 2^{10}}{2^9} = 40$ Disk blocks.



② Bit Map approach:



Bit map



Free disk blocks:
1, 4, 6, 7, 8, 10, 11 ...

Busy disk blocks:
0, 2, 3, 5, 9, 12 ...

→ In bitmap approach, every disk block will be to mapped with one binary bit.

① To ~~make~~ map 20K disk blocks, we need 20K bits.

② $\frac{20K\ bits}{1KB} = \frac{20K\ bits}{1K \times 8\ bits} = 2.5 \approx 3$ disk blocks

WB Pg114

Q.17. Hard disk = 40 MB = 40×2^{20} B

DB size = 2KB = 2^{11} B

DBA = 3B

No. of disk blocks = $\frac{40 \times 2^{20}}{2^{11}}$ = ~~$2^{9} \times 10$~~ $2^2 \times 2^9 \times 10 = 20K$

Bits = 20K bits

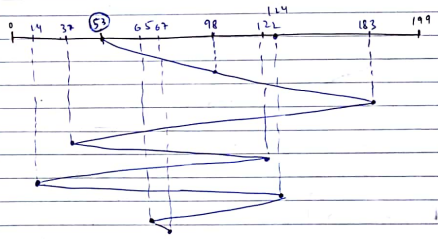
No. of disk blocks to store bit map = $\frac{20K}{2^{11} \times 8} \Rightarrow \frac{10}{8} \approx 2$

Disk Scheduling:

Track Requests: 98, 183, 37, 122, 14, 124, 65, 67

Goal: To minimize the average seek time of disk.

① First Come First Serve (FCFS):



* Total track movements made by r/w header (Seek Time):

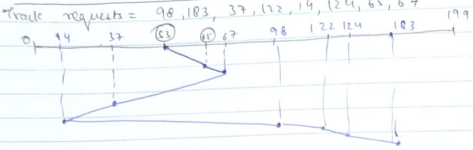
$= (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)$

$= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2$

$= 130 + 220 + 231 + 59 = 290 + 170 + 130 = 640$

② Shortest Seek Time First (SSTF): Nearest track next

Track requests = 98, 183, 37, 122, 14, 124, 65, 67

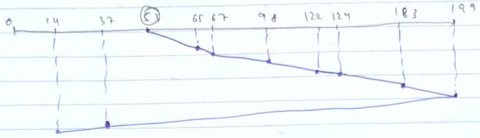


= 236

= 12+2+30+23+84+24+2+59

③ SCAN scheduling / Elevator:

Track requests = 98, 183, 37, 122, 14, 124, 65, 67

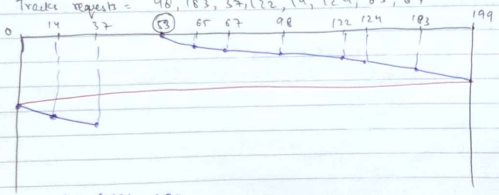


= 12+2+31+24+2+59+16+162+23

= 90+~~80~~ 10 + 201 = 331

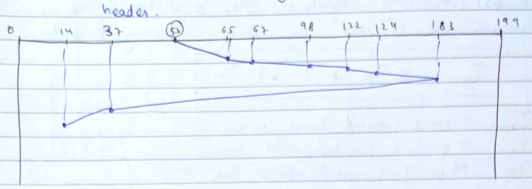
④ C-SCAN (Circular SCAN):

Tracks requests = 98, 183, 37, 122, 14, 124, 65, 67



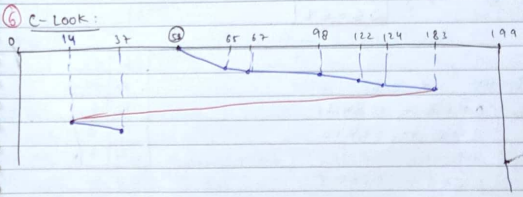
= 146+37+199 = 382

⑤ LOOK: Look for last pending request in the direction of r/w header.



= 130+146+23 = 299

⑥ C-LOOK:



= 130+23+169 = 322

CPU SCHEDULING

NOTE

↳ In SJF scheduling algorithm the burst time of the processes are expected by using following formula.

$$\tau_{n+1} = \alpha \cdot T_n + (1-\alpha) \cdot \tau_n$$

τ_{n+1} → Next expected burst time
τ_n → previous expected burst time
T_n → previous actual burst time.

'α' is a parameter which controls relative weight of recent and past history.

$0 \le \alpha \le 1$

WB Pg 82

Q.32. $\tau_1 = 10$ $\alpha = 0.5$

$\tau_2 = 0.5 * T_1 + 0.5 * \tau_1$
$= 0.5 * 5 + 0.5 * 10$
$= 7.5$

$\tau_3 = 0.5 * 8 + 0.5 * 7.5$
$= 4 + 3.75$
$= 7.75$

$\tau_4 = 0.5 * 3 + 0.5 * 7.75$
$= 1.5 + 3.875$
$= 5.375$

$\tau_5 = 0.5 * 5 + 0.5 * 5.375$
$= 2.5 + 2.6875$
$= 5.1875$

P.no	B.T
P_1	5
P_2	9
P_3	3
P_4	5

Q.33 $\alpha = 1$

$\tau_2 = T_1$ $\quad T_5 = T_4$

$\tau_3 = T_2$ $\quad T_1 - \tau_5 = 5$

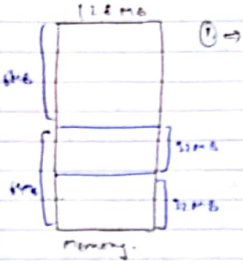
$\tau_4 = T_3$

WB Page 92

Q.14. (b) $n = 21$, $k = 12$

$P_9 \Rightarrow R_{12}, R_{10}$
$P_{10} \Rightarrow R_{10}, R_{12}$

BUDDY SYSTEM:



(1) ⇒ 13MB → In the buddy systems, initially memory will be single continuous free block.

→ whenever the request by the process arrives, memory will be divided into 2 half blocks.

→ If the request by the process is too small, then, the lower blocks of the memory is further divided into two half blocks again.

→ In the buddy system, memory will be allocated from lower blocks to higher blocks.

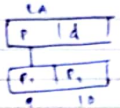
WB Pg 102

Q.27. PA = 36 bits
Page size = 4KB

No. of frames = 2^{24}
bits = $\frac{24}{24, 24, 24}$

Pg 103

Q.32.



1st level PT size + 2nd level PT size
$= 2^9 * 2 + 2^{10} * 4$
$= 1K + 4KB$
$= 5KB$