Operating Systems

Contents:

I Introduction and Background

II Process management
- process Concept
- CPU scheduling (40%)
- Synchronization
- Concurrent programming
- Deadlocks
- Threads

III Memory Management.
- → RAM chip Implementation
- → Loading Linking and Address Binding
- → Techniques Techniques (40%)
  - → paging
  - → multilevel paging
  - → Inverted paging
  - → Segmentation
  - → Segmented Paging
- → Virtual Memory

IV File System.

Text books

1. OS by Galvin
2. Modern OS by A.S Tenenbaum
3. O.S by William Stalling

avg → 8 to 10 (Marks)

faculty: Bala krishna Sir

EMAIL ID

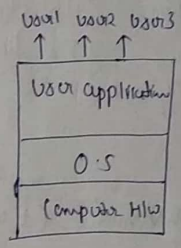
BALAKRISHNA.VEERALA@gmail.com

10/9/18

Introduction & Background

What is an OS?

OS is an interface b/w user and computer h/w.



User application

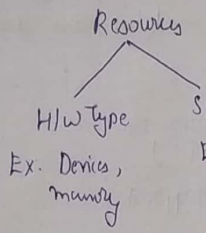
```
main()
{
  int x
  printf("Hello");
}
```

printf internally calls write() System call in order to communicate with the monitor.

System Call

System call is a request made by the user prg to the O.S in order to get any kind of service.

→ OS is also called as Resource allocator bcz it is responsible of allocating resources of a computer

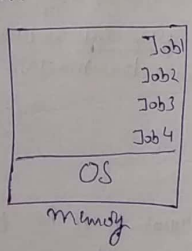


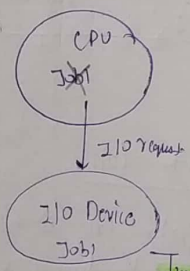
Goals of OS

1. Primary goal is Convenience (Easy to use)
2. Secondary goal is Efficiency (Stability)

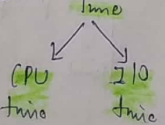
Types of OS

1) Batch OS.





for any job to complete it requires time

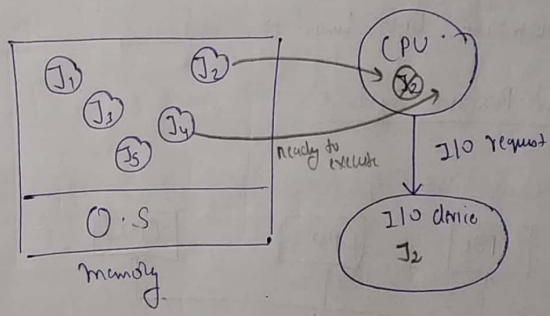


→ If the current job is completed completely then only another job will be scheduled on to CPU.
→ Increased CPU idleness.
→ Decreased Throughput of the system.

Throughput: No of jobs completed for unit time. is called throughput of the system.

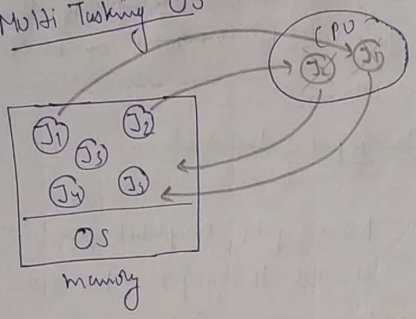
Ex: IBM OS/2

2) Multiprogramming OS



→ if the job is leaving the CPU to perform I/O opn then another job which is ready for execution will be scheduled on to the CPU.
→ Increased CPU Utilization
→ Increased Throughput of the system
Ex: Windows, Unix, Linux

3) Multi Tasking OS

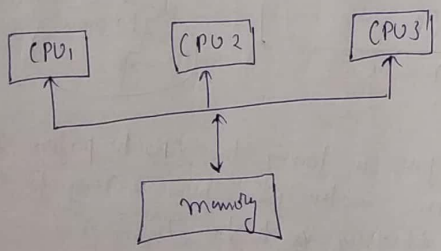


Jobs execute for 2ns then next job schedule

→ Multi tasking is an extension of multi programming OS.
→ The jobs will be executed in the time sharing mode.

Ex:- Windows, Unix, Linux etc.

4) Multi Processor Systems



→ Increased throughput of the system.
→ Reliability
  → Fault tolerant Systems

3. Economical (purchasing 1 system with n CPUs is more economical than purchasing n systems each with 1 CPU)

Ex:- Unix

5. Real Time Systems

The systems which are strict deadly time bound are called as real time system

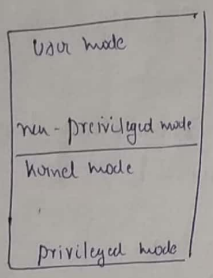
Real Time System

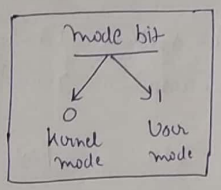
Hard real Time
Eg:- Satellite Systems
missile Systems
minor delay not accepted.

Soft real time
Eg:- banking sector
minor delay accepted

Ex: Sx works, Vx works, RTOS.

Dual Mode Operation





→ In the H/w level instn's are executed by using dual mode operation like
  i) User mode / non-privileged mode
  ii) Kernel mode / privileged mode / ~~monitor~~ monitor mode / System mode / Supervisory mode

→ The dual mode operation is used, in order to provide the protection & security to the user programs and also to the Operating System from "errant users"

→ It is purely the decision of O.S. in which particular mode, the instn has to be executed.

→ generally the previleged instn are executed in the Kernel mode & non-privileged instn's are executed in the user mode.

→ The mode ~~bit~~ bit is used to identify in which particular mode current instn is executing.

→ In the boot time the system always starts only in the Kernel mode.

→ The Operating Systems always runs only in the Kernel mode

NOTE: The mode switching takes very less time compared to process switching.

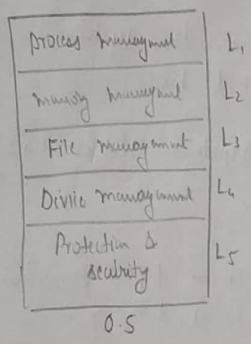
Privileged Instructions

1) I/O operation
2) Context Switching
3) Disabling the interrupts
4) Set the time of clock
5) Clearing the memory map (ie deallocating memory from process)
6) Changing memory map (ie moving process from one memory location to another)

Non-privileged Instructions

1) Reading the time of clock
2) Reading the state of the processor
3) Sending the final print o/p to the printer

Layered Approach



→ The design and implementation of the O.S will be done in multiple layers.

→ The main advantage of layered approach is modularity, abstraction & debugging

→ Modularity means, if any specific layer is updated then it will not have any impact in the other layer.

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 -ve value child process creation is unsuccessful.

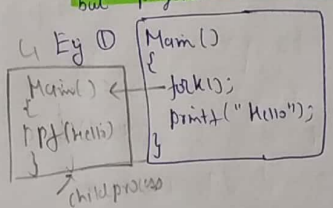
→ fork() returns the value 0 to the newly created child process.

→ fork() returns +ve value (pid of child process) to the parent process.

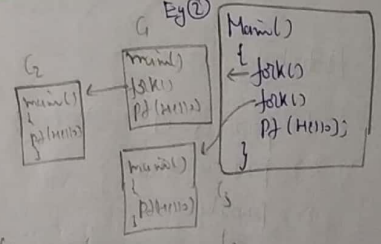
→ When the child process is created, new memory location will be allocated to the child process.

→ The parent and child processes will have same virtual address but physical addresses will be different.

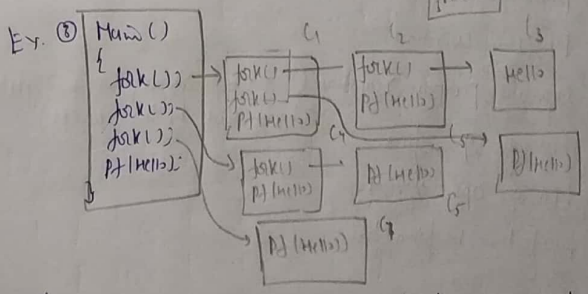
Eg ①



Eg ②



Eg ③



NOTE: If the program is having n fork() calls then it will create $2^n - 1$ child processes

Process Management

Process Concept

Def:- The prog under execution is called as process

- → It should reside in the Main memory
- → It is occupied the CPU to execute the instructions
- → Active and dynamic
  - ↑ bcz it is executing instr.
  - ↑ take some i/p & produce o/p

Attributes :-

1) process id
2) process state
3) priority
4) program counter
5) General purpose registers. (Keeps info about which registers are being used by the process)
6) List of open files
7) List of open devices
8) protection information

1) Process Id

process id is unique identification no. which is assigned by the OS. at the time of process creation.

2) Process state.

process state contains current state information of a process where it is residing.

3) Priority

priority is a parameter assigned by the OS at the time of process creation.

4) Program counter

Prog. counter contains address of the next instn to be executed.

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

→ All the context of the process will be stored in the process control block (P.C.B)

P.id	P.S
P.C	priority
G.P.R	L.O.F
L.O.D	protection information

P C B

→ Every process will have its own P.C.B.

→ P.C.Bs of the processes will be stored in the main memory.

→ P.C.Bs are implemented using Doubly linked list.

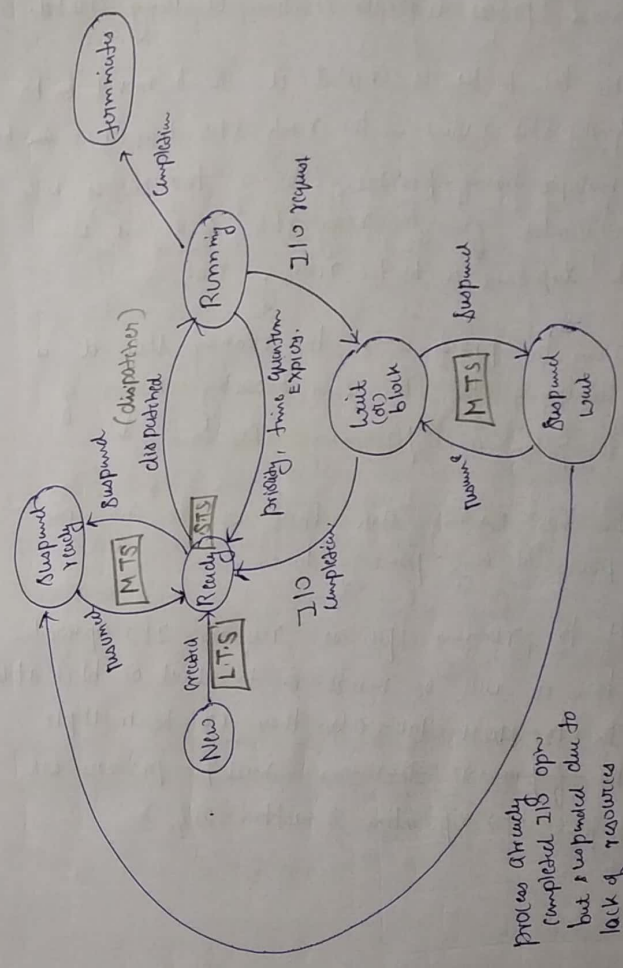
→ Process is having various diff. states:

1) New state
2) Ready state
3) Running state
4) Wait/block state
5) Termination or completion
6) Suspend ready
7) Suspend wait or suspend block

Various operations performed on the process

1) Creation
2) Scheduling
3) Dispatching
4) Executing
5) terminating or killing
6) Suspending
7) Resuming

Process State Diagram



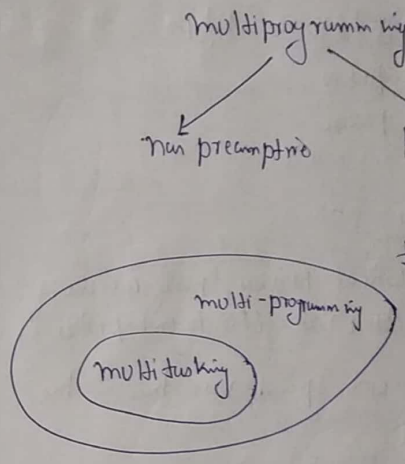
→ Initially process will be in the new state, it means process is under creation or being created.

→ Once the process is created it will be moved on to Ready state. And in the ready state there will be multiple no. of processes. One of the process will be selected from the ready state and that will be dispatched on to the running state.

→ When the process is in the running state it is occupying the CPU and executing instn's 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 I/O operation then it will be moved on to wait or block state. In the wait state also there can be multiple no. of processes; it means multiple processes will perform I/O operation simultaneously.



→ When the process is in the Ready, Running or wait/block state then it is residing in the Main memory.

→ When the resources are not sufficient to manage the processes, in the ready state (eg:- memory) then some of the processes will be suspended and they will be moved onto suspend ready state.

→ When the process is in the suspend ready then it is residing in the "backing store" (Secondary memory)

→ Always a low priority processes will be suspended first.

processes are categorized into 2 types

i) CPU bound process
ii) I/O bound process

CPU Bound Process :-

→ The processes which requires more amount of CPU time are called as CPU bound process.

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

I/O Bound Process :-

→ The processes which requires more amount of I/O time are called as I/O bound processes.

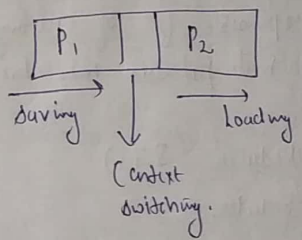
→ These processes will spend more time in the wait (or) block state.

Degree of Multi programming :-

The no. of processes present in the main memory at any point of time is called as degree of multiprogramming.

→ Each and every time when the process is moving from one state to another 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.

→ Context switching time is considered as overhead for the system.

NOTE: In some special cases if there is only 1 process still that is called as context switching.

Eg:- Round robin scheduling with 1 process.

In the operating system there are 3 different ~~Scheduling~~ schedulers

i) Long term scheduler (LTS)
(or) Job scheduler

ii) → It is responsible for of creating and bringing the new processes into the system

iii) Short term scheduler (STS)
(or) CPU Scheduler

→ STS is responsible of selecting one of the process in ready state for scheduling on to the CPU. (running state)

iii) Mid Term Scheduler (MTS)
(or) Medium term Scheduler

→ MTS is responsible of suspending and resuming the processes.

→ The job done by MTS is called as swapping.

Dispatcher :-

→ Dispatcher is responsible of loading the selected job onto the CPU. (this is called as dispatching)

→ Dispatcher is also responsible of performing context switching.

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

→ LTS controls degree of multiprogramming.

Gate
Q. Consider the system which has "n" CPU processors. Then what is the min & max. no. of processes that may present in the ready, running & block states?

Sol:

	min	max
Ready	0	"Any" no. of processes
Running	0	n
block	0	"Any"

n

Note at back →

NOTE

"Any" depends on max degree of multi programming in the system

→ Process is having various different times

i) Arrival Time (AT) :- (also called Submission time)

The time when the process is arrived into ready state is called as arrival time of a process

ii) Burst Time (BT) :-

The time required by the process CPU for its execution is called burst time of a process.

iii) Completion Time (CT) :-

The time when the process is completed its total execution is called as completion time of a process.

iv) Turn Around Time (TAT) :-

The time difference b/w completion time and arrival time is called as TAT of a process.

$$TAT = CT - AT$$

Waiting Time (W.T) :-

The time difference b/w TAT and BT, is called as WT of a process.

$$WT = TAT - BT$$

Response Time (R.T)

The time difference b/w first response and arrival time is called as R.T of a process.

~~RT~~

11/9/18

CPU Scheduling

The decision making of selecting the process in ready state for scheduling on to the CPU. is called as CPU scheduling

Who: STS (Short Term Scheduler)

Where: Ready State

When: 1. Process going from running to termination

1.2. " " " running to wait,

1.3. " " " ~~ready~~ to running to ready

2. Wait → ready [High priority process]
New → ready [High priority process]

Goal of CPU Scheduling

1. To maximize CPU Utilization & Throughput of the system.
2. To minimize the avg. TAT, avg WT & avg. RT of the processes.

1. First Come First Serve (F.C.F.S)

Criteria: Arrival Time
Mode: Non-preemptive

Q. Process id

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

find avg TAT = ?
& avg WT = ?

Sol. Gantt chart

P1	P2	P3	P4	P5
0	2	5	11	16	20

P.no	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	1	3	5	4	1
3	2	6	11	9	3
4	3	5	16	13	8
5	4	4	20	16	12

avg TAT = $\frac{2+4+9+13+16}{5} = \frac{44}{5} = 8.8$

avg WT = $\frac{0+1+3+8+12}{5} = \frac{24}{5} = 4.8$

Q.

P.no	AT	BT	CT	TAT	WT
1	6	6	31	25	19
2	2	5	8	6	1
3	4	6	22	18	12
4	1	2	3	2	0
5	5	3	25	20	17
6	3	8	16	13	5

Sol. CPU idle

	P4	P2	P6	P3	P5	P1
0	1	3	8	16	22	25	31

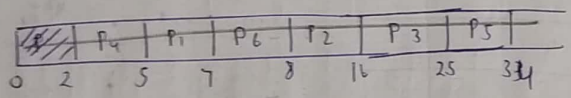
avg TAT = $\frac{25+6+18+2+20+13}{6} = \frac{84}{6} = 14$

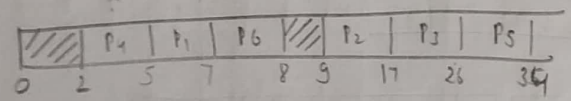
avg WT = $\frac{19+1+12+0+17+5}{6} = \frac{54}{6} = 9$

NOTE: if the arrival time of the processes are matching then schedule the process which has lowest process id.

Q.

P.NO	AT	BT	CT	TAT	WT
1	3	2	7	4	2
2	9	8	17	8	0
3	12	9	26	14	5
4	2	3	5	3	0
5	15	8	34	19	11
6	3	1	8	5	4





avg TAT = $\frac{53}{6}$ = 8.83

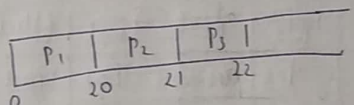
avg WT = $\frac{22}{6}$ = 3.66

Q1

P.NO	AT	BT
1	0	20
2	1	1
3	2	1

find Avg WT?

Sol:



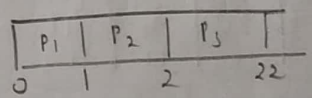
P.NO	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	1	21	20	19
3	2	1	22	20	19

Avg WT = $\frac{38}{3}$ = 12.67

Q2

P.NO	AT	WB	CT	TAT	WT
1	0	1	1	1	0
2	1	1	2	1	0
3	2	20	22	20	0

Sol:



Avg WT = 0

Convoy Effect

In the FCFS if the 1st process is having CPU bound process and followed by many IIO bound processes, then it will have ~~major~~ effect major effect on avg. WT of the processes. This effect is called as Convoy Effect

2. Shortest Job first (SJF)

Criteria :- Burst Time

Mode :- Non-preemptive

Q

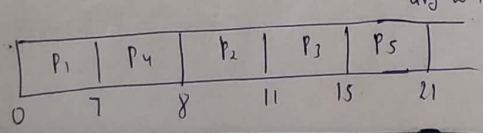
P.no	AT	BT	CT	TAT	WT
1	0	7	7	7	0
2	1	3	11	10	7
3	2	4	15	13	9
4	3	1	8	5	4
5	4	6	21	17	11

find avg TAT & avg WT

avg TAT = $\frac{52}{5}$ = 10.4

avg WT = $\frac{31}{5}$ = 6.2

Sol:



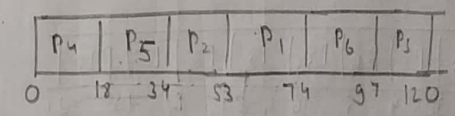
NOTE:

If the BT of the processes are matching then schedule the process which has lowest arrival Time.

Q

P.no	AT	BT		CT	TAT	WT
1	24	21	②	74	50	29
2	15	19	①	53	38	19
3	20	23	④	120	100	77
4	0	18	⌐	18	18	0
5	12	16	⌐	34	22	6
6	18	23	③	97	79	56

Sol:



avg TAT = $\frac{50+38+100+18+22+79}{6} = \frac{307}{6} = 51.16$

avg WT = $\frac{29+19+77+0+6+56}{6} = \frac{137}{6} = 31.16$

3 Shortest Job Remaining Time First (SRTF)

Criteria : Burst Time

Mode : preemtive

Q

P.NO	AT	BT	CT	TAT	WT
1	0	7	21	21	14
2	1	5	15	14	9
3	2	3	6	4	1
4	3	1	4	1	0
5	4	2	8	4	2
6	5	3	11	6	3

P1	P2	P3	P4	P3	P3	P5	P6	P2	P1
1	2	3	4	5	6	8	11	15	21

↑ all processes have arrive, it will work as SJF with non-preemptive

avg TAT = $\frac{50}{6}$ = 8.33

avg WT = $\frac{29}{6}$ = 4.83

Q

P.NO	AT	BT	CT	TAT	WT
1	6	3	9	3	0
2	2	5	13	11	6
3	4	1	5	1	0
4	0	6	19	19	11
5	3	6	25	22	16
6	2	4	4	2	0

Sol.

P4	P6	P6	P3	P2	P1	P2	P4	P5
0	1	2	3	4	5	6	9	13	19	25

avg TAT = $\frac{59}{6}$ = 9.66

avg WT = $\frac{33}{6}$ = 5.5

Q

P.NO	AT	BT	CT	TAT	WT
1	4	1	6	2	1
2	2	8	13	11	6
3	6	8	27	27	18
4	3	2	5	2	0
5	5	3	9	4	1
6	1	7	19	18	11

$\frac{64}{6}$ = 10.6 $\frac{27}{6}$ = 6.16

P3	P6	P2	P4	P4	P1	P5	P2	P6	P3
0	1	2	3	4	5	6	9	13	19	27

Q Grade 20'16 (2M)

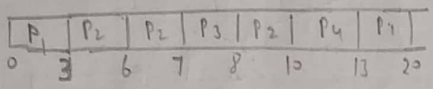
Q

P.No	AT	BT	CT	TAT
1	0	20	20	20
2	3	7	10	7
3	7	1	8	1
4	8	3	13	5

find avg TAT?

$\frac{33}{4} = 8.25$

Sol:

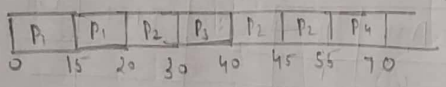


Q 25 P-81 (WB)

PNO	AT	BT	CT	TAT	WT
P1	0	20	20	20	0
P2	15	25	55	40	15
P3	30	10	40	10	0
P4	45	15	70	25	10

$\frac{25}{4} = 6.25$

Sol:

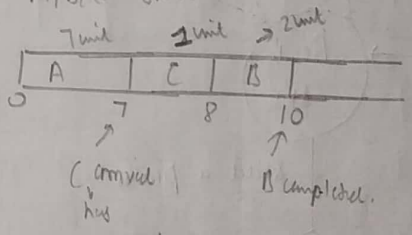


Dec 2

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

a) 11 and 3
b) 11 and 4
c) 12 and 3
d) 12 and 4

Sol:- A, B, C SRTF



Let C B.T be x_2 Let A B.T be x_1

After executing C for 1 unit we switched to B that means

$x_2 - 1 > 2 \Rightarrow x_2 > 3$

Hence min value of x_2 can be 4

Similarly, $x_1 - 7 > x_2$

$x_1 - 7 > 4$

$x_1 > 11$

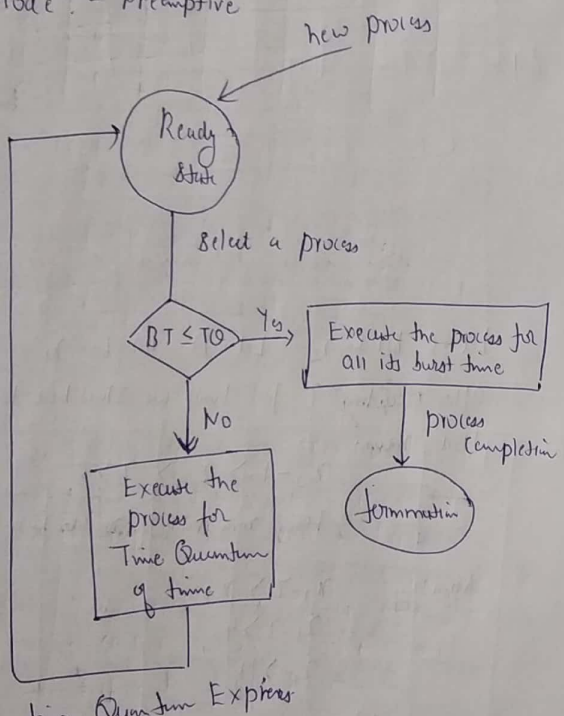
Hence min value of x_1 can be 12

Ans: d) 12 and 4.

4 Round Robin Scheduling

Criteria :- Time Quantum, Arrival Time (Time Slice)

Mode :- Preemptive



Q

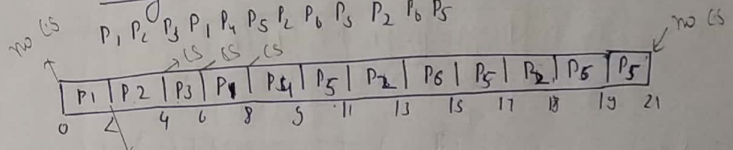
T.Q = 2

P.NO	A.T	B.T	C.T	TAT	WT	RT
1	0	4	8	8	4	0
2	1	5	18	17	12	1
3	2	2	6	4	2	2
4	3	1	9	6	5	5
5	4	6	21	17	11	5
6	6	3	19	13	10	7

$\frac{65}{6} = 10.83$ $\frac{44}{6} = 7.33$ $\frac{20}{6} = 3.33$

Ready Queue

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



Q TQ = 3

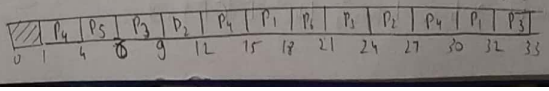
PNO	AT	BT	CT	TAT	WT	RT
P1	5	5/2/0	32	27	22	10
P2	4	6/3	27	23	17	5
P3	3	7/4/1	33	30	23	3
P4	1	9/6/3	30	29	20	0
P5	2	2	6	4	2	2
P6	6	3	21	15	12	12

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

avg TAT = $\frac{128}{6} = 21.33$

avg WT = $\frac{96}{6} = 16$

avg RT = $\frac{32}{6} = 5.33$



TQ = 2

P.no	AT	BT	CT	TAT	WT	RT
1	2	8	16	14	11	2
2	5	8	19	14	11	8
3	0	6	18	18	12	0
4	3	1	9	6	5	5
5	4	2	11	7	5	5
6	1	4	13	12	8	1

$\frac{71}{6} = 11.8$ $\frac{52}{6} = 8.6$ $\frac{21}{6} = 3.5$

P3 P6 P1 P3 P4 P5 P6 P2 P1 P3 P2

Sol.

P3	P6	P1	P3	P4	P5	P6	P2	P1	P3	P2

0 2 4 6 8 9 11 13 15 16 17 19

12/9/17

NOTE:

1) In round robin if TQ is less than no. of context switches will increase and response time will be less

2) In round robin if TQ is larger than no. of context switches will decrease and response time will be more

3) If the TQ is greater than all the burst time (ie very very large) then round robin behaves like FCFS.

Q. Consider a system which has 4 processes P_1, P_2, P_3, P_4 all arriving in ready queue in the same order at the T=0. The burst time requirement of these processes 4, 1, 8, 1 respectively. Then what is the completion time of process P_1 assuming Round robin scheduling with TQ = 1

P1 4 3 2 1
P2 1 0
P3 8 7 6 5
P4 1 0

Sol. P1 P2 P3 P4 P1 P3 P1 P3 P1

P1	P2	P3	P4	P1	P3	P1	P3	P1

0 1 2 3 4 5 6 7 8 9

Completion time of P_1 is 9

Q51 P-42 (WB)

Sol.

	AT	BT
P1	0	5
P2	1	1
P3	3	4

FCFS: P1 P2 P3

R.R: P1 P3 P2

FCFS

P1	P2	P3

0 5 12 16

TQ=2 Round Robin P1 P2 P1 P3 P2 P1 P3 P2 P3

P1	P2	P1	P3	P2	P1	P3	P2	P3

0 2 3 4 6 8 10 11 13 15

NOTE: In the SJF scheduling algorithm, the B.T's of the processes are expected by using the following formula

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

τ_{n+1} = next expected B.T

τ_n = previous expected B.T

T_n = previous actual B.T.

α = parameter which controls relative weight of recent and past history

$$0 \le \alpha \le 1$$

Q 32 & 33 P-82

Sol.32)

P.NO	B.T
P_1	5
P_2	8
P_3	3
P_4	5

$\tau_1 = 10 \quad \alpha = 0.5$

$\tau_2 = (0.5)(5) + (0.5)10$
$= 2.5 + 5$
$= 7.5$

$\tau_3 = (0.5)(8) + (0.5)7.5$
$= 4 + 3.75$
$= 7.75$

$\tau_4 = (0.5)3 + (0.5)(7.75)$
$= 1.5 + 3.875$
$= 5.375$

$\tau_5 = (0.5)5 + (0.5)(5.375)$
$= 2.5 + 2.6875$
$= 5.1875$

Q 32
Sol. $\tau_5 = \alpha T_n + (1-\alpha)\tau_{n-1}$
$= 1\, T_n + 0$
$\tau_5 = 5$ (i.e BT of T_4)
$\tau_2 = 1 \cdot T_2 = 5$

Imp
→ SJF requires to know the running times of all the processes in advance, which is practically not possible. Hence SJF can be practically implemented using predicted BT

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

↓
predicted BT of $n+1^{th}$ process

Q Consider a "n" processes sharing the CPU in round robin algorithm. The context switching time is "s" units, then what must be the time quantum "q" such that, the no of context ~~switching of for every "t" seconds of~~ 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) $q = \frac{t - ns}{n+1}$ b) $q = \frac{t + ns}{n-1}$

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

Sol: n=4 P1 | P2 | P3 | P4 | P1

s s s s

t

BT =
CS = S
TQ = q

$(n-1)s + nq \leq t$ ✗

$ns - s + nq \leq t$

$t \geq t + s$

$ns - s + nq \not\leq t$

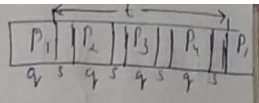
$q = t - ns$

$\left(\frac{t - ns}{n-1}\right)$ ✗

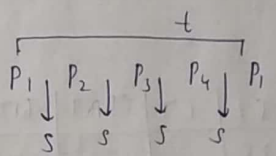
$\frac{t - nq}{n-1} = s$

$(n-1)q + ns \leq t$

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



n=4



4. Longest Job First (LJF)

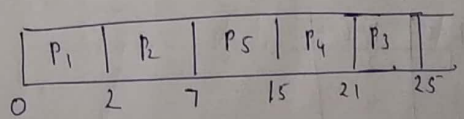
Criteria :- Burst time
mode :- Non-preemptive

Ex

P.no	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	1	5	7	6	1
3	2	4	25	23	19
4	3	6	21	18	12
5	4	8	15	11	3

avg TAT = $\frac{60}{5} = 12$

avg WT = $\frac{35}{5} = 7$



NOTE: if the BT of the processes are matching then schedule the process which has lowest AT.

Q

P.no	AT	BT	CT	TAT	WT
1	21	25	69	48	23
2	12	15	105	93	78
3	17	21	90	73	52
4	0	19	19	19	0
5	15	12	117	102	90
6	19	25	44	25	0

Sol:

P4	P6	P1	P3	P2	P5
0	19	44	69	90	105	117

Avg TAT = $\frac{360}{6} = 60$

Avg WT = $\frac{243}{6} = 40.5$

6 Longest ~~Job~~ ~~First~~ Remaining Time First (LRTF)

Criteria: Burst Time

Mode: Preemptive

Gate 2006

P.no	AT	BT
1	1	2
2	2	4
3	3	6
4	4	8

Sol:

P.no	AT	BT	CT	TAT	WT
1	1	2	18	17	15
2	2	4	19	17	13
3	3	6	20	17	11
4	4	8	21	17	9

17 = $\frac{48}{4} = 12$

P1	P2	P3	P4	P3	P4	P1	P3	P2	P3	P1	P2	P3	P4
0	1	2	3	4	7	8	9	10	11	12	13	14	15	16	17

P1	P2	P3	P4
18	19	20	21

avg TAT = 17

avg WT = 12

Gate 2006

Q

P.no	AT	BT	CT	TAT
1	0	12	12	12
2	0	8	13	13
3	0	4	14	14

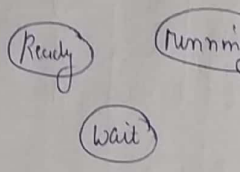
What is avg TAT using LRTF?

Avg TAT = 13

Sol:

P3	P2	P3	P2	P3	P1	P2	P3	P1	P2	P1
0	4	5	6	7	8	9	10	11	12	13	14

CPU Time, I/O Time Concept :-



Life cycle	Scenarios
CPU time	1) CPU ✓
I/O time	2) I/O ✓
CPU time	3) CPU, I/O, I/O ✓
⋮	4) I/O, I/O, I/O ✓
Complete	5) CPU, CPU, I/O ✗

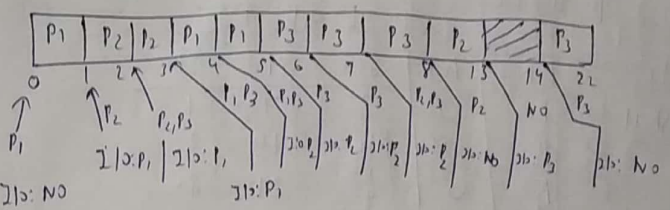
Q

P.NO	AT	Execution Time		
		CPU time	I/O time	CPU time
1	0	10	20	20
2	1	20	40	80
3	2	30	60	80

What is the completion times of processes P1 P2 P3 using SRTF algorithm ?

NOTE: 1) The 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.



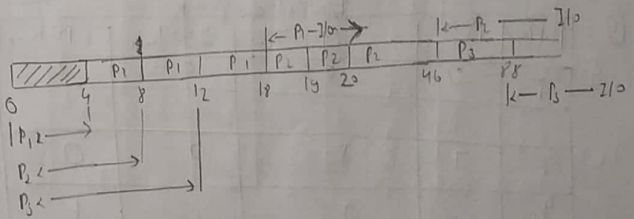
P.NO	CT
1	5
2	13
3	22

Q

P.NO	AT	Execution Time		
		I/O time	CPU time	I/O time
1	0	4	4 6	2
2	0	8	28 26	4
3	0	12	42	6

SRTF

Sol.



P NO	CT
1	18 20
2	46 50
3	88 94

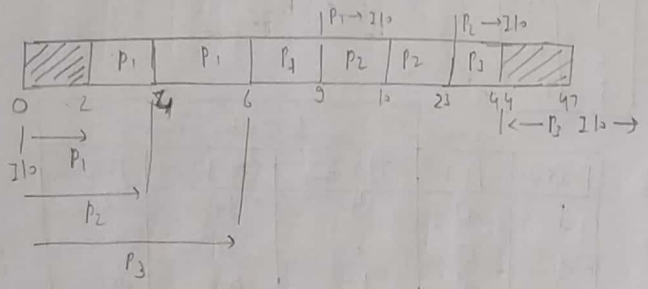
Q28 P-81 (WD)

Sol:

SRTF

	P.NO	AT	I/O	CPU	I/O
10	P_1	0	2	3	1
20	P_2	0	4	14	2
30	P_3	0	6	21	3

Execution Time



% CPU idle = $\frac{5}{47} \times 100 = 10.6$

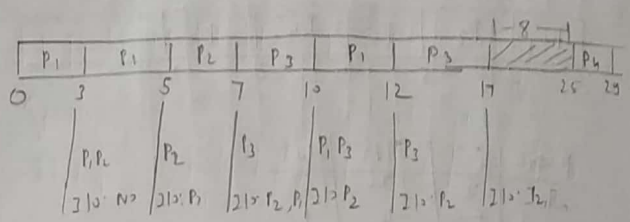
Q

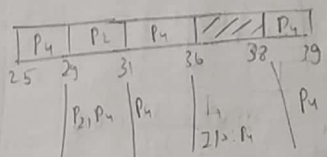
Execution Time

P.NO	AT	CPU Time	I/O time	CPU time
1	0	8	5	2
2	3	2	2	2
3	7	8	0	0
4	25	5	2	1

using SRTF

Sol:





Ans

P.NO	CT
P_1	12
P_2	31
P_3	17
P_4	39

7) Priority Based Scheduling

Criteria : Priority

Mode : Non-preemptive

Priority	P.NO	AT	BT
2 [L]	1	0	2
4	2	1	3
3	3	2	6
6	4	3	8
8 [H]	5	4	9

8 - highest
2 - lowest

P1	P2	P5	P4	P3
0	2	5	14	22	28

P.NO	CT	TAT	WT
1	2	2	0
2	5	4	1
3	28	26	20
4	22	19	11
5	14	10	1

avg TAT = $\frac{61}{5} = 12.2$

avg WT = $\frac{33}{5} = 6.6$

NOTE: if the priorities of the processes are matching then schedule the process which has lowest AT.

Priority	P.NO	AT	BT	CT	TAT	WT
4	1	4	6	14	15	9
[H] 7	2	6	3	9	3	0
6	3	3	4	13	10	6
6	4	2	2	6	4	2
[L] 1	5	1	3	4	3	0
3	6	2	2	21	19	17

Sol:

P5	P4	P2	P3	P1	P6
0	1	4	6	9	13	19	21

Avg TAT = $\frac{54}{6} = 9$

Avg WT = $\frac{34}{6} = 5.6$

Priority Based Scheduling

Criteria : priority

mode : preemptive

Priority	P.NO	AT	BT	CT	TAT	WT
Low 2	1	1	32	22	21	18
3	2	2	4	15	13	9
3	3	2	5	20	18	13
5	4	3	2	12	9	7
8	5	4	1	5	1	0
high 9	6	5	6	11	6	0

avg TAT = $\frac{68}{6} = 11.33$

avg WT = $\frac{47}{6} = 7.83$

Sol:

P1	P2	P4	P5	P6	P4	P2	P3	P1
0	1	2	3	4	5	11	12	15	20	21

Priority	P.NO	AT	BT	CT	TAT	WT
[L] 2	1	1	8	47	46	38
3	2	2	16	32	30	25
5	3	3	18	26	23	15
2	4	0	37	39	39	30
6	5	4	6	19	15	9
[H] 7	6	5	9	14	9	0

TAT = $\frac{162}{6} = 27$ WT = $\frac{115}{6} = 19.16$

Sol:

P4	P3	P2	P3	P5	P6	P5	P3	P2	P4	P1
0	1	2	3	4	5	14	19	26	32	39	47

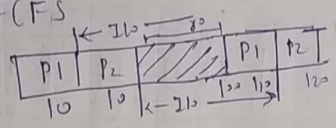
Imp

Q29 P-81 (WB)

Sol:

P.No	AT	CPU	IO	CPU	IO
P1	0	10	90	10	90
P2	0	10	90	10	90

a). FCFS



idle = 80

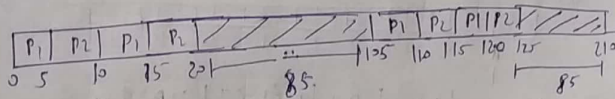
Idle = $\frac{80}{120} \times 100 = 75\%$

b) SRTF → same as FCFS

c) R.R TQ = 5ms

P1 burst = 20
P2 burst = 0

P1 P2 P1 P2



Idle = 85

c) Same as FCFS

13/9/18

Highest Response Ratio Next (HRRN)

Criteria: Response ratio
Mode: Non-preemptive
Response ratio = $\frac{W+S}{S}$

How we consider WT w.r.t to current Time.

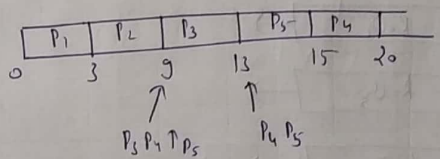
W: waiting time
S: Service time (or) Burst time.

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

Q

P.NO	AT	BT	CT	TAT	WT
1	0	3	3	3	0
2	2	6	9	7	1
3	4	4	13	9	5
4	6	5	20	14	9
5	8	2	15	7	5

avg TAT = 8 ← 40/5
avg WT = 4 ← 20/5



P3 P4 P5

P4 P5

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

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

P_5 R.R = $\frac{1+2}{2} = 1.5$

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

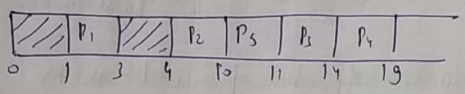
$P_5 = \frac{5+2}{2} = 3.5$

Q

P.No	AT	BT	CT	TAT	WT
1	1	2	3	2	0
2	4	6	10	6	0
3	5	3	14	9	6
4	6	5	19	13	8
5	8	1	11	3	2

avg TAT = $\frac{33}{5}$ = 6.6

avg WT = $\frac{16}{5}$ = 3.2



I) t=10

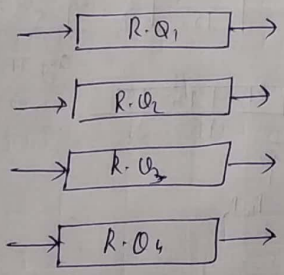
$P_3 = \frac{5+3}{3} = 2.67$ $P_4 = \frac{4+5}{5} = 1.8$ $P_5 = \frac{2+1}{1} = 3$

II) t=11

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

Multilevel Queue Scheduling

R.Q → ready Queue



→ Depending on the priority of the process in which particular ready queue the process has to be placed will be decided.

→ The high priority process will be placed in the top level ready queue and low priority process will be placed in the bottom level R.Q

→ Only after completion of all the processes from the top level ready queue further level R.Q 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 the process is called as starvation

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

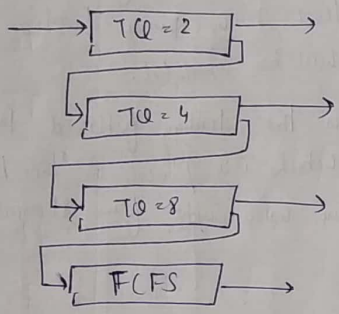
Ageing

If WT of a process increased then priority of the process will be increased.

If the age of a process ~~is increased~~ increases, by increasing the priority the process will definitely get a chance to

to execute and starvation problem will be avoided.

Multi level Feedback Queue Scheduling



In the above algorithm every process will definitely get a chance to execute for some amount of time and still there is a possibility of starvation

Q8 P-Ty(WB

TQ = BT = 40
38
34
19
2

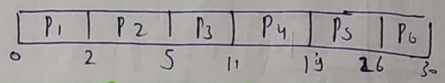
Level 5 queue process will execute completely.

Q

P.NO	AT	BT	CT
1	0	2	2
2	1	3	5
3	2	6	11
4	3	8	19
5	4	7	26
6	5	4	30

What is Throughput of the system using FCFS?

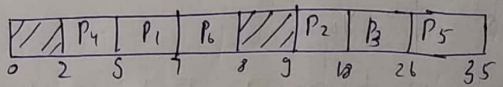
Sol:



$$\text{Throughput} = \frac{\#\text{ of processes}}{\max(CT) - \min(AT)} = \frac{6}{30-0} = \frac{6}{30} = 0.2 = 20\%$$

Q

P.NO	AT	BT
1	3	2
2	9	9
3	12	8
4	2	3
5	15	9
6	3	1



$$\text{Throughput} = \frac{6}{35-2} = \frac{6}{33} = 0.18 = 18\%$$

Schedule length = Max(CT) - Min(AT)

	Algorithm	Starvation
1.	FCFS	No
2.	NP-SJF	Yes
3.	PSRTF	Yes
4.	R.R	No
5.	NP-LJF	Yes
6.	LRTF	No
7.	NP-Priority	Yes
8.	Pre-Priority	Yes
9.	HRRN	No
10.	MLQ	Yes
11.	MLFQ	Yes

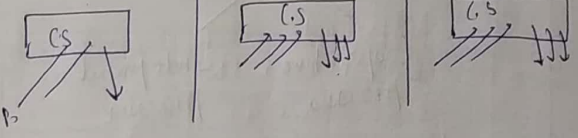
NOTE: Majority of the operating systems practically implement ~~RR~~ a Round Robin Scheduling

EX: Windows, Linux, Unix etc

Synchronization checking Technique (last class)

1) checking progress

Strict Alternation	Peterson's algo	TSL inst
Process P_0 P_1	Process P_0 P_1	Process P_0 P_1 P_2 ...
Progress x	Progress ↓	Progress ↓



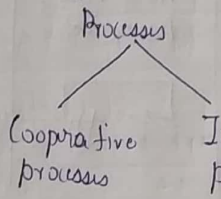
Let P_0 enter into C.S, When it comes out, again make P_0 go into C.S without giving chance to P_1

Note: We can apply the above technique only after checking for deadlock.

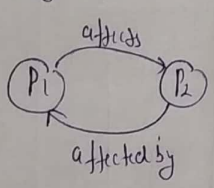
Synchronization

Text book - Modern O.S by Tennabum

processes are categorized into 2 types:-



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



Cooperative

$P_1 \;||\; P_2$

Independent

Understanding Synchronization

1. The problem arises by not having synchronization blw the processes
2. Conditions to be followed to achieve synchronization
3. The solutions (wrong and write)

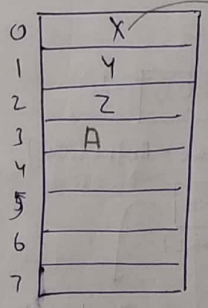
Problems Arises by NOT Having Synchronisation Between the processes.

① Consumer Producer Problem

N = 8

count = 3 2 4

Producer in 3 4



(consumed) Consumer

out 0 1

here we have 3 items but count is 4. Here we have problem of inconsistency

Buffer [0..... N-1]

```
Void count = 0
void producer (void)
{
    int itemp;
    While (true)
    {
        produce_item (itemp)
        While (count == N);
        Buffer [in] = itemp
        in = (in + 1) mod N
        count = count + 1;
    }
}
```

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

```
Void Consumer (void)
{
    int itemc
    While (true)
    {
        While (count == 0);
        itemc = Buffer [out]
        out = (out+1) mod N
        count = count - 1
        process_item (itemc)
    }
}
```

I. load R_c, m [count]
II. DEC R_c
III. Store m[count], R_c

"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 variable used by both producer and consumer to identify no. of items present in the buffer at any point of time.

Shared Resource

1. count variable
2. Buffer

Conditions to be followed

1. When the buffer is full, producer is not allowed to produce the item into the buffer.
2. When the buffer is empty, consumer is not allowed to consume the item from the buffer.

Analysis

Universal Assumption

The running process can get preempted at any point of time after completion of the current instruction.

Itemp = A

Itemc = X

P - I Rp [3]

P - II → preempt

C → I Rc [3]

C - II

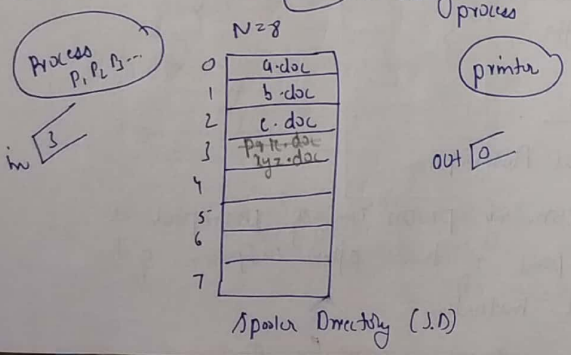
C - III [X 2]

P - III [X 4]

Inconsistency Problem

The producer & consumer are not properly synchronized, while sharing common variable count. Hence it is leading to the problem of inconsistency.

Printer Spooler Daemon → background process



Enter - File → respective process register

I. load Ri, m[in] → file name

II store SD[Ri], "F-N"

III INC Ri

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

Shared resources

1) in variable
2) Spooler directory

Analysis

R.Q [P1 | P2]

pqr.doc xyz.doc

Here we have data loss problem.
File given by P1 is overwritten by File of P2.

P1 → I
P1 → II
P1 → III → preempt
P2 → I
P2 → II
P2 → III
P4 → IV

R1 [3]
R2 [3]
[X 4] 3+1

Loss of Data Problem

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

Deadlock

If The processes are not properly synchronized while sharing the common resources (or) common variables then it is also possible for deadlock to occur.

Definitions

1. Critical Section (CS)

The portion of prg. text where shared variables (or) shared resources will be placed.

eg: `count = count + 1 ;`

↑ in Producer code

2. Non Critical Section

The portion of prg. text where the independent code of the processes will be placed.

3. Race Condition

The final value of any variable depends on the execution sequence of processes. This type of condition is called as race condition

Count = 4

P - I
P - II
——— → preempt
C - I
C - II
C - III
———
P - III

Count = 2

C - I
C - II
——— → preempt
P - I
P - II
P - III
———
C - III

NOTE: To avoid the race condition only one process is allowed to enter into critical section.

The Conditions to be followed to achieve Synchronization

1. Mutual Exclusion
   - → No two processes may be simultaneously present inside the critical section at any point of time.
   - → Only one process is allowed to enter 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 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 to enter into critical section
   - → If 2 or more processes are trying to enter into critical section then one process should be definitely allowed to enter 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 process. This situation should not arise.
   - → If bounded waiting is not satisfied then it is possible for starvation.

4. No assumptions related to H/W and processor speed. (ie architectural neutral)

Solutions

I. Software Type
   a) Lock Variables
   b) Strict alteration (or) Dekkers algorithm
   c) Petersons algorithm

II Hardware Type
   a) TSL Instruction set.
      ↳ Test and set lock

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

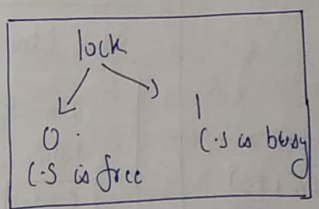
IV Programming language Compiler Supported type
   a) Monitors

I Software Type Solutions

a) Lock Variables

Entry Sections :- (respective process registers)

I. Load Ri, 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

1. ME not satisfied

$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III
$P_1 \rightarrow$ preempt
$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III
$P_2 \rightarrow$ IV
$P_2 \rightarrow$ preempt
$P_1 \rightarrow$ IV
$P_1 \rightarrow$ V ← Hence both the processes are inside C.S

after preemption when process is resumed, it starts its exec from next instr i.e $P_1 \rightarrow$ IV

We have proved that both the processes P_1 & P_2 are entering into C.S at the same time. Hence Mutual Exclusion is not satisfied and the solution is bound to be incorrect.

b) Strict Alteration (or) Deckar's Algorithm
(process take "turn" to enter into C.S)

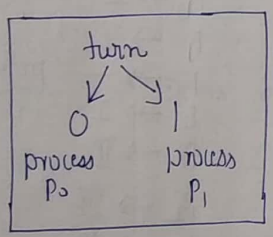
This solution is valid only for 2 process.

Process "P_0" code

```
While (True)
{
    non_cs ();
    While (turn != 0);
    [C.S]
    turn = 1;
}
```

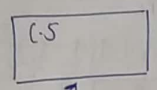
Process "P_1" code

```
While (true)
{
    non_cs()
    While (turn != 1);
    [C.S]
    turn = 0;
}
```



Analysis

Turn = 0 or 0



P_1

although C.S is empty but P_1 can't enter bcz P_0 is not interested to enter into CS and it is blocking P_1 also.

1. ME is satisfied
2. Progress is not satisfied

We have proved that progress is not satisfied and the solution is bound to be incorrect.

Important Points

1. The preemption is just a temporary stop, the process will come back and continue the remaining execution.
2. If there is any possibility of solution becoming wrong by taking the preemption then consider the preemption.

Gate 3. If any solution is having deadlock then progress is not satisfied

~~CBS P-30(b)~~

Q15 P-90 (WB)

Sol:- $S_1 = 1$ $S_2 = 0$

P_1	P_2
1. While ($S_1 == S_2$);	1. While ($S_1 != S_2$);
CS	C.S
2. $S_1 = S_2$;	2. $S_2 = not(S_1)$;

I - case

$P_1 \to 1$, $P_2 \to 1$ | $P_1 \to 2$, $P_2 \to 2$

1. ME is satisfied

II - case

$P_1 \to 1$, $P_1 \to 1$ ↻ trap in While

$P_1 \to 2$

Since even though C.S is empty but P_1 can not Enter into C.S. It has to wait for P_2. but P_2 is not interest in going into CS.

2. No progress

Ans: option a)

c) Peterson Solution

It is a two 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);
}
```

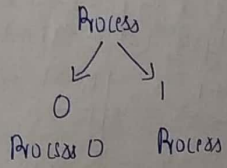
C.S

```
void Leave_region (int process)
{
  6. interested [process] = FALSE
}
```

initially

interested [0] = FALSE

interested [1] = FALSE



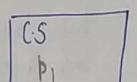
"Turn" is a shared variable used by both the processes P_0 and P_1

Analysis ①

initially

interested [0] = ~~FALSE~~ TRUE
interested [1] = FALSE TRUE

Turn = 0 / 1



busy waiting in while loop

Process P_0	Process P_1
Other = 1	Other = 0

Analysis ②

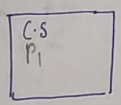
initially

interested [0] = ~~FALSE~~ TRUE
interested [1] = ~~FALSE~~ TRUE

Turn = 1 0

trap in while loop

Process 0	Process 1
~~turn~~ other = 1	other = 0



Hence ME is satisfied.

Analysis ③

at any point of time only 1 process present inside C.S

$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$
$P_0 \to 5$ → False
CS

$P_1 \to 1$
$P_1 \to 2$
$P_1 \to 3$
$P_1 \to 4$
$P_1 \to 5$ ↻ trap

II

$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$
$P_0 \to 5$ - False
$P_0 \to 6$

2. P_0 try again bcz CS is Empty

$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$
$P_0 \to 5$ - False
CS

1. Now P_1 is not interested to go in CS

Hence Progress is satisfied.

Analysis ④

$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$
$P_0 \to 5$
$P_0 \to$ CS → preempt

$P_1 \to 1$
$P_1 \to 2$
$P_1 \to 3$
$P_1 \to 4$
$P_1 \to 5$ ↻ trap in while loop

continue

$P_0 \to 6$

→ Now P_0 try to enter into CS again

$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$
$P_0 \to 5$ ↻ trap in while loop

We can clearly see that, only after P_1 goes to C.S, then only P_0 can go to C.S.

Hence bounded waiting is satisfied.

We have proved all the 3 conditions are satisfied in the above solution and the solution is bound to be correct.

To check
i) ME → if P_0 enter into CS, now try to make P_1 enter into CS.
ii) Progress → make both the process enter into CS simultaneously.
iii) Bounded Wait → P_0 enter into CS, P_1 try to enter, P_0 again try to enter into CS continuously.

Q. Assume that both the processes P_0 & P_1 are trying to enter into critical section at same time, then which process will enter into C.S ?

a) The process which executes stmnt 2 first

b) " " " stmnt 3 first

c) " " " " 4 first

d) We can't say anything.

Which ever process sets turn variable first enters into C.S. first

Sol:
1. int other
2. other = ~~process~~ 1 - process
3. interested [process] = TRUE $\leftarrow P_0, P_1$
4. turn = process $\leftarrow P_1, P_0$
5. while (turn == process && interested [other] == TRUE)

$P_0 \to 1$ interest: False | False
$P_0 \to 2$ True True
$P_0 \to 3$
→ preempt
$P_1 \to 1$
$P_1 \to 2$
$P_1 \to 3$
$P_1 \to 4$ turn = 1
$P_1 \to 5$ ↻
→
$P_0 \to 4$ turn = X0
$P_0 \to 5$ ↻
→
$P_1 \to 5$
C.S

$P_1 \to 1$ interest: False | False
$P_1 \to 2$ turn turn
$P_1 \to 3$
→
$P_0 \to 1$
$P_0 \to 2$
$P_0 \to 3$
$P_0 \to 4$ turn = 0
$P_0 \to 5$ ↻
→
$P_1 \to 4$ turn = 01
$P_1 \to 5$ ↻
→
$P_0 \to 5$
C.S

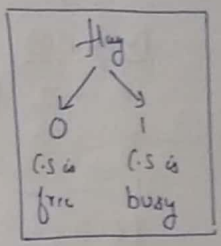
a) TSL (Test and Set Lock) ← H/w Type Solution

TSL register flag

Copies the current value of flag into register and stores the value of '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 ① R_i flag = 01
$P_1 \to$ I 0
$P_1 \to$ I
$P_1 \to$ III
C.S
→ preempt R_2
$P_2 \to$ I 1 flag = 11
$P_2 \to$ II
$P_2 \to$ III ↻

Hence ME satisfied

Analysis ②
$P_2 \to$ I 0 flag = 0 1
$P_2 \to$ II
$P_3 \to$ III
C.S
→ preempt
$P_1 \to$ I 1 flag = 11
$P_1 \to$ II
$P_1 \to$ III ↻

Analysis ③

Consider → $P_1 \rightarrow$ I [0] flag = ∅1

$P_1 \rightarrow$ II

→ preempt

$P_2 \rightarrow$ I [1] flag = X1

$P_2 \rightarrow$ II

$P_2 \rightarrow$ III ↺

→ P_1 resumes

$P_1 \rightarrow$ III

[C.S]

Hence progress is guaranteed

Analysis ④

$P_1 \rightarrow$ I [0] flag = ∅1

$P_1 \rightarrow$ II

$P_1 \rightarrow$ III

[C.S]

→ preempt

$P_2 \rightarrow$ I [1] flag = X1

$P_2 \rightarrow$ II

$P_2 \rightarrow$ III ↺

→ P_1 resume

$P_1 \rightarrow$ flag = X0

Now P_3 try to Enter into C.S again

$P_3 \rightarrow$ I [0] flag = ∅1

$P_3 \rightarrow$ II

→ preempt

$P_2 \rightarrow$ I [1] flag = X1

$P_2 \rightarrow$ II

$P_2 \rightarrow$ III ↺

→ P_3 resumes

$P_3 \rightarrow$ III

[C.S]

Hence we can see that Bounded wait is not satisfied

P_2 is unlucky process which can't enter into C.S bcz C.S is always busy by other processes hence it will starve.

Even though if we use Round Robin algorithm bounded wait is not satisfied.

Ready Queue P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_7 P_3

P_1 starts exn, enters into C.S but got preempted but it could release C.S. P_2 & P_3 have to wait bcz C.S is not released. Hence their TQ expires and they try again by the time P_2 TQ expires P_4 & P_5 arrives. Hence when P_1 again gets chance it execute completely and releases C.S, and now P_4 enters into C.S and not P_2.

NOTE: i) If some process is in the critical section then all other processes which are trying to enter into C.S will be repeatedly checking for I, II, III instr's.

ii) These processes are busy in checking those instrs and waiting to enter into C.S. This is called Busy waiting

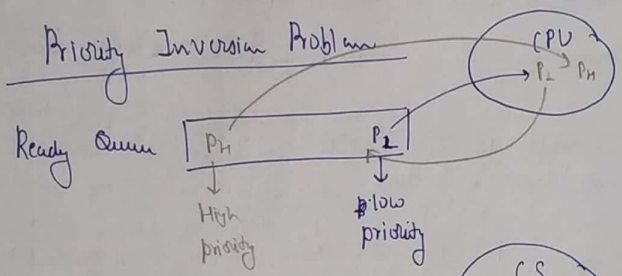
iii) With the busy waiting they are wasting the CPU cycles (time). Busy waiting is also called Spin lock

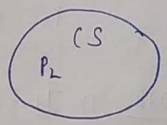
iv) To avoid busy waiting problem Concept of Semaphore will be used

Solutions	M.E	Progress	bounded waiting
1) Lock variable	X	X	X
2) Strict alternation (or) Dekker's algorithm	✓	X	X
3) Peterson's algorithm	✓	✓	✓
4) TSL instr register flag	✓	✓	X

14/9/18

Priority Inversion Problem





Live Lock

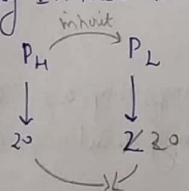
$P_L \rightarrow$ ready

$P_H \rightarrow$ Run

Low priority executing in CS, when P_H comes, P_L gets preempted by it could release CS. Now P_H is trying to get into CS, but it can't bcz CS is not released by P_L (i.e P_L didn't complete its execn and got preempted)

Solution :-

priority Inheritance



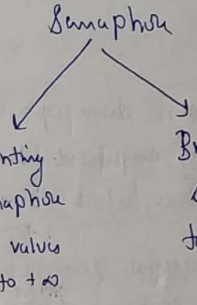
both priority match, hence arrive time is seen, and P_L continue in CS.

Operating System Type of Soln

1) Semaphore

Semaphore is an integer variable used by various processes in a mutual exclusive manner to achieve synchronization ie one by one

The improper usage of semaphore will also give the wrong results.



The 2 different Opn are performed on the semaphore variable

1) Down() ; ~~also called~~ 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~~ suspended list ();
    }
}
```

```
Up( Semaphore S)
{
    S.value = S.value + 1
    if (S.value <= 0)
    {
        // Select a process from
           suspended list and wakeup();
    }
}
```

NOTE: i) After performing down opn if the process is getting suspended then it is called as unsuccessful down opn.

ii) If it is unsuccessful down opn then the process will not continue further exec in the code.

iii) After performing down opn if the process is not getting suspended then it is called as successful down opn.

iv) If it is successful down opn then the only process will continue further exec in the code

v) +ve value of the counting semaphore indicates no. of successful down opn we can perform

vi) down opn on the counting semaphore will be successful only if the initial value of semaphore is $S \geq +1$

vii) There is no unsuccessful "Up" opn, And "Up" opn is always successful.

viii) The process performing "Up" opn will definitely continue further exec in the code

Q Consider a system where the initial value of the counting semaphore $S = +13$. The various semaphore opn's like

12 P, 15 V, 7 P, 8 V, 2 P, 5 V

are performed. Then what is the final value of counting semaphore.

Sol: 20

S = 13 → 1 → 16 → 8 → 17 → 15 → 20

12P 15V 7P

Eg: $S = +N$

NOTE: The -ve value of counting semaphore indicates no. of processes blocked (ie in the suspended list)

Eg:

S = +x ∅ -x -2

P_1	P_2	P_3
P	P	P
V	X	X

Gate 2016 (2M)

① Consider a system which has counting semaphore variable S. The various semaphore op's like

20 P, 12 V, are performed.

Then what is the ~~initial~~ largest initial value of semaphore S. so that 1 process will remain in the blocked state.

Sol:

$$\begin{bmatrix} x + 20 - 12 = -1 \\ x = 11 - 20 \\ x = -9 \end{bmatrix} \times \qquad \begin{aligned} x - 20 + 12 &= -1 \\ x &= 19 - 12 \\ x &= +7 \end{aligned}$$

Let x be initial semaphore value.

Binary Semaphore

```
Down ( Semaphore S)
{
    if (S.value == 1)
    {
        S.value = 0;
    }
    else
    {
        block the process and place
        its PCB in the suspend list()
    }
}
```

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

NOTE: ① After performing down op^n, if the process is getting suspended ~~then~~ then it is called as unsuccessful down op^n

ii) If it is unsuccessful down opn then process will not continue further exen in the code.

iii) After performing down opn if the process is not getting suspended then it is called as successful down opn.

iv) If it is successful down opn then only process will continue further exen in the code.

v) Down opn on a binary semaphore will be successful only if the initial value of semaphore S = +1

vi) There is no unsuccessful "Up" opn and "up" opn is always successful.

vii) The process performing "Up" opn will definitely continue further exen in the code.

S = 1, 0, 0, 1, 1

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

S.L P_2 P_3

Imp Q22 P-91 (WB)

Sol. S = 1

9P, 14V, 6P, 8V, 3P, 2V

= 0

S = 1 0 ~~1~~ 0

~~1~~ ~~0~~ 2

Important Points :-

1. Every semaphore (variable) (value) will have its own suspended list.
2. The down and up opn are atomic.
3. If more than 1 process is in the suspended list then every time when we perform 1 up opn, 1 process will wakeup is from ~~from~~ suspended ~~list~~ list and that is based on First In First Out.
4. If 2 or more processes are in the suspended list and there is no other process to wake up these processes then those processes are said to be involved in the deadlock.

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

```
While(true)
{
  1 P(mutex);
    CS
  2 V(mutex);
}
```

The process "P_{10}" executes the following code

```
While (TRUE)
{
  1 V(mutex) ;
  2 C.S
  3 V(mutex);
}
```

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

b) 3 — II Case — V
c) 1 — III Case — P

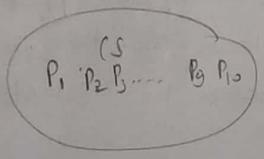
P V

What is the maximum no. of processes that may 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

Sol:



P_1-1 | $P_{10}-1$ | P_2-1 | $P_{10}-2$ / $P_{10}-3$ | P_3-1 | $P_{10}-1$ | P_4-1 | $P_{10}-2$ / $P_{10}-3$ | ....

P

follow similar pattern. P_{10} brings 2 process at a time inside CS.

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

Process "P" code

```
While ( True)
{
  W: _____
  print ('0');
  print ('0');
  X: _____
}
```

Process "Q" code

```
While( True)
{
  y: _____
  print ('1');
  print ('1');
  Z: _____
}
```

What should be the binary semaphore operations on W, X, y, z and what must be initial values of binary semaphores "S" and "T" inorder to get print O/p always 0011001100.....

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(S), Z = V(T), S = 1 T = 1;

c) W = P(S), X = V(T), y = P(T), Z = V(T), S = 1, T = 1;

c) W = P(T), X = V(T), y = P(S), Z = V(S), S = 1, T = 0

~~d)~~ W = P(S), X = V(T), y = P(T), Z = V(S) S = 1, T = 0

Sol: c) S = 1 T = 0

W : P(T) y = P(S)

X : V(T) Z = V(S)

d) S = 1 T = 0 → here initial T value is 0 Hence o/p will never start from "1100110...". & here we have strict alternation

W = P(S) y = P(T)

X = V(T) Z = V(S)

00

Q. Which of the following will be ensure that, the O/p string will never contain the 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 = P(T), X = V(S) y = P(S), Z = V(T), S = T = 1

Sol: ~~x~~ d) W = P(T) y = P(S) S = T = 1

X = V(S) Z = V(T)

010 X

(d) W = P(S) y = P(S) S = 1

X = V(S) Z = V(S)

I → P 00110011 or 11001100 or 0000... or 1111...

Q 17 P - So (WB)

Sol: $S_X = S_Y = 1$ deadlock - When both process are suspended none can proceed..

d) P(Sx) P(Sx)

P(Sy) P(Sy)

When V(Sx) → the suspended list of Sx is not empty, hence it will wake P_2 up. Now P_2 will start from L_4: P(Sy) i.e next line next start

Q 16 P - So (WB)

Sol: I - case initially $S_0 = 1$ $S_1 = 0$ $S_2 = 0$

$P_0 \to 1$	$P_1 \to 1$	$P_0 \to 1$	$P_2 \to 1$	$P_0 \to 1$
$P_3 \to 2$	$P_1 \to 2$	$P_0 \to 2$	$P_2 \to 2$	$P_0 \to 2$
$P_3 \to 3$		$P_3 \to 3$		$P_3 \to 3$
$P_3 \to 4$		$P_3 \to 4$		$P_3 \to 4$
0		0		0

$P_0 P_1 P_3 P_2 P_3$

II case initially $S_0 = 1$ $S_1 = 0$ $S_2 = 0$

$P_0 \to 1$	$P_1 \to 1$	$P_2 \to 1$	$P_3 \to 1$
$P_0 \to 2$	$P_1 \to 2$	$P_2 \to 2$	$P_3 \to 2$
$P_3 \to 3$			$P_3 \to 3$
$P_3 \to 4$			$P_3 \to 4$
0			0

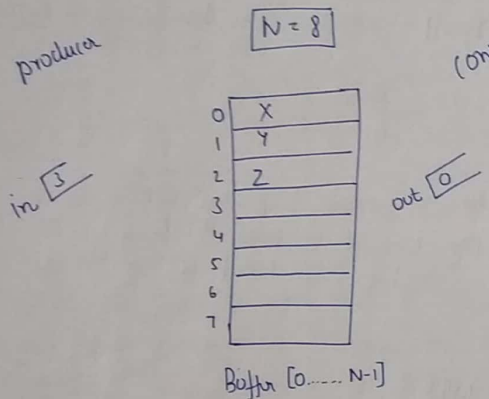
$P_3 P_1 P_2 P_3$

15/9/18

Classical Problems of IPC

↳ Inter Process Communication

1. Producer Consumer with Semaphore




```
Semaphore mutex = 1
Semaphore Empty = N;
Semaphore Full = 0;
Void Producer (void)
{
    int itemp;
    While (True)
    {
        Produce_item (itemp);
        down (Empty)
        down (mutex)
        Buffer [in] = itemp
        in = (in+1) mod N
        Up (mutex)
        Up (Full);
    }
}
```

```
Void Consumer (void)
{
    int itemc;
    While (TRUE)
    {
        down (Full);
        down (mutex);
        Itemc = Buffer [out];
        out = (out+1) mod N;
        Up (mutex)
        Up (Empty)
        process_item (itemc);
    }
}
```

"mutex" is binary Semaphore used by the producer and consumer to access the buffer in a mutual Exclusive manner.

"Empty" is counting Semaphore variable represents no. of slots empty in the buffer at any point of time.

"Fully" is a counting Semaphore variable represents no of slots full in buffer at any point of time.

Q What happens if we interchange down(Empty) & down(Mutex) in the Producer code

a) No problem the soln still works correct.

b) Both producer & 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.

Soln:

P	C
1 down(Mutex)	1 down(Full)
2 down(Empty)	2 down(mutex)
3 up(mutex)	3 up(mutex)
4 up(Full)	4 up(Empty)

If Empty = 0 & Full = N (ie when buffer is full)

P → 1 Mutex = X 0

P → 2 <- suspend

C → 1 Full = ~~N~~ N-1

C → 2 <- suspend Mutex = 0 - 1

Both the producer & consumer are in suspended list and there is no other process to wake them up. Hence they are in deadlock.

Q What happens if you interchange Up(mutex) & a) Up(Full) in the producers code?

Soln:

P	C
1 down(Empty)	1 down(Full)
2 down(mutex)	2 down(mutex)
3 Up(Full)	3 Up(mutex)
4 Up(Mutex)	4 Up(down)

Initially Empty = N, Full = 0 (ie Empty buffer)

P → 1 Empty = ~~N~~ N-1

P → 2 mutex = X 0

P → 3

→ preempt

C → 1 Full = X 0

C → 2 <- suspend

→ P resumes

P → 4 Mutex = 0 1 wakeup Consumer

→ C → 3 Mutex = X 1

Q What Happens if you interchange

d) i) down(mutex) & down(full) in Consumer deadlock

a) ii) up(mutex) & Up(Empty) in Consumer

i)

P	C
1 down(Empty)	1 down(mutex)
2 down(mutex)	2 down(Full)
3 Up(mutex)	3 Up(mutex)
4 Up(Full)	4 Up(down)

Assume initially empty buffer

Empty = N Full = 0 Mutex = 1

C → 1

→ preempt

P → 1

P → 2 <- suspend in suspend list of mutex

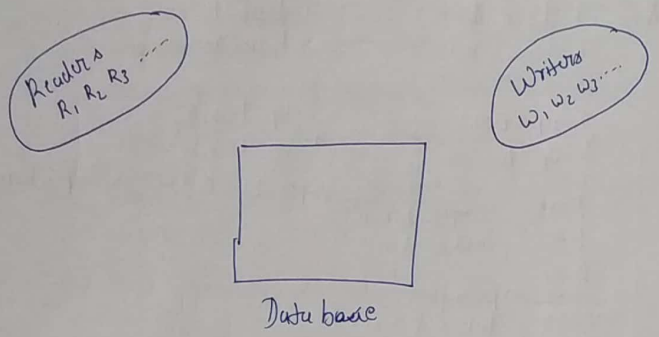
→ C → 2 <- suspend in (suspend list) of Full

ii)

P	C
1 down(Empty)	1 down(Full)
2 down(mutex)	2 down(mutex)
3 Up(mutex)	3 Up(Empty)
4 Up(Full)	4 Up(mutex)

No prob soln works fine

2. Readers Writers Problem




```
int rc = 0
Semaphore mutex = 1;
Semaphore db = 1;
void Reader (void)
{
  While (true)
  {
    1. down (mutex)
    2. rc = rc+1
    3. if (rc==1) down (db)
    4. Up(mutex)
       [ D.B ]
    5. down (mutex)
    6. rc = rc - 1
    7. if (rc==0) Up(db);
    8. Up(mutex)
    process_item (item())
  }
}
```

```
Void writer (void)
{
  While(True)
  {
    1. down (db)
       [ D.B ]
    2. Up (db)
  }
}
```

4 Conditions to be followed

1) R → W X
2) R → R ✓
3) W → R X
4) W → W X

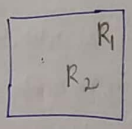
→ "rc" is integer variable represents readers count (ie no. of readers present in the data base at any point of time).

→ "mutex" is binary semaphore used by the readers in a mutual exclusive manner.

→ "db" is a binary semaphore variable used by readers and writers in a mutual exclusive manner.

Analysis ①

rc = 0 ✓ 2
mutex = ✓ 0 ✓ 0 ✓ 1
db = ✓ 0 → writer gets suspended, due to unsuccessful down on db



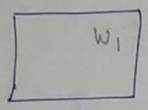
1. R → W X Satisfied
2. R → R ✓ Satisfied

Analysis ①

rc = 0 !
mutex = Y 0
db = Y 0

W1 gets suspended due to unsuccessful down on db.

R1 gets suspended due to unsuccessful down on db



3. W → R ✗ satisfied
4. W → W ✗ satisfied

Q. What happens if we ~~history~~ interchange

i) down(mutex) & rc = rc+1 in readers code.

a) No problem the soln still works correct.

b) multiple readers are not allowed into db

c) Both reader and writer will enter into D.B. at the same time.

d) It is possible for deadlock.

$R_4 \to 1$ rc = 0 !
preempt
$R_1 \to 1$ rc = Y 2
$R_1 \to 2$
$R_1 \to 3$
$R_1 \to 4$
D.B
R_1 resume
$R_1 \to 2$ mutex = Y 0
$R_2 \to 3$
$R_3 \to 4$ mutex = 0 !
D.B

$W_1 \to 1$ db = 0 !
D.B

Here R_1, R_2 & W_1 are simultaneously present in dB.

Q What happens if we interchange "if condition" and Up(mutex) in the readers code ?

Sol: 1. down(mutex)
2. rc = rc+1
3. Up(mutex)
4. if(rc == 1) db(1)

Ans optimal ✓

rc = 0 !
$R_1 \to 1$ mutex = Y 0 !
$R_1 \to 2$ db = 1
$R_1 \to 3$
preempt
$R_2 \to 1$ rc = Y 2
$R_2 \to 2$ mutex = Y 0 !
$R_2 \to 3$ db = 1
preempt
$W_1 \to 1$ db = Y 0
D.B

R_1 → Enter DB
R_2 → Enter in DB.

R_1 W_1
R_2
DB

Q What happens if we interchange Up ~~down~~(mutex) & d) ~~& rc = rc-1~~ in the readers code ? if(rc == 0) Up(db);

Sol:- 1, 2, 3, 4,
D.B

R_1 R_2 R_3
rc = 3
mutex = 1

1. down(mutex)
2. rc = rc - 1
3. Up(mutex)
4. if(rc == 0) Up(db)

① $R_1 \to 1$ | ② $R_2 \to 1$ | ③ $R_3 \to 1$
$R_1 \to 2$ | $R_2 \to 2$ | $R_3 \to 2$
$R_1 \to 3$ | $R_2 \to 3$ | $R_3 \to 3$
$R_1 \to 4$ | $R_2 \to 4$ | preempt
rc = 2 | rc = 1 | rc = 0

④ W_1 → block on db.
$R_4 \to$ rc = 0 !
$R_4 \to$ block on db

$R_3 \to 4$ rc ≠ 0
R_3 does not perform Up(db)

Here no reader and writer ever enter into C.S. despite C.S is empty. This is deadlock

Imp

Q19 & 20 (WB) P-91

int R=0, W=0
semaphore mutex=1

Sol:

```
void reader(void)
{
L1: down(mutex);
    if(W==1)
    {
2       1
    goto L1;
    }
    else
    {
3   R=R+1;
4.      2
    }
    D.B
    down(mutex)
    R=R-1
    Up(mutex)
}
```

```
void writer(void)
{
L2: down(mutex)
    if(      3      )
    {
2       Up(mutex);
    goto L2;
    }
2>  }
3. Up(mutex)  W=1
4. W=1        Up(mutex)
    D.B
    down(mutex)
    W=down(mutex);
    W=0;
    Up(mutex);
}
```

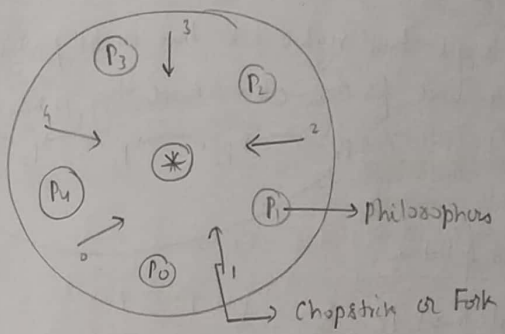
Sol:- (a) Initially R=0, W=0
mutex = 1
$R_1 \rightarrow L_1$
option (a) & (c) are eliminated bcz R_1 can't perform down on mutex bcz it will be suspended
Hence 1 → must be Up(mutex)
2 → must be Up(mutex) to satisfy R=0 ✓
3 → W==1 or R>1 then W should wait

2> $W_1 \rightarrow 1$
preempt ← $W_1 \rightarrow 3$ mutex = 1
$R_1 \rightarrow 1$ mutex = 0
$R_1 \rightarrow 3$
$R_1 \rightarrow 4$
D.B
W_1 resume
$W_1 \rightarrow 4$ → D.B.

Hence Reader & writer both simultaneously enter into CS

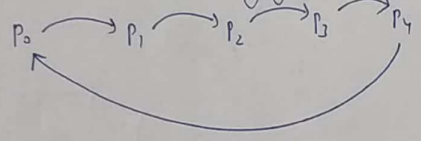
Dining Philosopher's Problem

N = 5




```
void philosopher(int i)   ↳ philosopher no.
{
    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 are trying to take their right fork then nobody will get their right fork then all the philosophers will wait for each other and they go into dead lock.



Correct Solution

```
#define N 5                          → left philosopher
#define LEFT (i+N-1)%N               → right philosopher
#define RIGHT (i+1)%N
#define THINKING 0
#define HUNGRY 1
#define EATING 2
Semaphore mutex = 1;
Semaphore S[N]; // all S[i]'s are initialized to 0.
int state[N]; // an array to keep track of every one's state
Void Philosopher(int i)              → philosopher no.
{
    While (TRUE)
    {
        take_forks(i);
        eat();
        Put_forks(i);
    }
}
```

```
Void take_forks(int i)
{
    down (mutex)
    state[i] = HUNGRY;
    test (i);
    Up(mutex);
    down(S[i]);
}
```

↳ controll the philosopher

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

```
Void put_forks (int i)
{
    down (mutex);
    State[i] = THINKING
    test (LEFT);
    test (RIGHT);
    Up (mutex);
}
```

Note - Even though deadlock is avoided still starvation may be present, there might be a philosopher who never gets a chance to eat bcz if an individual philosopher is allowed to eat repeatedly.

"mutex" is binary semaphore used by the philosophers in the mutual exclusive manner.

"S[N]" is an array of binary semaphore, initially all the assigned to 0.

"state[N]" is an integer array used to keep track of every philosopher's state. Initially all philosophers will be in the thinking state

Analysis :-

mutex = 1

P_0 = T
P_1 = T
P_2 = T H E
P_3 = T
P_4 = T

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

Q Assume that philosopher P_1 & P_3 are in the eating state. Then philosopher P_2 is also trying to go into eating state then which stmnt in the above code is controlling the philosopher?

a) down (mutex)
b) If condition
c) test function
d) down (S[i])

Sol: mutex = 1 0 1

P_0 = T
P_1 = E
P_2 = T H
P_3 = E
P_4 = T

state[2] = 1
S[2] = 0 0 ← suspend

Q What happens if we interchange up(mutex) & down (S[i]) in take_forks ( ).

a) No problem, the soln still works correct.
b) More than 2 philosophers can go into eating state at the same time.
c) It is possible for dead lock
d) None of the above

Sol: assume 2 philosophers are eating

mutex = 1 0

P_0 - T
P_1 - E
P_2 - T H
P_3 - E
P_4 - T

suspend on down mutex in put_forks( )

S[0] = 0
S[1] = 0
S[2] = 0 ← P_2 suspended
S[3] = 0
S[4] = 0

Q. When P_1 and P_3 are in the eating state then if the philosopher P_2 is trying to go into Eating state then it will be suspended. Then how the philosopher P_2 will come back to eating state?

Sol: When P_1 & P_3 finished eating put both the forks down in any order and return to thinking.

Gate

Q Let P[0] .... P[4] be the processes and m[0] ---- m[4] be the binary Semaphore mutexes, all initialized to 1. Each process P_i executes the following code.

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

Consider the below stmnts

I. M.E is satisfied

II. M.E is NOT satisfied

III. It is possible for deadlock

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

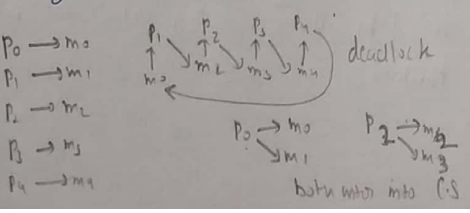
a) only I

b) only II

c) only I, II

d) only I, III

Sol:



Monitors (Never Asked in gate)

Monitors is programming language compiler support type of soln to achieve Synchronization.

Monitors is collection of variables, condition variables and procedures combine together in a special kind of module (or) package.

The processes running outside the monitor can not directly access the internal variables of the monitor but however they can be ~~called~~ (Call) procedures of monitor.

Monitors has an important property that only 1 process can be active inside the monitor at any point of time.

Syntax:

```
monitor Example        <- name of the monitor
{                      <- keyword
    Variables
    condition variables;
    Procedure P1
    {
    }
    Procedure P2
    {
    }
}
```

Condition Variables

Condition x, y;

The 2 different opn are performed on the condition variables of the monitor

1) wait();
2) signal();

wait();

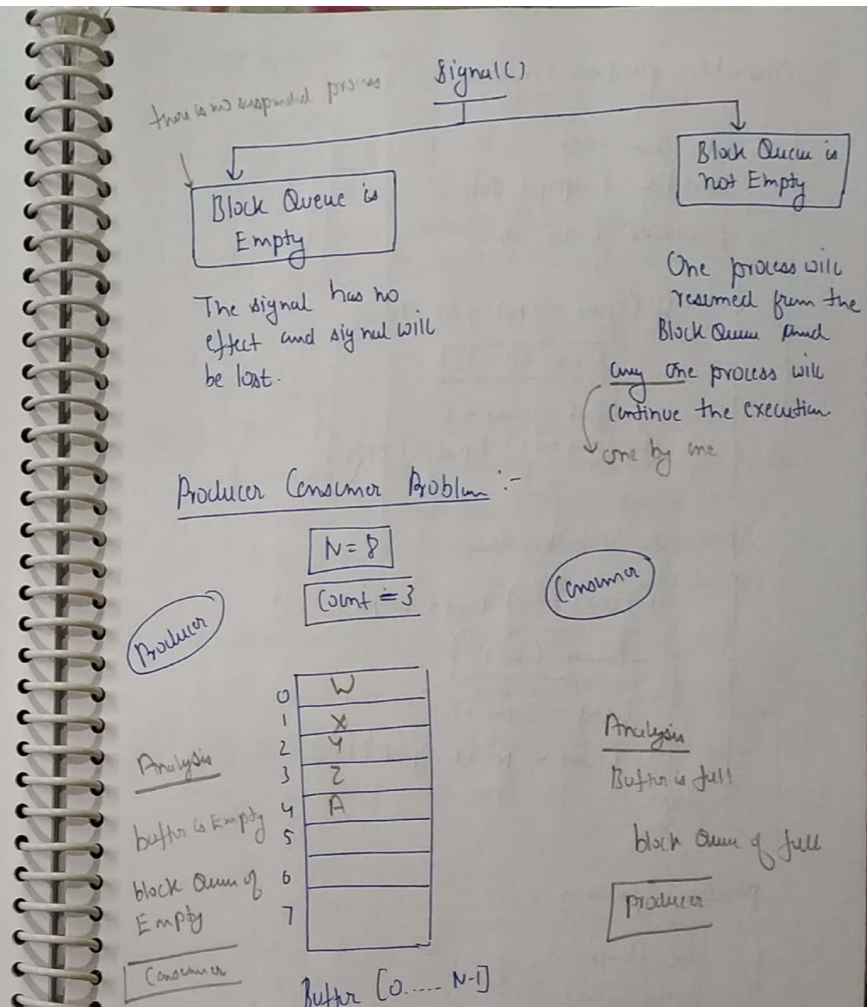
x.wait(); (or) wait(x);

The process performing wait() opn 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);

—



```
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 itemp;
    While (true)
    {
        produce_item (itemp);
        producer_consumer. Enter_item;
    }
}
```

```
procedure consumer
{
    int itemc;
    While (true)
    {
        producer_consumer. Remove_item;
        process_item (itemc);
    }
}
```

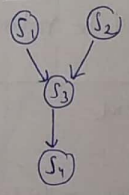
NOTE: Even though we are using count variable there will not be any inconsistency because inside the monitor only 1 process can be active at any point of time.

Concurrent Programming

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

Read set = { b, c, e, f, a, d, g, i }
Write set = { a, d, g, h }

Precedance Graph



NOTE:

1) Any 2 stmnts S_i, and S_j can be executed concurrently (or) parallely if they are following the below conditions.

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

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

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

2) The real concurrent programming is possible only on the multiprocessor system.

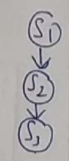
3) The concurrent word has different meanings

Concurrent
- → They can execute concurrently (or) parallely
- → They don't have any dependency
- ⊛ → Any one can start first.

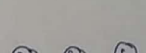
4) The concurrent program will be written by using

parallel ← par begin – par end (or) ~~co begin~~ co begin – co end → concurrent

begin
S_1 ;
S_2 ;
S_3 ;
end

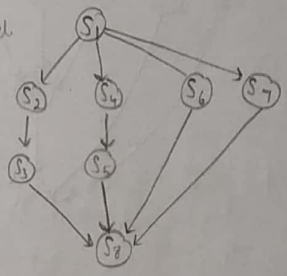


par begin
S_1 ;
S_2 ;
S_3 ;
par end

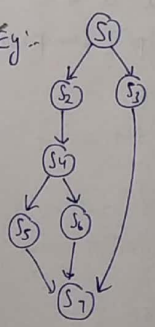


Eg:

```
begin                <- by default if nothing is given at start and end
  S1 ;
  par begin
    begin
      S2 ;
      S3 ;
    end
    begin
      S4 ;
      S5 ;
    end
    S6 ;
    S7 ;
  par end
  S8 ;
end
```

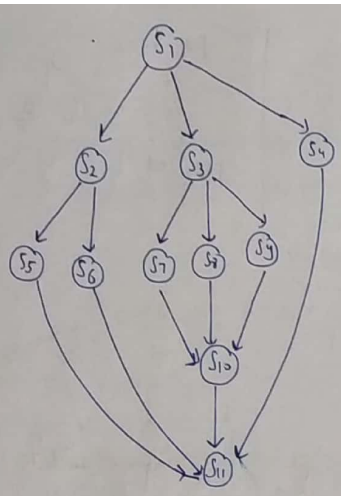


Eg:



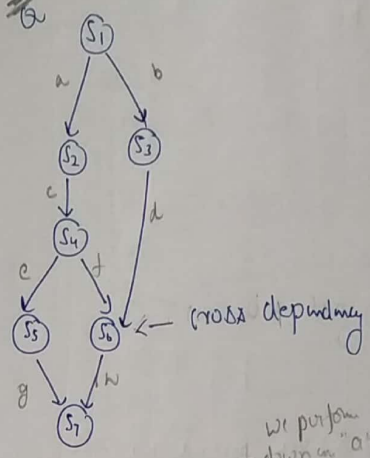

```
S1 ;
par begin
  begin
    S2 ;
    S4 ;
    par begin
      S5 ;
      S6 ;
    par end
  end
  S3 ;
par end
S7 ;
```

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 ~~Comm~~ concurrent prog.d by using par begin and par end for all the precedence graphs. But it is very much possible with the help of Semaphore operations



a
b
c
d
e
f
g
h

sol:

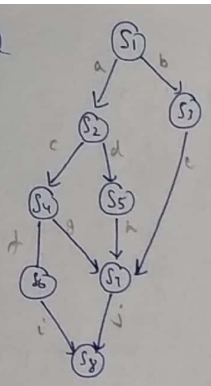
```
par begin
    begin S1, par begin V(a), V(b), par end, end;
    begin P(a), S2, V(c), end.
    begin P(b), S3, V(d), end;
    begin P(c), S4, par begin V(e), V(f), par end, end
    begin P(e), S7, V(g), end
    begin P(d), P(f), S6, V(h), end
    begin P(g), P(h), S7 end;
par end
```

we perform "down" on "a" to make sure that S1 is executed

we do "up" op^n to check dependency for S2 & S3

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

Q



pw

par begin
begin S_1 par begin V(a); V(b) par end end
begin P(a) S_2, par begin V(c); V(d) par end, end
begin P(b) S_3 V(e) end
begin P(c) S_4 par begin V(f) V(g) par end end
begin P(d) S_5 V(h) end
begin P(f) S_6 V(i) end
begin P(g) P(e) S_7 V(j) end
begin P(i) P(j) S_8 end

Q int x = 0, y = 0;

Par begin

1 | begin P_1
  | x = 1
2 | y = y + x;
  | end

1 | begin P_2
  | y = 2;
2 | x = x + 3;
  | end

Par end

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

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

Which of the following claims are possible

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

Sol: $P_1 \to 1$ x = 0 1
$P_2 \to 1$ y = 0 2
$P_2 \to 2$ x = X 4
$P_1 \to 2$ y = X 6
III x = 4, y = 6

$P_2 \to 1$ y = 0 2
$P_2 \to 2$ x = 0 3
$P_1 \to 1$ x = X 1
$P_1 \to 2$ y = X 3
II x = 1 y = 3

Q25 P-92 (WB) initially S=2 x=0

Sol

X	W	Y	Z
1 P(S)	P(S)	P(S)	P(S)
2 x=x+1	x=x+1	~~y~~ x=x-2	x=x-2
3 V(S)	V(S)	V(S)	V(S)

S = 2 X Ø X Ø X Ø X 0

x = Ø - 2 - 4 X 2

① X→1, X→2 ② Y→1, Y→2, Y→3 ③ Z→1, Z→2, Z→3 ④ X→3 ⑤ W→1, W→2, W→3

prompt →

max value of x = +2

W	X	Y	Z
P(S);	P(S);	P(S);	P(S)
I. load R_w, m[x]	I. load R_x, m[x]	I. load R_y, m[x]	I. load R_z, m[x]
II INC R_1 R_w	II INCR, R_x	II DECR R_y	II DEC(R_z)
III Store m[x], R_w	III Store m[x], R_x	III Store m[x], R_y	III Store m[x], R_z
V(S);	V(S)	V(S)	V(S)

minimum value of x = -4

① Y→1, Y→2 ⑤ Z→1, Z→2, Z→3 ② X→1, X→2, X→3 ③ W→1, W→2, W→3 ④ Y→3

S = 2 X Ø 1

x = Ø X 2 - 2 - 4 minimum value of x = -4

Jump

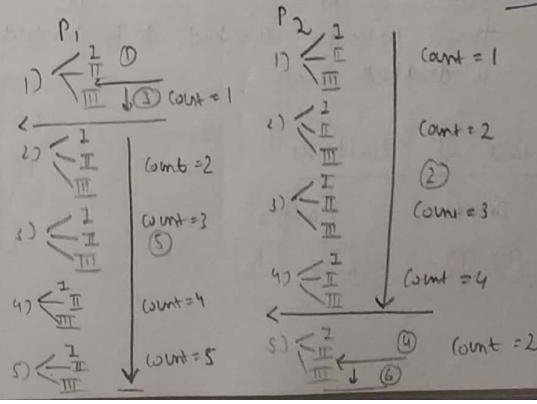
Consider the following concurrent program

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

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

NOTE: count = count + 1; will execute in "3" instructions like load, increment, and then store.

Ans: 2



Count = 0, 4, 7, 8, 12

P_1 : I 1st iteration R_1

P_1 : II 1st iteration R_1 0 1

→ preempt

P_2 : 1, 2, 3, 4

→ preempt

P_1 : III 1st iteration

P_2 : 5th iteration → I

P_2 : 5th iteration → II R_2 1, 2

P_1 : 2 3 4 5 iterations

P_2 : 5th iterations → III

16/9/18

Deadlocks

Def: 2 or more processes are waiting on some event to happen which never happens then those processes are said to be involved in a deadlock.

Basics of deadlocks

processes: P_1 P_2

resources: R_1 R_2

Resource with instances

R_1 . } Single instance of resource R_1

R_1 }

R_1 : 2 instances of resource "R_2"

Requesting Edge

i) P_1 → R_1 process P_1 is requesting for 1 instance of resource R_1

ii) P_2 → R_2 process P_2 is requesting for 2 instances of R_2.

Allocation Edge

R_1 → P_1 1 instance of resource R_1 is allocated to process P_1.

2 instances of resource R_2 are allocated to process P_2.

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

R.A.G = G(V, E)

- Processes, Resources
- Requesting, allocating

The resource request / release life cycle

i) The process will make a request for the resource

ii) O.S clearly validates the request of the process.

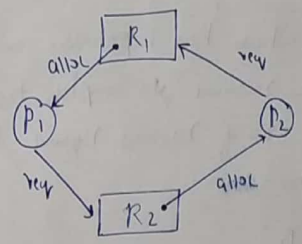
iii) If the request made by the process is valid then O.S will check for the availability of resource.

iv) If the resource is freely available then it will allocated to the process otherwise the process has to wait.

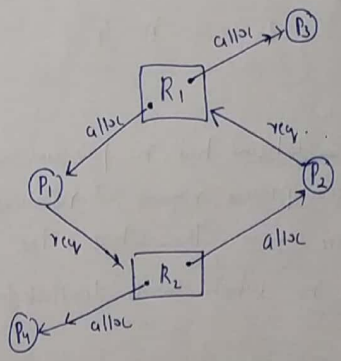
v) If all the resources required for the start of execn are allocated then the process will go into execn.

* vi) Once execn of the process is completed then it will release all the resource.

Eg:



deadlock exists in the R.A.G



No deadlock exists in the R.A.G

Q. Consider a system which has n processes and 6 Tape drives. Each process requires 2 Tape Drives to complete their execn. then what is max value of n which ensures deadlock free execn?

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

causing deadlock

Sol: $\begin{bmatrix} 1 & 2 & 3 & \cdots & n \\ 1 & 1 & \cdots & & \end{bmatrix} \times \begin{matrix} P_1 & P_2 & P_3 & P_4 & P_5 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$ (P_6: 1)

$n = 5$

Q Consider a system which has 3 processes and each process requires 2 resources to complete their exec. What is minimum no of resources required to ensure deadlock free exec.

Sol:

P_1	P_2	P_3	
1	1	1	3
1			

$n = 4$

Q Consider a system which has n processes and 6 resources. Each process requires 3 resources to complete their exec. Then what is the maximum value of n which ensures deadlockfree exec.?

Sol:

P_1	P_2	P_3
2	2	2

causing deadlock

$2 \times n = 6$

$n = 3$ ← causes deadlock

$\max n = 3 - 1 = 2$

max $n = 2$

Q 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 min no of resources required to ensure deadlockfree exec.?

Sol

P_1	P_2	P_3	
4	8	12	= 25

Imp

Q23 P - 58 (WB)

Sol:

P_1	P_2	P_3	P_4	$R = 20$
4	3	7	$x = 3$	
3	2	6	$x-1 = 2$	

$x = 3$

max demand of $P_4 = 2$ → P_4 should get all the resources so that it can execute and avoid deadlock

$R = 20$

P_1	P_2	P_3	P_4	
4	3	7	x	
3	2	6	x	$20 - 11 = 9$

Concept: To 1 process give all the resources and to remaining processes give 1 less than what required.

1) Necessary Condition: The deadlock may be possible or may not be possible

2) Sufficient Condition: The deadlock never possible

↳ The least upper bound which satisfies the Condition is called as sufficient Condition.

P-97

13

Sol:- P_1 P_2 P_3 ..... P_n $R = m$

S_1 S_2 S_3 ..... S_{n-1}

S_1-1 S_2-1 S_3-1 S_n-1

$$(S_1+S_2+\dots+S_n)-n < m$$

$$S_1+S_2+\dots+S_n < m+n$$

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

Q10 - P-96

Sol:- P_1 P_2 P_3 ..... P_n

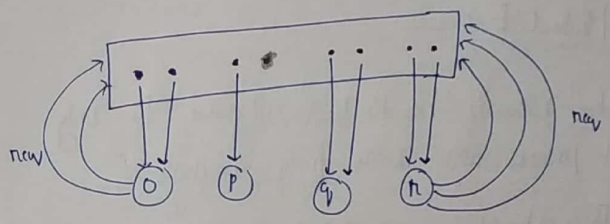
x_1 x_2 x_3 ..... x_n

y_1 y_2 y_3 ..... y_n

$$\sum_{i=1}^{n} x_i = R$$

$p \neq q$ $y_p = y_q = 0$

b) $x_p + x_q \geq \max y_k \quad k \neq p,q$



$x_0 = 2$ $y_0 = 2$

$x_p = 1$ $y_p = 0$

$x_q = 2$ $y_q = 0$

$x_n = 2$ $y_n = 3$

$y_p = y_q = 0$

Concept of deadlocks

1) Deadlock characteristics
2) Deadlock prevention
3)* Deadlock avoidance
4) Deadlock Detection
6) Deadlock Recovery

Deadlock characteristics (Behaviour of deadlock)

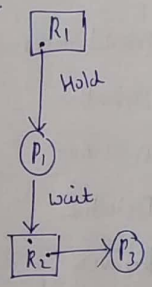
1. Mutual Exclusion

→ The resource has to be allocated to only 1 process (or) it is freely available

→ There should be a on 1 to 1 relationship b/w the resource and the process

2. Hold And Wait

→ The process is holding the resource and waiting on some other resource simultaneously.



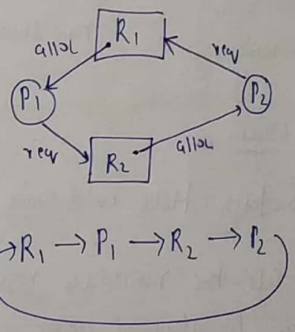
3) No preemption

→ The resource has to be voluntarily ~~released~~ released by the process after completion of exec n.

→ 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.

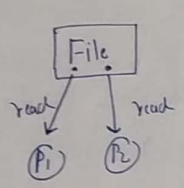


NOTE: If all the above 4 conditions are existing simultaneously in the system then definitely there exists a deadlock.

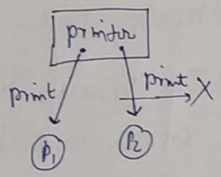
Deadlock Prevention

1. Mutual Exclusion

→ It is not possible to dissatisfy the mutual exclusion always because of sharable and non-sharable ~~req~~ resources.



sharable resource



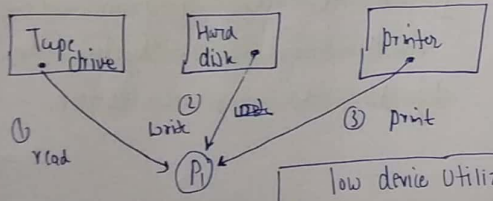
non sharable resource

2) Hold And Wait

→ To dissatisfy Hold and wait

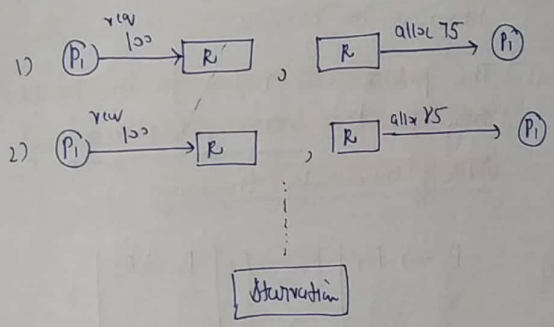
i) Allocate all the resources required by the process by start of exec.

ii)



low device Utilization
(or) less resource Utilization

iii) The process should release all the existing resources by making new request



If the above condition is followed to dissatisfy hold and wait then it is possible for the process to go into starvation

3) NO Preemption

Let 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

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

if the process P_2 is in the execn then the process P_1 has to wait

if the process P_2 is not in the execn & it is waiting on some other resource R_2 then preempt ~~resource R_2~~ resource R_1 from process P_2 and allocate it to process P_1

4. Circular Wait

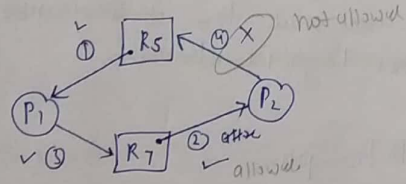
To dissatisfy Circular Wait

i) Resources will be assigned with a unique numerical numbers

ii) The process can request for the resource only in the "increasing (or) decreasing" order of enumeration. (numbering)

E.g: $P_1 \rightarrow R_7$ ✓ | $P_1 \rightarrow R_5$ ✗ | $P_1 \rightarrow R_9$ ✓

E.g ②:



Deadlock Avoidance

Deadlock avoidance is implemented by using Bankers Algorithm

	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_1	3 2 2	2 0 0	5 3 2	1 2 2
P_2	9 0 2	3 0 2	7 4 3	6 0 0
P_3	2 2 2	2 1 1	7 5 3	0 1 1
P_4	4 3 3	0 0 2	10 5 5	4 3 1
		7 2 4	10 5 7	

total available

A B C
↓ ↓ ↓
10 5 7

Resources → A, B, C

processes → P_0 P_1 P_2 P_3 P_4

Remaining need = max need − current alloc

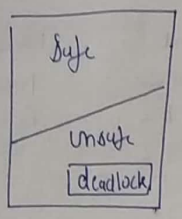
Available = total resources − out of

A B C
3 3 2

Current avail = total avail − current alloc

Safe Sequence

P_1, P_3, P_0, P_2, P_4



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

2. If the system is in the unsafe state then it is possible for the deadlock ~~to~~

3. The order in which we satisfy the remaining need of all the processes is called as safe sequence

4. The safe sequence may not be unique, there can be multiple safe sequences.

5. The unsafe state ~~purely~~ purely depends on the behaviour of the processes.

*6. The deadlock ~~and~~ avoidance is less restrictive than deadlock prevention

7. Each and every time when the process is requesting for any resource the bankers algorithm is implemented to identify whether the system is in the safe state or in unsafe state..

8. 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.

Q2 P-95

Sol:

	max A B C	allot A B C	avail A B C	need A B C
P_0	6 5 4	0 3 4	4 3 1	6 2 0
P_1	3 4 2	2 1 2		1 3 0
P_2	1 0 4	0 0 2		1 0 2
P_3	3 2 5	1 2 1		2 0 4

P_1 4 3 1
2 1 2
6 4 3

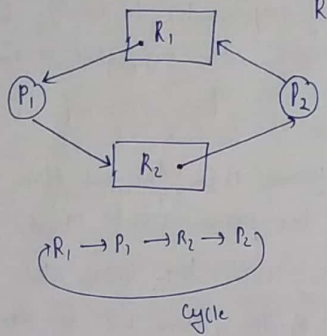
6 4 3 we can satisfy P_0 also & P_2 also

$P_1 P_0 P_2 P_3$
$P_1 P_2 P_3 P_0$] - both are safe sequence

Deadlock Detection

i) The resources are of single instance type

Run: Cycle detection algorithm



1) If all the resources are of single instance type then cycle in resource allocation graph is necessary & sufficient for occurring of deadlock

2) If all the resources are not of single instance type then cycle in the R.A.G is just a necessary condition but not sufficient condition for occurring of deadlock.

ii) Resources are of multiple instance type

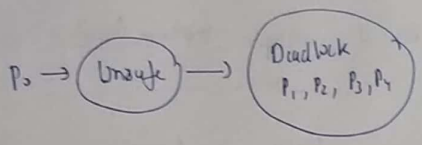
	Current allocation A B C	Current available A B C	Remaining need A B C
P_0 →	0 1 0	0 0 0	0 0 0 → P_0
P_1 →	2 0 0	0 1 0	2 0 2 → P_1
P_2 →	3 0 3	3 1 3	0 0 0 → P_2
P_3 →	2 1 1	5 1 3	1 0 0 → P_3
P_4 →	0 0 2	7 2 4	0 0 2 → P_4
		7 2 4	

P_0, P_2, P_1, P_3, P_4

NOTE: What happens if the process P_2 is requesting for additional resources of $P_2 \rightarrow (0,0,1)$

0 0 0 (old)
0 0 1 (new)
0 0 1 (latest)

	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	↑	0 0 1 → P_2
P_3 →	2 1 1	Unsafe we can't satisfy P_2	1 0 0 → P_3
P_4 →	0 0 2	$P_2(0,0,1)$	0 0 2 → P_4

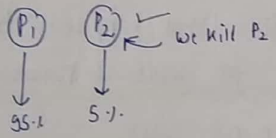


Deadlock Recovery

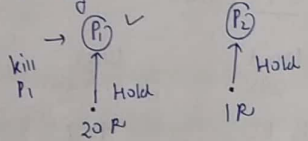
i) Killing the process.

a) Kill all the processes which are involved in the deadlock.

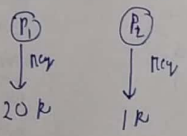
b) kill one after the other
- → low priority will be killed first
- → based on % of process completion



- → based on no. of resources process is holding



- → no of resources the process is requesting

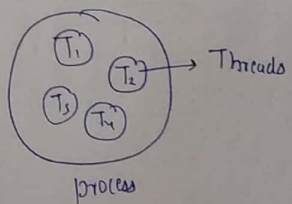


2) Resource Preemption
- → Resources will be preempted from the processes which are involved in the deadlock and the preempted resources will be allocated to some other processes so that there is a possibility to recover system from deadlock
- → If the above condition is followed then there may be a possibility for a process to go into starvation

3) Ostrich algorithm
- → ignore the deadlock

Threads

Def: The light weight process
- → # instructions are less
- → # context is less

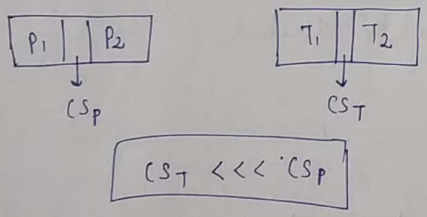


Advantages of Threads

i) Responsiveness

If the process is divided into multiple threads then if one thread has completed the exec^n then immediately o/p will be responded this response will be faster compare to the response of the process

ii) Faster Context Switching



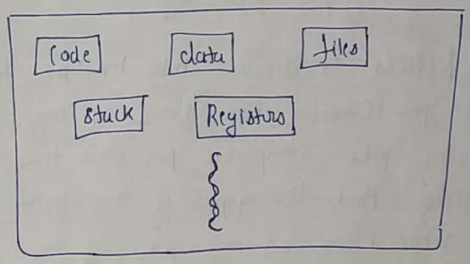
$CS_T <<< CS_P$

The C·S time b/w Threads will be very less compared to C·S time b/w processes. bcz the threads will have less ~~Context~~ compared to context the processes.

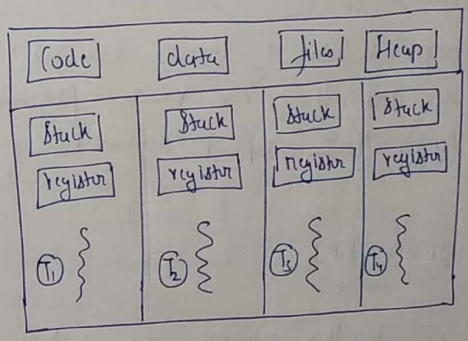
3. Effective Utilization of multiprocessor System

If the process is divided into multiple threads then different threads can be scheduled on to different processors so that process exec^n will be faster.

4. Resource Sharing



Single threaded processes



Multi-threaded process.

→ The resources like code, data (global variables), files, Heap & memory will be shared among all the threads within the process. but 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 you consider one thread as one job then no. of jobs completed per unit time will increase. And throughput of the system will be enhanced.

6) Economical

The implementation of threads does not require any cost and there are various programming lang. API's which support implementation of threads.

Eg: JAVA API

Threads are further categorized into 2 types

i) User level Threads

ii) Kernel Level Threads

User level Threads	Kernel Level Threads
1. User level threads are implemented by User (or) programmer	1. Kernel Level Threads are implemented by O.S.
2. O.S does not know about user level threads, it views user level thread as a process only. O.S can not recognize user level threads.	2. Kernel-level threads are recognized by O.S.
3. If one user-level thread is performing blocking system call, then the entire process will go into block state	3. If one kernel level thread is performing blocking system call then another thread will continue the exec.
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 H/w support required for user level threads.	7) Scheduling of kernel-level threads require H/w support

NOTE: "GRAVG" scheduling is used in the threads.

I/O Operations

I/O of the processes is categorized into 2 types

i) Synchronous I/O
ii) Asynchronous I/O

i) Synchronous I/O

→ In synchronous I/O the process performing I/O opn will be placed in the block state till the I/O opn is completed.

→ Once I/O opn is completed an interrupt service routine (ISR) will be initiated which brings the process from block state to ready state

ISR - Interrupt service routine

ii) Asynchronous I/O

→ In asynchronous I/O while initiating the I/O request "handler fxn" will be registered
↑ all information regarding I/O opn given to handler

→ The process is not placed in the blocked state and it continues to execute the remaining code after initiating the I/O request.

→ Once the I/O operation is completed the signal mechanism is used to notify the process that the data is available and registered handeler function will be asynchronously invoked.

Memory Management

Text book - O.S by William Stalling

Main Memory / Primary memory / Physical Memory / RAM

functionality: Allocating and deallocating the memory to the process.

goal: The efficient Utilization of memory by minimizing the internal and external fragmentation.

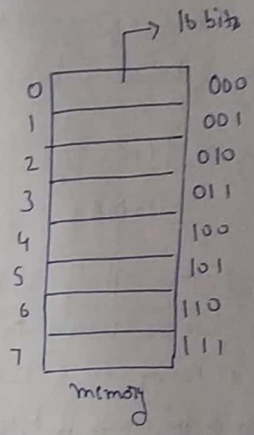
Memory is represented as

Memory
- Word Addressable
  Eg: 512 kW
- byte Addressable
  Eg: 512 kB

Basics

$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$

$2^{10} = 1024 \cong 1k \approx 10^3 \approx 1k$
$2^{20} = 1M$
$2^{30} = 1G$
$2^{40} = 1T$



Capacity of memory = # of words in memory * word size

Capacity = $8 \times 2B = 16B$

Q Consider a system which has 512 GW and each word is having size of 128 bits then what is capacity of memory in Bytes?

Sol: Capacity = 512 GW
= 512 G x 7bits $\frac{2^7}{2^3}$ bits
= $512 G \times 2^4$
= 8 TB

Q Consider a system which has 64 MW and each word is having size of 48 bits then what is the capacity of memory in bytes?

Sol: Capacity = $64M \times \frac{48}{8}$

= 384 MB

Q Consider a system where 32 binary bits are used to represent all the words of the memory and each word is having size of 32 bits then what is the capacity of the memory in bytes

Sol: Capacity = $2^{32} \times \frac{32}{8} = 2^{34} = 16GB$

RAM chip Implementation

RAM chip Size = 128B

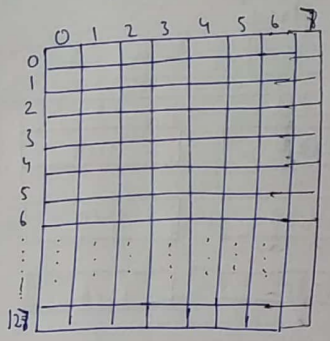
by using 128B RAM chip organize Main memory capacity of 16kB

a) # of RAM chips required

b) Draw Memory Organization map

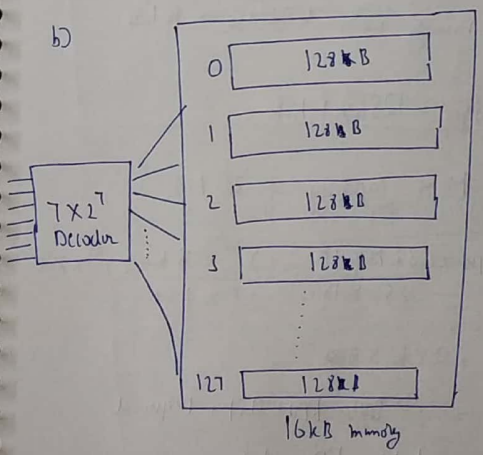
c) What is the size of decoder required?

Sol: RAM chip size = 128B



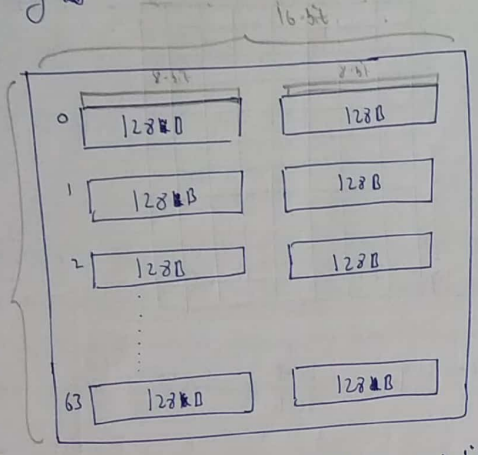
a) # ram chips = $\frac{16kB}{128B} = 128$

b)



c) Decoder required to select rows

a) Draw the memory organization map with word size of memory is 16 bit.



16kB memory with word size as 16 bits

Q RAM chip size = 256×1 bit

Organize the M.M capacity of 32 MB

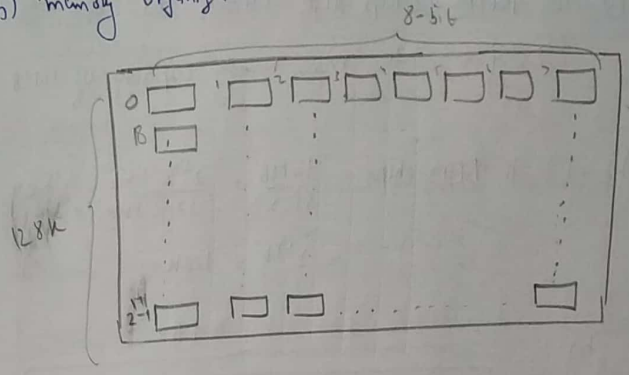
Sol. a) # RAM chips = $\frac{32M \times 1B}{256 \times 1bit} = \frac{2^{25} \times 8\ bit}{2^{8} \times 1\ bit} = 2^{17} \times 8bit$

$= 128k \times 8$

$= 2^{20}$ RAM chips required

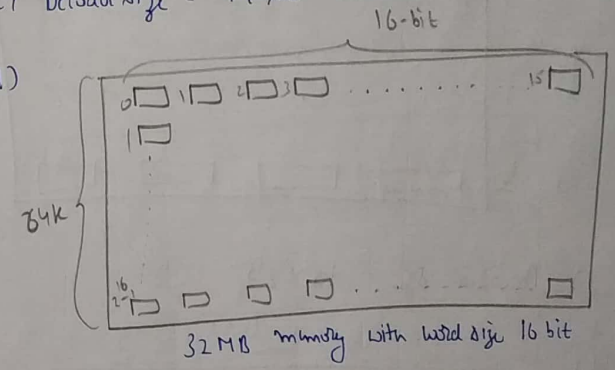
$= 1M$ RAM chips "

b) memory organization with word size of 8 bit



c) Decoder size = 17×2^{17} decoder

d)



32MB memory with word size 16 bit

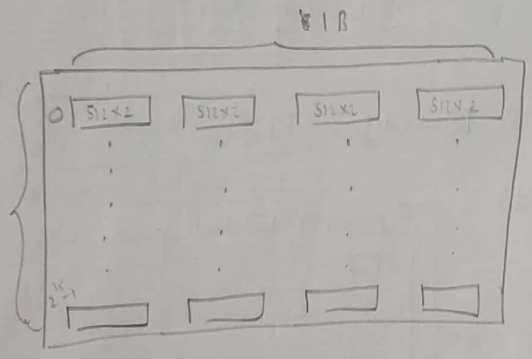
Q RAM chip size = 512 X 2-bit

Organize the main memory capacity of 16MB

Sol a) # RAM chips = $\frac{16MB}{512 \times 2} = \frac{2^{24} \times 8\,bit}{2^9 \times 2^1 bit} = 2^{15} \times 4$

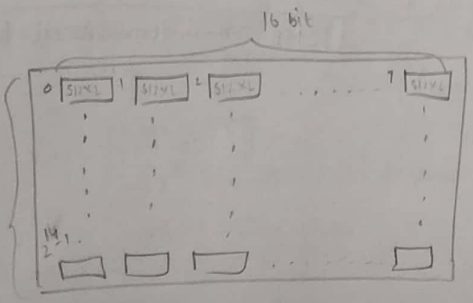
$= 2^{17} = 128K$

b)



c) Decoder size = 15×2^{15}

d)



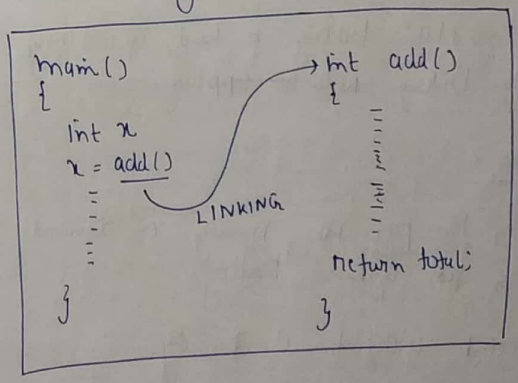
Loading, Linking and Address Binding

Loading :-

bringing the program from secondary memory to main memory is called as Loading.

Linking :-

Establishing the linking blw all the modules of the program (or) all the functions of the program in order to continue the program execn is called as linking.



Loading and Linking is further categorized into 2 types:-

1) Static
2) Dynamic

Static

→ Loading the entire prg. into memory b4 start of prg. execn. is called as static Loading

→ Inefficient utilization of memory.

→ Prg execn will be faster

→ If the static loading is used, accordingly static linking will be applied.

Dynamic

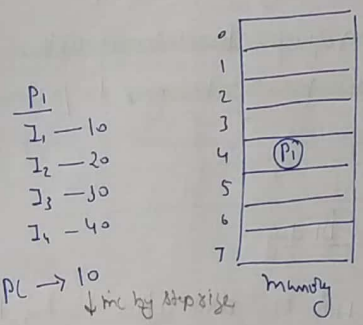
→ Loading the prg. into memory on demand is called as dynamic loading

→ Efficient utilization of memory.

→ Prg execn will be slower

→ If the dynamic loading is used, accordingly the dynamic linking will be applied.

Address Binding



Def: Association of prg instns and data with to the actual physical memory locations is called as address binding.

It is categorized in 3 types

1) Compile time A·B
2) Load time A·B
3) Execution time AB or Dynamic AB

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 A·B will be done by loading the prog. into memory.

→ The compiler requires to interact with the O.S with memory manager to perform address binding.

2) Load Time Address Binding

→ This type of A·B will be done after loading the prog. into memory.

→ The Load time A·B will be done by O.S Memory Manager (ie Loader)

3) Execution Time (or) Dynamic Address Binding

→ The A·B will be postponded even after loading the prog. into memory.

→ The prog. will keep on changing the locations in the memory till the time of prog exec.

→ This type of A·B will be done by processor at the time of prog exec.

NOTE: Majority of the O.S practically implements dynamic loading, dynamic Linking & dynamic A·B.

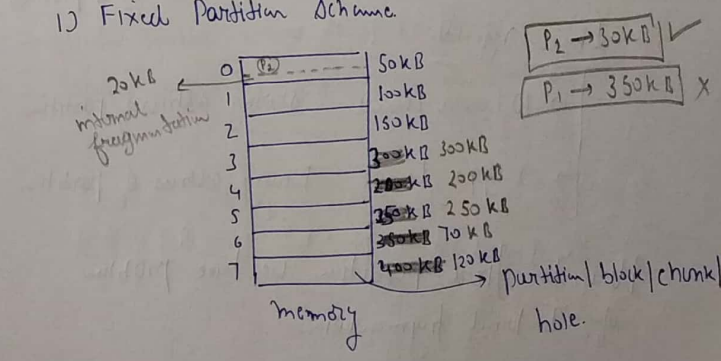
Ex: - Windows, Unix, Linux etc.

Memory Management Techniques

- Contiguous
  - 1) Fixed partition scheme
  - 2) Variable partition scheme
- Non-contiguous
  - 1) → Paging
  - 2) → Multi Level paging
  - 3) → Inverted paging
  - 4) → Segmentation
  - 5) → Segmented paging

Contiguous Memory Management Techniques

1) Fixed Partition Scheme



→ In the fixed partition scheme, memory is divided into fixed no. of partitions

→ fixed means no. of partitions are fixed in the memory

→ In every partition only 1 process will be accomodated.

→ The 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 "limit registers". Security is provided

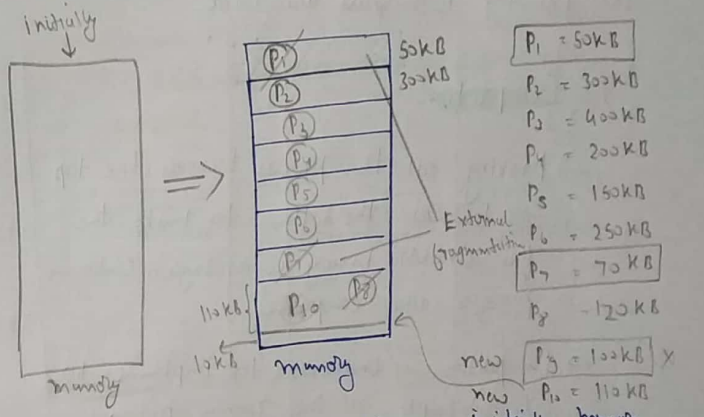
Limit register :-

→ 1) Lower Limit : Starting address of partition

→ 2) Upper Limit : Ending address of partition

Here In fixed partition we have problem of Internal fragmentation

Variable Partition Scheme

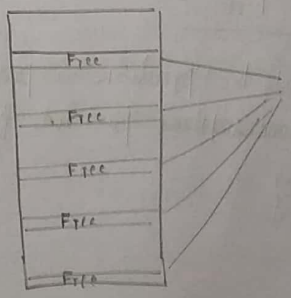


P_1 = 50KB
P_2 = 300KB
P_3 = 400KB
P_4 = 200KB
P_5 = 150KB
P_6 = 250KB
P_7 = 70KB
P_8 = 120KB
P_9 = 100KB ×
P_{10} = 110KB

→ In the variable partition scheme, initially memory will be single continuous free block

→ When ever the request by the process arrive accordingly the partition will be made in the memory

→ If the smaller process keep on coming then the larger partitions will be made in smaller partitions



To avoid the problem of external fragmentation the following techniques are used

1) Compaction

→ Moving all the process towards the top (or) towards the bottom to make the free available memory in a single continuous place is called compaction.

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

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 free available partitions.

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

3) Worst fit :-

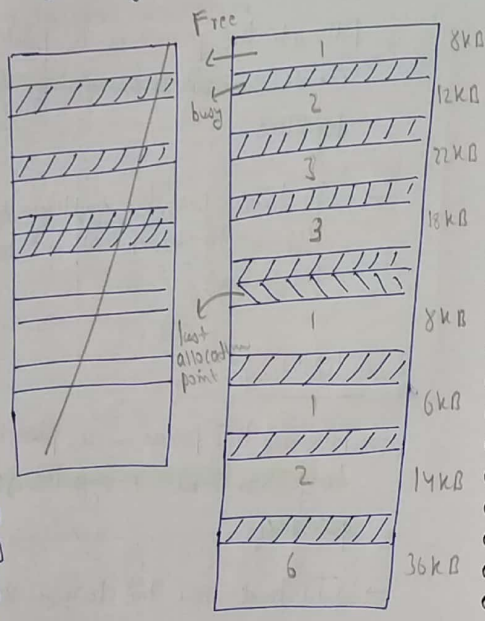
→ Allocate the process in a partition which is largest sufficient among the free available partitions

→ 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 fit first fit but it will search for the first sufficient partition from last allocation point.

Q Consider the following memory map



i) First fit
ii) Best fit
iii) Worst fit
iv) Next fit

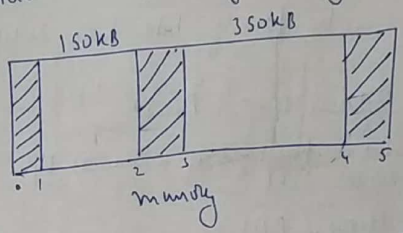
i) First fit : 22KB
ii) Best fit : 18KB
iii) Worst fit : 36KB
iv) Next fit : 36KB

(P) 16KB

Q How many successive request of 6KB 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 request from the processes are 300KB, 25KB, 125KB, 50KB respectively.

The above results could be satisfied with ?

(Assume variable partition scheme)

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

17/3/18

Non-Contiguous Memory Management Techniques

1) Paging

1) Logical Address space (LAS) (or) Virtual Address Space (VAS)

→ Represented in form of Words (or) Bytes

Ex: 512 kW (or) 512 kB

2) Logical Address (L.A) (or) Virtual Address (V.A)

→ Represented in the form of bits. Ex: 32-bit

3) Physical Address space (P.A.S)

→ Represented in the form of words (or) Bytes.
Ex: 32kW (or) 32kB

4) Physical Address (P.A)

→ Represented in the form of bits. Ex: 16-bit

assume word addressable

Ex:1) LA = 27 bits

LAS = 128 MW

Ex:2) LA.S = 32kW

L.A = 15 bits

assume byte addressable

Ex:3) P.A = 48 bits

256TB

Ex:4: P.A.S = 16GB

= 34 bits

→ The technique of mapping (CPU generated logical address to physical address. is called as paging.

Assume L.A = 13 bits

LA.S = 2^{13} = 8kW

P.A = 12 bits

P.A.S = 4kW

page size = 1kW = frame size

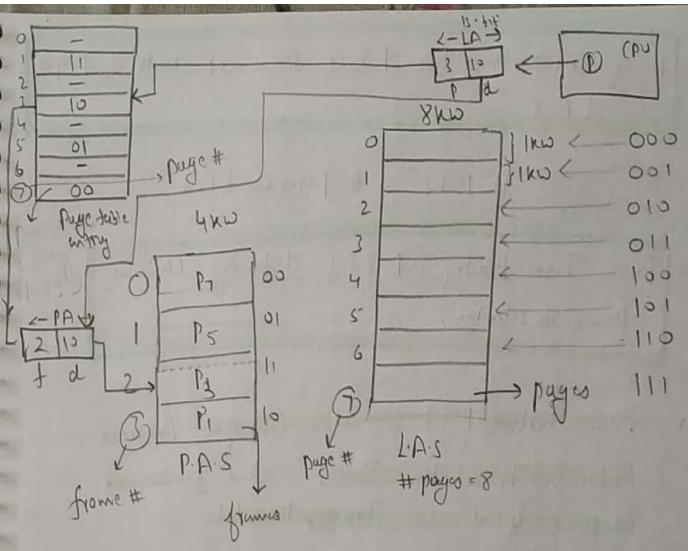
$$\#\text{ pages} = \frac{L.A.S}{\text{page size}}$$

$$\#\text{ pages} = \frac{8kW}{1kW} = 8$$

(assume word addressable)

$$\#\text{ frames} = \frac{P.A.S}{\text{frame size}}$$

$$\#\text{ frames} = \frac{4kW}{1kW} = 4$$



→ The logical Address space (L.A.S) is divided into equal size pages.

→ The P.A.S is divided into equal size frames

→ page size is always same as frame size

page size = = 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

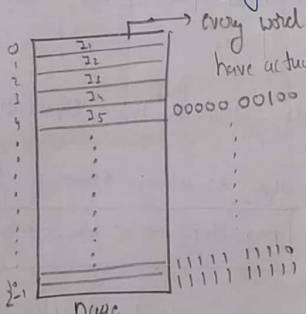
→ # of Entries in the PT is same as # of pages in L.A.S

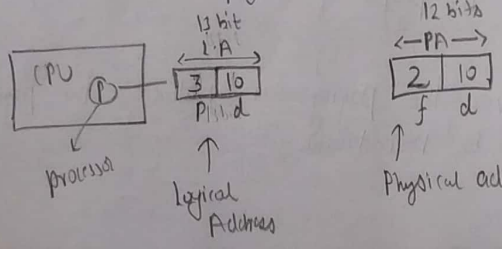
P.T.E = # pages in L.A.S

→ Page Table Entry (P.T.E) definitely contains frame number.

→ Page Table (P.T) is also called as ~~Address~~ Address Translation Table. And frame no. is also called as Translation bits.

→ page size = 1kw means page will have 1k words

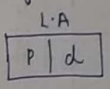




I p = no. of bits required to represent pages of L.A.S

(or)

page #



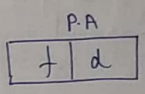
d = no. of bits required to represent page size

(or)

word number of the page

(or)

page offset



II f = no. of bits required to represent frames of P.A.S

(or)

frame #

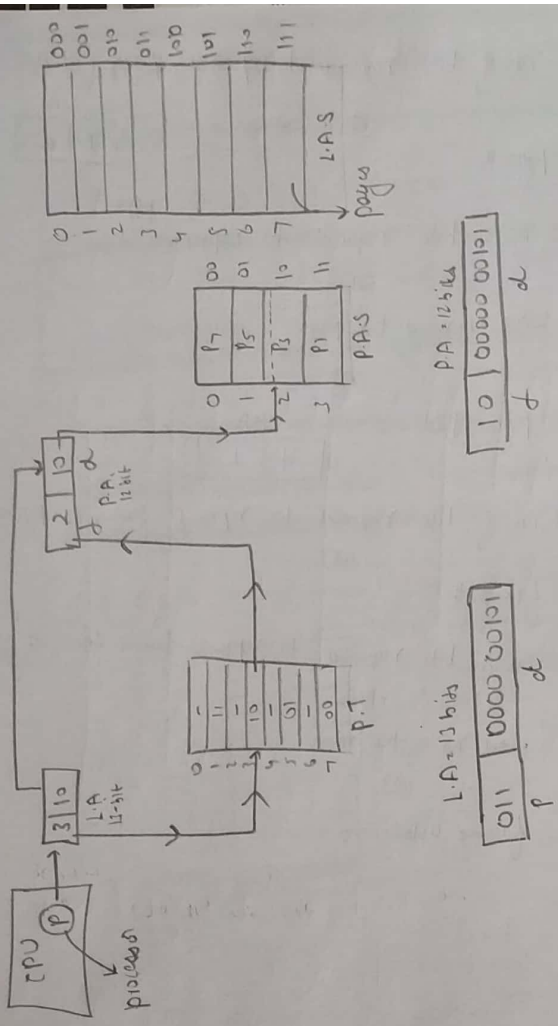
d = no of bits required to represent frame size or

(or)

word no of the frame

(or)

frame offset



Important Points (Crazy)

i) Whenever the process is created, paging will be applied on the process. and page table (P.T) will be created and Base address of page table will be stored in the P.C.B

ii) Paging is w.r.t every process and every process will have its own page table.

iii) Page tables of the processes will be stored in the main memory.

iv) There is no external fragmentation in the paging.

v) The internal fragmentation exists in the last page and internal fragmentation in the paging is considered as $\boxed{P/2}$, where 'p' is page size
↳ means (half) page)

vi) Maintaining the PT is considered as overhead for the system.
↳ PT just contains mapping & there are no instr

NOTE: i) Windows practically follows page size of 4KB

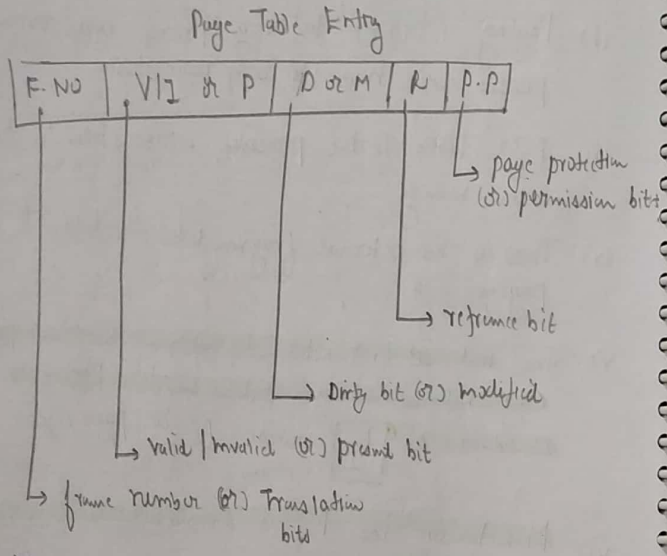
ii) PTE definitly contains frame #, but some times often along with frame #, it may also contain additional bits like :

1) Valid / Invalid bit or Present bit

2) Dirty bit or modified bit

3) Reference bits

4) page protection bits



1) Valid / Invalid bit (or) Present bit

→ This bit is used to identify whether the page is available or not available in the main memory.

0 → not present

1 → present

2) Dirty bit (or) modified bit

→ This bit is used to identify whether the page is modified or not modified by the processor while accessing the page as part of the exec.

1 → modified by the processor

0 → not modified by the processor

3) Reference bit

→ This bit is used to identify how many times page is referred, ~~refered~~ by the processor while exec.

4) Page Protection (or) Permission bit

→ This bit is used for the protection and security from unauthorized access.

Q) Consider a system which has L.A of 27 bits & P.A of 21 bits and memory is word addressable & page size is 4KW. Then calculate # pages, & # frames.

Sol: # pages = $\frac{2^{27}}{2^{12}} = 2^{15} = 32K$

frames = $\frac{2^{21}}{2^{12}} = 2^{9} = 512$

Date 9/5

File Systems

Q Overhead for formatting a disk is 96 bytes for a 4000 byte sector find unformatted capacity of the disk

a) Given, no of surfaces: 8
Outer diameter of disk: 12cm
inner diameter of disk: 4cm
Inner ~~Inter~~ track space: 0.1mm
Number of sectors per track: 20

Sol:- no of tracks = $\frac{\text{recording width}}{\text{inner space b/w track}}$

recording width = $\frac{(\text{Outer diameter} - \text{inner diameter})}{2}$

$= \left(\frac{12-4}{2}\right)$

$= 4$ cm

$\therefore$ no of tracks = $\frac{4\text{ cm}}{0.1\text{mm}}$ = 400 tracks

Since unformatted disk space has been asked, so no 96 bytes in 4000 bytes sector would be wasted.

unformatted capacity = $S \times \frac{I}{S} \times \frac{\text{Sec}}{T} \times \frac{B}{\text{Sec}}$

$= 8 \times 400 \times 20 \times 4000$

$= 256000000$ Bytes

$= 256$ MB

b) Given, disk rotating speed = 3600 rpm
find data transfer rate

3600 r — 60 s

1 r — $\frac{1}{60}$ s

$\frac{1}{60}$ s — 1 track

$\frac{1}{60}$ sec ⟶ 20 × (4000 − 96) overhead for formatting, since formatting space need to be removed

1 sec ⟶ 20 × 3914 × 60 bytes

1 sec ⟶ 136968 bytes

Gate 2007

Maximum cardinality of the request set so that head changes its direction after servicing every request if total no of tracks are 2048. and head can start from any track.

Request set → $2^n - 1$

$n = \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047\}$

∵ no of tracks are 2048 ∴ $|n| = 11$

Test

given m frames

length of page reference string is p with "n" distinct page numbers occurring in it.

find upper bound & lower bound for any page replacement algorithms

Worst case ⇒ if m = 1 ie we have 1 frame, then for every page reference, there will be a page faults.

So total length is "p" Upper bound is p

Best case → all the pages ~~can be accommodated~~ in reference string must be in m frames.

minimum → n (∵ we have n distinct pages)

FL - Basic 4

→ TLB miss on a page fault requires exception mechanism

→ TLB misses can be handeled either in H/w or in S/w

→ CPU scheduling criteria for performance evaluation
  1) CPU Utilization and throughput
  2) TAT and WT
  3) Response Time

→ blocking refers to operations that block further execution until that operation finishes.
Non-blocking refers to code that doesn't block execution

User level threads are suitable for non-blocking threads means it require less I/O opn. biz When user level thread are blocked all threads will be blocked.

Switching b/w User threads only require procedure calls not context switching

→ If deadlock is there then no progress
→ If bounded waiting not there then there is possibility of starvation

→ Monitor s
  → prevent multiple process from executing monitor code at the same time
  → Uses condition variable
  → hides mutual exclusion details from calling function
  → Process can not access monitor's data from procedures declared outside monitor

→ Rand Robin scheduling gives always give better performance compared to FCFS. → False (if TQ is very large)

→ An advantage of System call is to provide an interface b/w running prg. & Operating System. → False (b/w prg & OS)

→ In a system with Virtual memory Context switching includes extra overhead in switching of address space
- Hence VM increases Context switching overhead

⇒ User-level threads switching does not require Context switching

→ Message passing in inter process communication requires Kernel support

⇒ A lock that uses busy waiting is called spink lock

→ The set of pages that a process is currently using is called working set

→ Loading the pages before letting process run is called pre paging

→ Unique behaviour of Optimal Page replacement algorithm

⇒ reference string - 12345 12345
↳ worst case scenario for LRU & FIFO
↳ best case scenario for MRU

reference string - 1 2 3 4 5 5 4 3 2 1
↳ best case scenario for LRU & FIFO
↳ worst case scenario for MRU

by observing reference string, optimal convert itself into best possible page replacement algorithm

Tests

Schedulers

LTS is responsible for controlling degree of multiprogramming
STS is responsible for effective context switching
MTS is responsible for swapping

During context switching b/w 2 threads following values will be changed
i) Program counter
ii) Stack pointer

→ Zombie process
A process becomes zombi process if for example when the parent process is available and it has not yet called wait() & then child process becomes zombie process.
"but child has called exit()"

→ A process which has finished the execution but still has entry in the process table to report to its parent process is known as zombie process

→ A child process always first becomes a zombie b4 being removed from the process table

---

LRU, FIFO, optimal PRA all are static
working set dynamic PRA.
↳ observe recent past, and make decision about nearest future. (ie recent past of which pages have been refered)
→ dynamically allocate frames to process
→ process preempt with 3 working set = {1,2,3}, again when reloaded, it will not use demand paging, it is loaded with

same working set, it had to preemption.

→ User level threads are scheduled by thread library, and kernel knows nothing about it.

→ Scheduler process ~~can~~ not does not interrupt running process

→ In OS, indivisibility of operation means processor can not be preempted.

→ During context-switching from process A to B, Operating system invalidates TLB so that TLB coincides with currently executing prg.

Test

```
for(i=1 ; i<=n; i++)
    fork();
```

→ no of child process created / no of new process created $= 2^n - 1$

Capacity of FAT (File Allocation table) = $\frac{\text{disk capacity}}{\text{block size}}$

→ Thrashing can be reduced by i) decreasing degree of multiprogramming
ii) Increasing RAM size

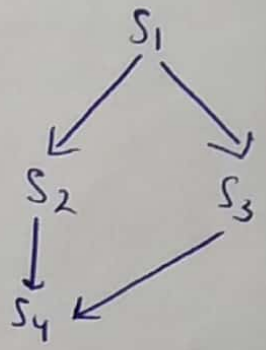
→ Smaphores are used to synchronize critical resources to prevent contention

~~I did mistake —~~

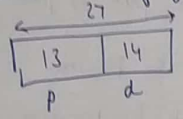
I did mistake – Smaphores are used to synchronize critical resource to prevent deadlock

```
S1
Par begin
Begin
S2 ; S4
End;
S3;
Par end
```

Whatever is present blw they execute concurrently

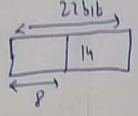


Q Consider a system where no of pages are 8K, and page size is 16KB. Memory is byte addressable and P.A is 22 bits. Then Calculate L.A & no of frames.?

Sol: L.A = 

= 27 bit

frames = 2^8 = 256

P.A = 

Q Consider a system which has L.A.S of 128 MW. And P.A is 24 bits. And memory is word addressable. The P.A.S is divided into 8K frames. Then what is the page size and how many pages in L.A.S.

Sol: page size = $\frac{2^{24}}{2^{13}} = 2^{11}$ = 2KW

pages = $\frac{2^{27}}{2^{11}} = 2^{16}$ = 64K

Q Consider a system which has L.A of 32-bits. and P.A.S = 64MB. Memory is Byte addressable and page size is 4KB. Then what is approx. size of Page Table in bytes

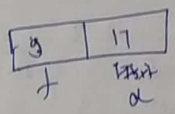
Sol: # frames = $\frac{2^{26}}{2^{12}} = 2^{14}$ = 14-bit for frame no.

# of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$. PT size = $2^{20} \times \frac{16}{8}$ = 2M

Page Table size = # of Entries * size of PTE

Q Consider a system having P.T with 4K entries and L.A is 29 bits then what is P.A if the system has 512 frames.

Sol: page size = $\frac{2^{29}}{2^{12}} = 2^{17}$ = 128KB

P.A = 

= 26 bits

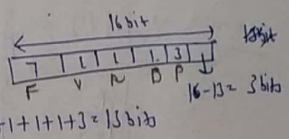
Q40 P-103 (WD)

Sol: L.A.S = P.A.S = 2 16 bytes

P.S = 512B

frames = $\frac{2^{16}}{2^9} = 2^7$

P.T.E size = 2B



7+1+1+1+3 = 13 bits

Guate 2016

Q (1M)

Consider a system which has L.A of 40 bits. & Page size = 16KB memory is byte addressable. And P.T.E size is 48 bits then what is P.T size _____ in Mega Bytes.

Sol: # pages = $\frac{2^{40}}{2^{14}} = 2^{26}$

PT size = $2^{26} \times \frac{48}{8}$ = 64 x 6 MB

= 384 MB

Q Consider a system which has L.A.S = P.A.S = 'S' Bytes and page size is 'P' Bytes and P.T.E size is "e" Bytes. Memory is byte addressable. Then what is the Optimal value of page by minimizing memory overhead of maintaining page table size and internal fragmentation in paging

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

Sol:

L.A.S = P.A.S = S P.S = P

P.T.E = e

pages = $\frac{S}{P}$ PT size = $\frac{S}{P} * e + \frac{P}{2}$

overhead → $f(P) = \frac{S}{P} * e + \frac{P}{2}$ → internal fragmentation ← minimize

$\frac{df(P)}{dP} = \frac{-Se}{P^2} + \frac{1}{2} = 0$

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

$\sqrt{2Se} = P$

$\frac{d^2f(P)}{dP^2} = \frac{+2Se}{P^3} > 0$ at $P = \sqrt{2Se}$

~~maximum~~ ~~minimum~~ ~~maximum exists~~

∴ minima exists at $P = \sqrt{2Se}$

Performance of Paging

→ The main memory access time is = "m"

→ The page tables are stored in the main memory.

Then the formula for Effective Memory Access Time

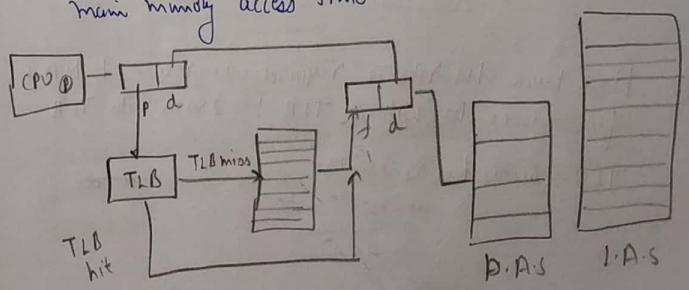
E.M.A.T = 2m

→ The Translation Lookaside buffer (TLB) is added to improve the performance of paging.

→ TLB is H/W device implemented by using associative registers.

→ TLB contains frequently refered page no's and corresponding page frame no's.

→ TLB access time is very less compare to main memory access time



TLB access time = C

TLB Hit ratio is α

Then formula for E.M.A.T

EMAT = C + (1-α)(m) + m

(C: access TLB; (1-α)(m): access PT with miss rate; m: access MM to get the byte.)

$$EMAT = \alpha(c+m) + (1-\alpha)(c+2m)$$

EMAT = (LA → PA) + access byte from MM

18/9/18

Q Consider a system where TLB access time is 20ns & M.M access time = 100ns. TLB Hit ratio is 95%. Then what is EMAT with TLB & without TLB

Sol: With TLB

E.M.A.T = (0.95)(20+100) + (0.05)(20+200)

= 120 × 0.95 + 0.05 × 220

= 114 + 11

= 125 ns

Without TLB

EMAT = 2m

= 2 × 100 = 200 ns

Q How much Hit ratio is required to reduce E.M.A.T from 300ns to without TLB to 250ns with TLB.

TLB access time is 60 ns.

Sol: m = 150 ns

250 = α(60+150) + (1-α)(60+300)

250 = 210α + 360 − 360α

150α = 110

$\alpha = \frac{11}{15} = 0.733$

Hit ratio is 73.3%.

Multilevel Paging

Q Consider a system which has L.A of 32 bits and Page size of L.A.S = 4KW and memory is word addressable and P.T.E size is 1W. Then what is PT size.

Sol: # of pages = $\frac{2^{32}}{2^{12}} = 2^{20}$

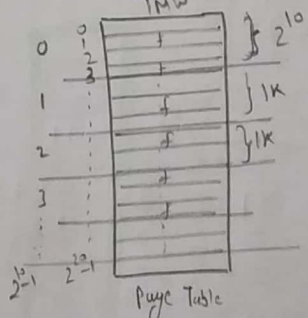
P.T size = $2^{20} \times 1W = 1MW$

Since we need to go for multilevel paging ∵ PT size > Page size

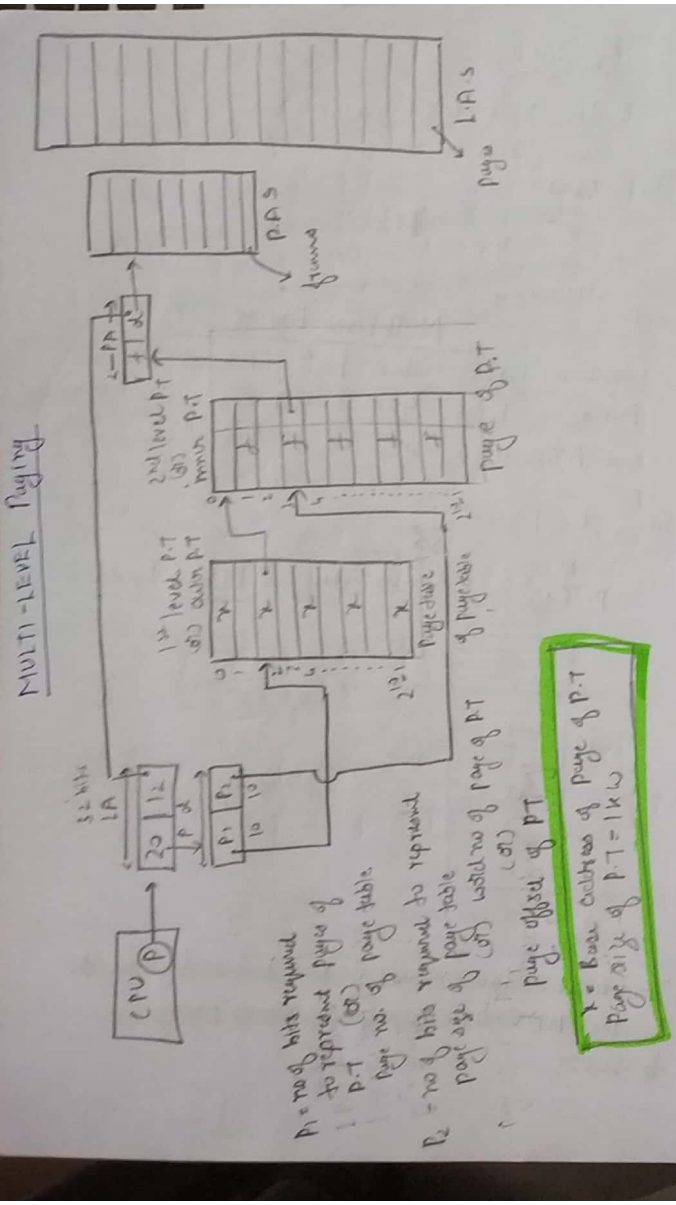
Page size of P.T = 1KW

# of pages in P.T = $\frac{2^{20}W}{2^{10}W}$ = 1K

# of pages in P.T: 0 to $2^{10}-1$



→ To avoid the overhead of maintaining large size P.T in the M.M the concept of multilevel paging will be used.



→ In the multi level paging, paging will be applied on the page table and instead of bringing the entire P.T into memory The pages of P.T will be brought into memory.

Q Consider a system with 2-level paging applicable. The P.T has divided into 2k pages and Each page is having 4k Entries. The memory is word addressable. The P.A.S is 64MW which is divided into 8k frames. PTE size in both the levels is 2W. Then Calculate

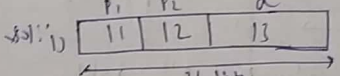
i) Length of logical address?

ii) Length of Physical address?

iii) 1st level P.T size?

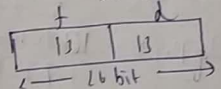
iv) 2nd Level P.T size? (Page of P.T)

Sol:- i)



pages $= \frac{2^{26}}{2^{13}} = 2^{13}$

ii)



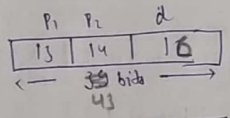
iii) P.T size (1st level) = $2^{13} \times 2W = 8kW$

iv) P.T size (2nd level) = $4k \times 2W = 8kW$
(Page of PT)

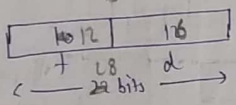
Q Consider a system with 2-level paging applicable. The P.T is divided into 8k pages and each page is having 16k entries. Memory is Byte addressable. The P.A.S is 256 MB which is divided into 64KB frames. the page P.T.E size in both the levels is 32 bits. Then calculate

i) Length of L.A
ii) Length of P.A
iii) 1st level P.T size
iv) 2nd level P.T size (page of P.T)

Sol: i)



ii)



iii) 1st level PT size = $2^{15} \times \frac{32}{8} = 2^{15} = 32KB$

iv) 2nd level PT size (page of P.T) = $16k \times \frac{32}{8} = 64kB$

Performance of 2-level Paging

→ The m.m access time is "m"

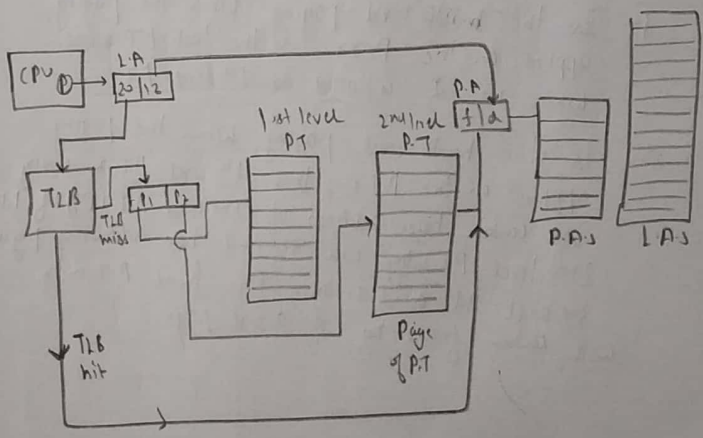
MM A.T = m

→ The P.T's are stored in the M.M, then the formula for E.M.A.T

E.M.A.T = 3m

→ The TLB is added to improve the performance of paging.

→ The TLB contains frequently referred Page Numbers and corresponding frame no's.



→ TLB Access Time = 'c'

TLB Hit ratio is 'x'

Then formula for E.M.A.T

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

For K-level Paging $[c + [(1-x)(km)] + m]$

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

$\downarrow$ $km+m$

Important Points:

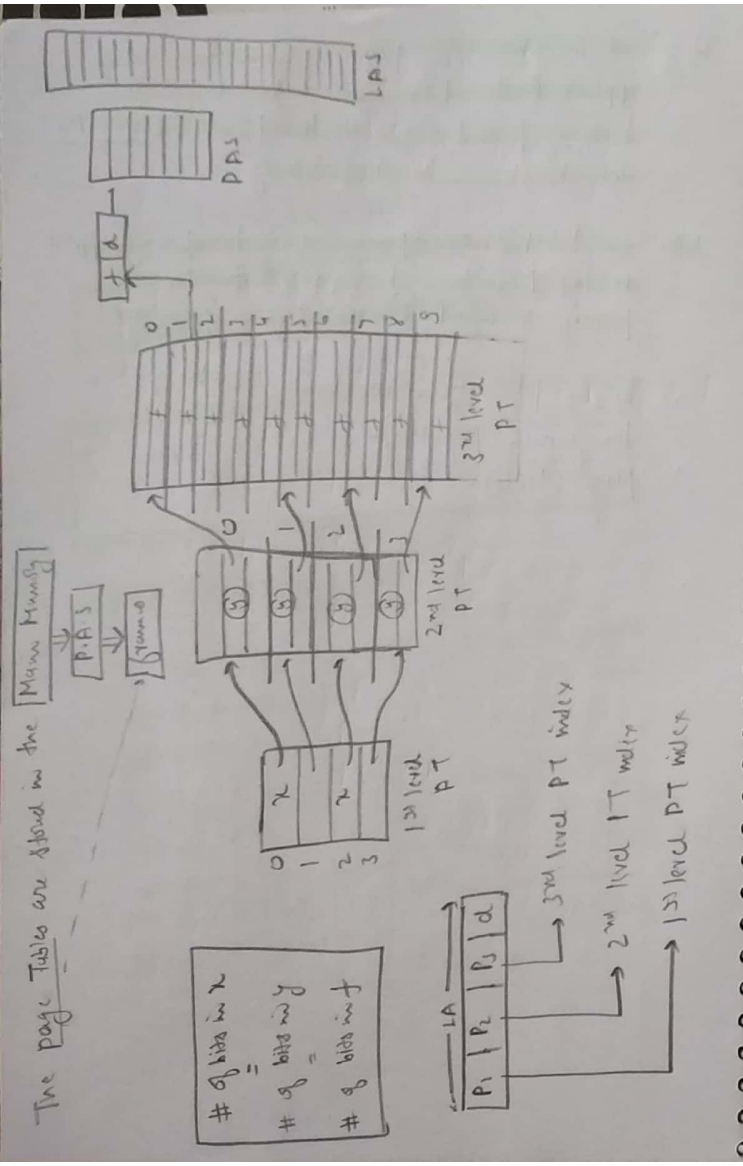
1) In the multi-level paging when the paging applied on the P.T's the last PT which we get is called as 1st level P.T.

2) In the multi-level paging when the paging applied on the P.T's, then 1st level PT entry will contain base address of 2nd level PT (page of P.T) 2nd level PT entry will ~~2nd level~~ base address of pages of 3rd level PT and so on.. The final P.T entry will contain frame no. of actual page.

3) In the multi level paging when the paging applied on the PT's then whatever may be the levels of ~~pages~~ paging, all the PT's (page of P.T) will be stored in the main memory.

4) In the multi-level paging, when the paging applied on the PT's then whatever may be the levels of paging; all the P.T.E's will contain frame no.

5) If the page size is not mentioned in the problem then generally page size will be same in all the places. (levels)



Q25 P-132 (WB)

Sol: LA = 32 bit

pages = $\frac{2^{32}}{2^{10}} = 2^{22}$

PT size = $2^{22} \times 4B$

= 4MB > P.size

Page size = 1KB

$\frac{16MB}{2^{10}} = \frac{2^{24}}{2^{10}} = 2^{14}$ pages

$PT_2 = 2^{14} \times 4B = 64KB > P.size$

$\frac{64KB}{2^{10}} = 64B$

$PT_3 = 64 \times 4B = 256B < P.size$

Concept: **P.T size ≤ Page size**

1st Time Paging

P.T size = $\frac{2^{32}}{2^{10}} \times 4B = 2^{22} \times 4B = 16MB$

now 16MB > Page size

2nd Time Paging

P.T size = $\frac{16MB}{2^{10}} = 2^{14} \times 4B = 64KB$

now 64KB > Page size

3rd Time Paging

1st level P.T size → P.T size = $\frac{64KB}{2^{10}} \times 4B = 2^6 \times 4B = 2^8 B = 256B$

now **256B ≤ Page size** ✓

Gate 2013 (2M)

Q Consider a system which has logical address of 46-bits and P.A of 32-bits. Memory is Byte addressable. The P.T.E size is 32-bits. The O.S uses 3 level paging for ~~L.A~~ Logical to physical address translation. and 1st level PT size is exactly same as page size. Then what is the page size?

Let Page size be "P"

Sol: 1st level PT = $P/4$

2nd level PT = $(P/4) * (P/4)$

3rd level PT = $(P/4)^3 = \frac{2^{46}}{P}$

$P^4 = 4^3 \times 2^{46}$

$= 2^6 \times 2^{46}$

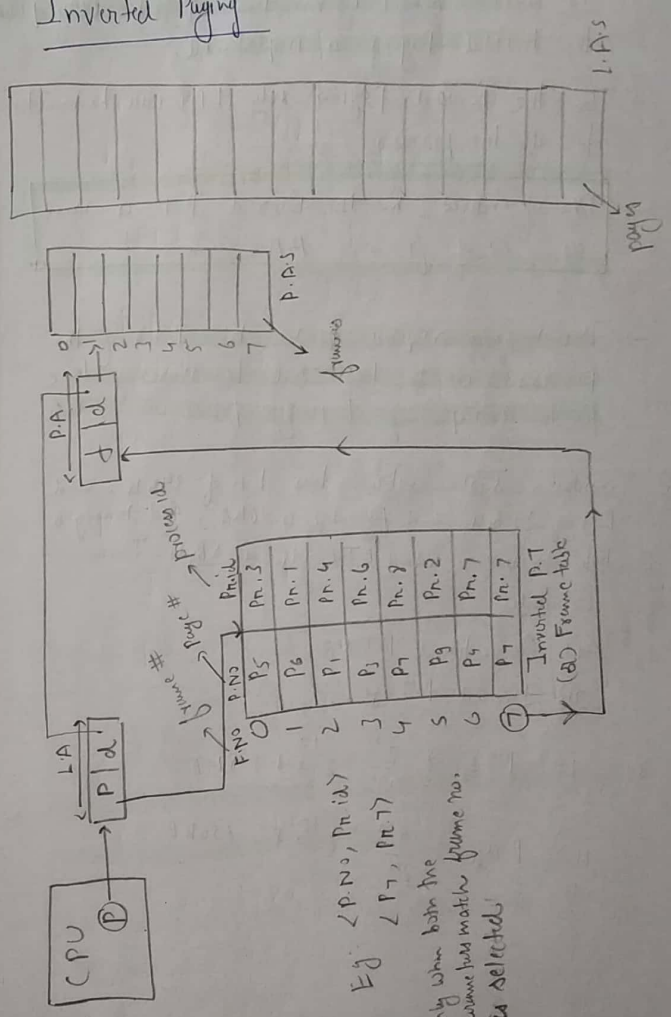
$P^3 = 2^{52}$

$P = 2^{13}$

$P = 8KB$

19/9/18

Inverted Paging



→ To avoid overhead of maintaining P.T's for every process the Inverted Paging is implemented.

→ In the inverted paging only 1 PT will be maintained for all the processes.

→ No. of entries in the inverted P.T is same as no. of frames in P.A.S

→ The memory required to maintain P.T of the processes 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 of 34 bits. and P.A of 29 bits and page size is 16KB. and memory is byte addressable. And P.T.E size is 8Bytes. Then Calculate

i) Conventional P.T size ?

ii) Inverted P.T size ?.

Sol: i) PT size = $\frac{2^{34}}{2^{14}} = 2^{20} \times 8 = 8MB$

ii) P.T size = $\frac{2^{29}}{2^{14}} = 2^{15} \times 8 = 256KB$

Q2) P-102 (WB)

Sol. P.T.E = 4B.

$P.T = \frac{2^{32}}{2^{12}} = 2^{20} \times 4 = 4MB$

$I.P.T = \frac{2^{17}}{2^{12}} = 2^5 \times 4 = 128B$

$Ratio = \frac{PT}{I.P.T} = \frac{2^{22}}{2^7} = \frac{2^{15}}{1}$

Q5 P-100 (WB)

Sol: V.A = 32bit P.A = 30 bit

P.S = 4KB

$\#frames = \frac{2^{30}}{2^{12}} = 2^{18}$

Size $I.P.T = \frac{2^{30}}{2^{12}} \times 2^{18} \times (20+12)$

$= 2^{18} \times \frac{(32)}{8} 4$

$= 2^{20} B$

I.P.T size $= 2^{20} B$

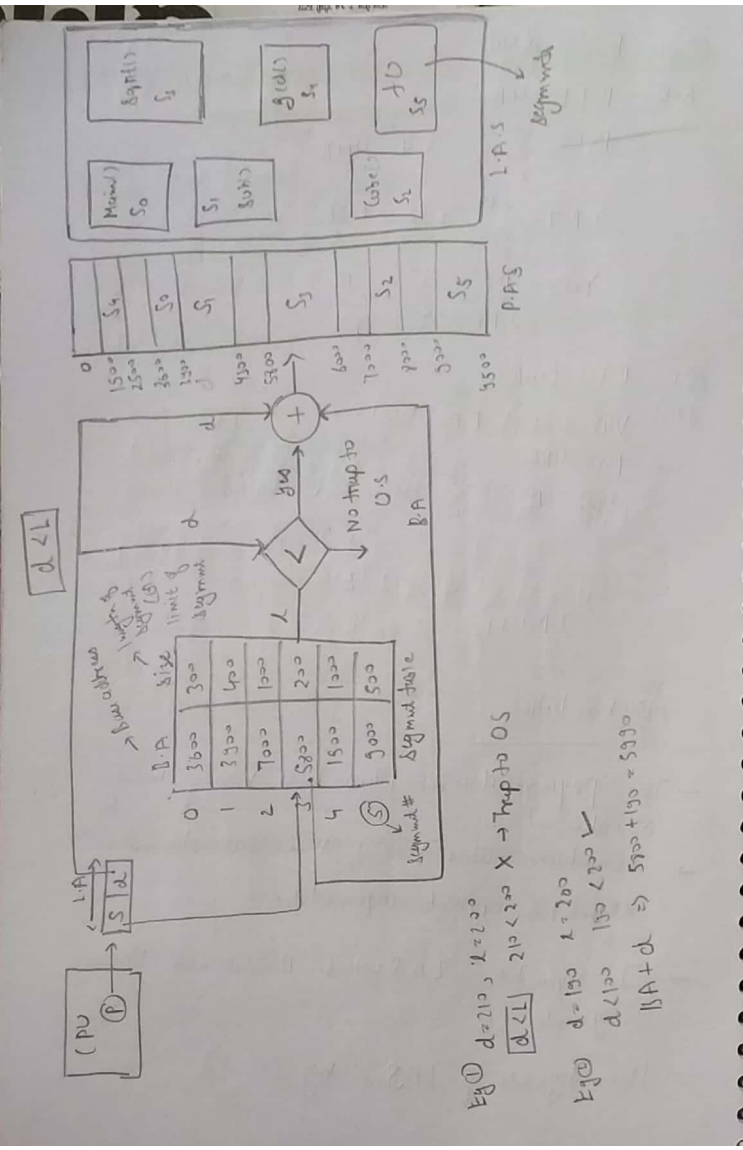
Segmentation

→ The paging does not follow users view of memory allocation

→ To achieve users view of memory allocation the segmentation will be implemented

→ In segmentation, L.A.S will be divided into various segments

→ The segments of L.A.S will vary in size.



Eg① $d = 210$, $L = 230$

$210 < 230$ ✗ → Trap to OS

$d < L$

Eg② $d = 130$ $L = 200$

$130 < 200$ ✓

$BA + d \Rightarrow 5900 + 130 = 5930$

→ 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 no.

d = no. of bits required to represent segment size

(or)

Word no. of the segment.

(or)

Segment offset.

→ No. of entries in the segment table is same as no of segments in L.A.S.

NOTE: Variable size segments are brought from L.A.S to P.A.S so it is similarly behaving like Variable partition scheme. Hence Segmentation still suffers from External Fragmentation

Q24 P-132 (WS)

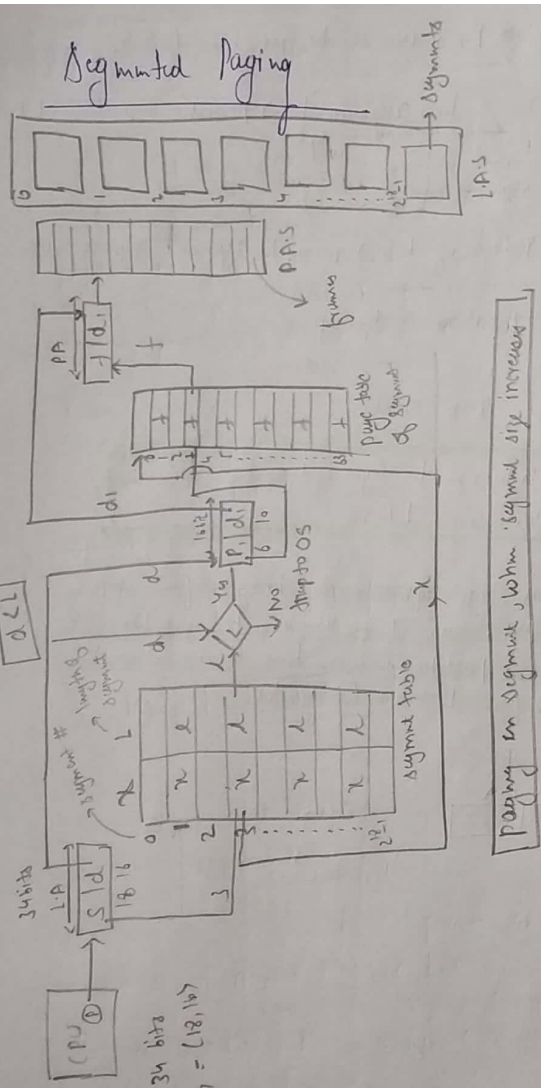
Sol: L.A

S	222

$222 < 302$

$498 + 222 = 720$

Segmented Paging

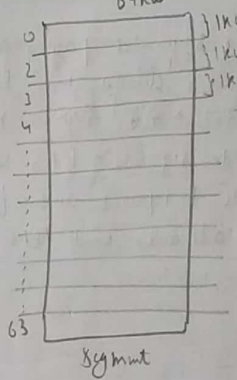


Paging on segment, when segment size increases

λ = No. entries of PT of respective segment

Page size of segment = 1KW

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



# of pages in segment = $\frac{64KW}{1KW} = 64$

→ 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 entries in P.T of segment is same as no. of pages in a segment.

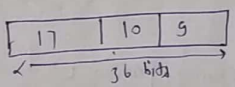
p_1 = no of bits required to represent pages of segment (or) page # of segment

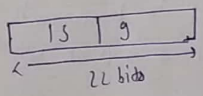
d_1 = no of bits required to represent page size of segment (or) word # of page of segment (or) page offset of segment.

→ Page size of segment is same as frame size of PA-S

Q Consider a system using segmented paging architecture. The segment is divided into 1k pages & each page is having 512 entries. Segment no. require 17 bits to represent all the segments of L.A.S. And frame no requires 13 bits to represent all the frames of PA-S. Memory is word addressable and P.T.E size is 2W. Then calculate

i) Length of L.A
ii) Length of P.A
iii) P.T size of segment.

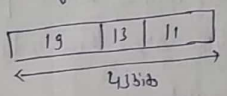
Sol. i) L.A = 

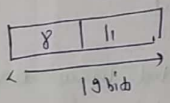
ii) P.A = 

iii) P.T size = $2^{10} * 2W$
= 2kW

Q Consider a system using segmented paging architecture. The segment is divided into 8K pages & 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 512 MB. And P.T.E size size is 8 bits. Then calculate

i) Length of logic L.A
ii) Length of P.A
iii) P.T size of segment
iv) no of frames in P.A.S

Sol. i) 

ii) 

iii) P.T size = $2^{13} \times 8$bit = 8kB

iv) # frames = $2^8 = 256$

frames = $\frac{2^{19}}{2^{11}} = 2^8$

Q2? P-1?2(WB)

Sol. P.A.S = L.A.S = 2^{16}B

# of segments = 8

Let # of pages be = $\frac{2^{16}}{x}$

P.T.E size = 2B

P.T size = $\frac{2^{16}}{x} * 2 \leq x$

$2^{17} \leq x^2$

$i^7 \leq x$

128 B ≤ x

8. segment size = $\frac{2^{16}}{8} = \frac{2^{16}}{2^3} = 2^{13} = 8K$

Let page size be x

20/9/18

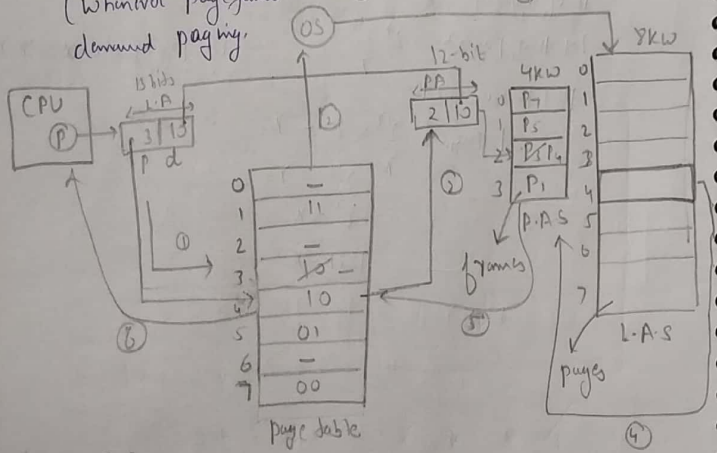
Virtual Memory

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

Virtual memory is implemented using demand paging (or) demand segmentation

Demand Paging

Loading the page into memory on demand (whenever page fault occurs) is called as demand paging.



1) LA > PA
2) (P) ⇒ L.A

Important Steps

① → The CPU is trying to access the page in the main memory, but the page is currently not available in the P.A.S. (This is called as page fault)

② → The prg. execn will stop and the signal will be sent to the O.S regarding the page fault

③ → The O.S will search for the required page in the L.A.S (Virtual memory)

④ → The required page will be brought from L.A.S to P.A.S

4.1 → The page replacement algorithm will be used for the decission making of selecting the page for the replacement

⑤ Respective P.T.E will be updated accordingly

⑥ The signal will be sent to the CPU to continue prg execn, & CPU refer/relax's the required page in the M.M & continue the prg execn.

Page fault Service Time

→ The time taken to service the page fault is called as page fault service time.

→ Page fault service time includes, the time to perform all the 6 steps above.

→ page fault service time = 'S' & M.M.A.T = 'm'
page fault rate = "p"
Thm the formula for E.M.A.T.

EMAT = m + p(s)

E.M.A.T = p(S) + (1-p)(m)

NOTE: P.T access time is neglected bcz S >>> m (S: ms, m: ns)

Q Consider a system which has a M.M.A.T = 35 ns & P.F service time is 175 ns and page hit ratio is 75%. Thm what is EMAT?

Sol: E.M.A.T = 0.75 (35) + (0.25) (175)
= 26.25 + 43.75
= 70 ns

Q.20 P-107

Sol:
Thm exec = i
P.F.S.T = 'j'
page fault rate = k
= $\frac{1}{k} \times j + i$

$\frac{1}{k} \times j + (1-\frac{1}{k})i$
$\frac{j-i}{k} + i$
$\because j >>> i$
$\frac{j}{k} + i$

P.F.S.T = i + j
P.F. rate = $\frac{1}{k}$

$\frac{(i+j)}{k} + (1-\frac{1}{k})i = \frac{i}{k} + \frac{j}{k} + i - \frac{i}{k}$
$= i + \frac{j}{k}$

Q37 P-109

Sol: P.S.T = 10ms P.F. rate = $\frac{1}{10^6}$
m = 20ns

E.M.A.T = $\frac{1}{10^6}(10ms) + \left(1-\frac{1}{10^6}\right)(20ns)$
$= \frac{10 \times 10^{-3}}{10^6} + 20ns$ $\quad 1 >>> \frac{1}{10^6}$
= 10 ns + 20 ns
= 30 ns

Q13 P-106

Sol: E.M.A.T < 2ms

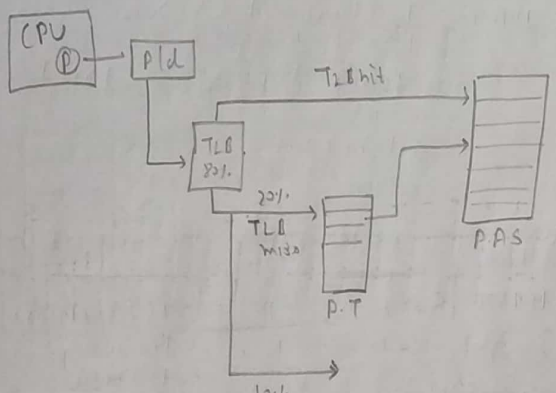
P.S.T = 0.7 (20) + (0.3)(8)
= 14 + 2.4
= 16.4 ms

p(16.4) + (1-p)(1ms) < 2
16.4p + 1 - p < 2
15.4p < 1
p < $\frac{1}{15.4}$
p = 0.064

Q21 P-107

Sol: E.M.A.T = (0.8)(29) + (0.2)(29)
P.F.ST = E.P.F.S.T = 0.1 (200) + (10.9)(10)
= 20 + 9
= 29ms
= (0.8)(29) + (0.2)(29)
= 8 + 5.8 = 13.8

Sol:



P.F.R P.F.T P.F.R

EMAT = 0.80(0+10) + 0.20(0+0.10(200) + 0.90(10))

= 13.8 ms

TLB MR — TLBAT — MMAT

21/9/2018

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

0,1 → page numbers referred by the process in the logical address.

NOTE: (Gate) The no. of frames allocated to the process is decided by instruction set architecture.

First In First Out (F.I.F.O)

Assume 4 frames are allocated to the process

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	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	4	7	7	7	
7	7	7	7	7	3	3	3	3	3	3	3	3	3	2	2	2	2	2	
F	F	F	F	H	F	H	F	H	H	F	H	H	F	F	H	H	F	H	H

Total Pagefault = 10

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

Page faults = 15

Reference string

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

3 frames

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

1	5
2	3
3	4

PF = 9

4 frames

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

1	4
2	5
3	2
4	3

Page faults = 10

Belady's Anomaly (V.V Imp Guys)

By increasing no. of frames to the process, the no of page faults should decrease but instead they are increasing this problem is called as Belady's Anomaly

Optimal page Replacement

In the event of page fault, replace the page which is not used for longest duration of time in the future.

4 frames

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

F F F F F F F F

Total page faults = 8

3 frames

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

F F F F F F F F F

Total page faults = 9

LRU page replacement

In the event of page fault, replace the page which is least recently used. (ie the page which has not been referred for the longest time)

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

4-frames

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

7
0
1
2

Total Page fault = 8

Most Recently Used

In the event of page fault, Replace the page which is most recently used

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	3	0	3	2	2	2	0	0	0	0
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	3	0	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
F	F	F		F		F	F		F	F	F		F					

Page faults = 12

3 frames

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

7
0
1

Total Page fault = 16

Q17 P - 107

Sol. Page size = 100 records

Record #	Page #
0 - 99	0
100 - 199	1
200 - 299	2
300 - 399	3
400 - 499	4
500 - 599	5

Address: 0100, 0200, 0400, 0499, 0510, 0530, 0560, 0120, 0220 0240, 0260, 0320, 0370

Reference string: 1, 2, 4, 5, 1, 2, 3.
F F F F F F F

1

Page fault = 7

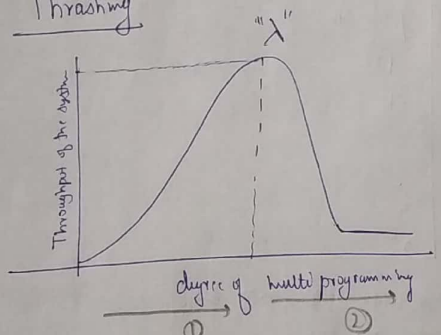
NOTE: ① The reference string will never contain continuous same page.

Ex: 1, 2, 3, 44, 3, 41 ← invalid

1, 2, 3, 4, 3, 2, 1 ← valid

② If there is only 1 page alto frame allocated to the process then every page referred will be a page fault

Thrashing



Ex: No of free frames = 400 (assume to every process equal # of frames need to be distributed)

① [# of process = 100
Each process Pi gets = 4 frames

② [# of process = 400
Each process Pi gets = 1 frame

Case ①

→ In the initial degree of multi programming upto the point of "λ" the CPU utilization, throughput of the system are very high and system resources are utilized perfectly.

Case ②

→ If we further increase the degree of multi programming the CPU utilization, throughput of the system will drastically fall down and system will spend more time only in the page replacement. This situation in the system is called as thrashing. and bcz of this the time taken to complete the exec of process increases.

Causes of Thrashing

1. High degree of multi programming
2. Lack of frames (main memory)

Recovery of Thrashing

1. Do not allow the system to go into Thrashing and instruct the long term scheduler 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 they can recover the system from thrashing

(Imp)

Q. Consider a system with following parameters

System is in thrashing {
- CPU Utilization is 10%.
- Paging Disk is 90%. → time spent on page replacement

Then which of the following factors will improve the CPU utilization

i) Install faster CPU. → No
ii) Install bigger disk : → No
iii) Increase degree of multiprogramming → No already Thrashing
iv) Decrease degree of multiprogramming → Yes
v) Install more main memory → Yes
vi) Decrease page size → No (# of pages required by process will increase)
vii) Increase page size → likely (# of pages will decrease)

NOTE: The main purpose of Virtual memory is, by allocating less memory to the process we are trying to execute large size process and more no. of processes, indirectly we are trying to increase degree of multiprogramming

Gate

→ If the Virtual memory is removed from the system then efficient implementation of multiprogramming & multi user is not possible

File Systems

1) File :- Collection of logically related entities.

2) Attributes :-
   1) Name
   2) Type
   3) Size
   4) location
   5) Owner
   6) Creation date
   7) Last modified date
   8) Permission
   9) Password

→ All these attributes of the file are called as file context

→ file context will be stored in the F.C.B (file control block)

3) Types :-
   1) .doc
   2) .txt
   3) .exe
   4) .class
   5) .bat
   6) .dll
   7) .pdf
   8) .png

10) .xls / .xlsx

11) .c / .cpp / .java / .php / .js / .html / .py

12) .mps / .mp4

Various Operation Performed on file

1) Create
2) Open
3) Write
4) Read
5) Truncate
6) Append
7) Save
8) Save as
9) Close
10) delete
11) Rename
12) Cut / copy / paste
13) hide
14) Print

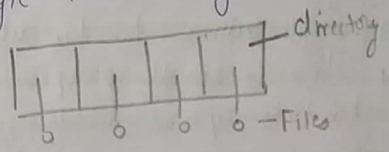
Access Methods :-

1) Sequential Access
2) Random Access

NOTE: For the better classification of files, they will be stored in the directory

Directory Structure

1) Single level directory Structure

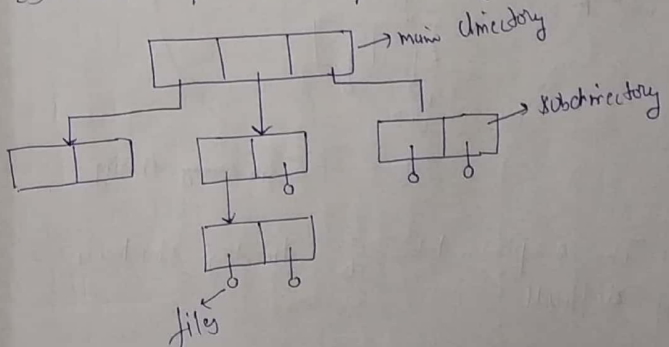


→ implementation of single level directory structure is simple / easy, searching is easy but we have to search only in single directory.

→ Two files can not have same names

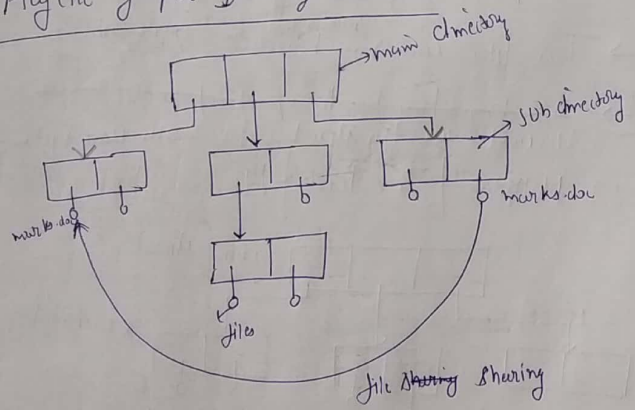
Disadv → Searching time for the specific file will be more.
We can't use this structure for multi-user system.

2) Multilevel / Tree level / Hierarchy level



1) Implementation of directory structure is difficult.
2) Better classification of the files as per the criteria
3) Searching time for the files will be less.
disadv
4) If the same file exists in the two different directories then if one file is updated then other file has to be updated accordingly otherwise there will be a inconsistency.

Acyclic graph Directory Structure



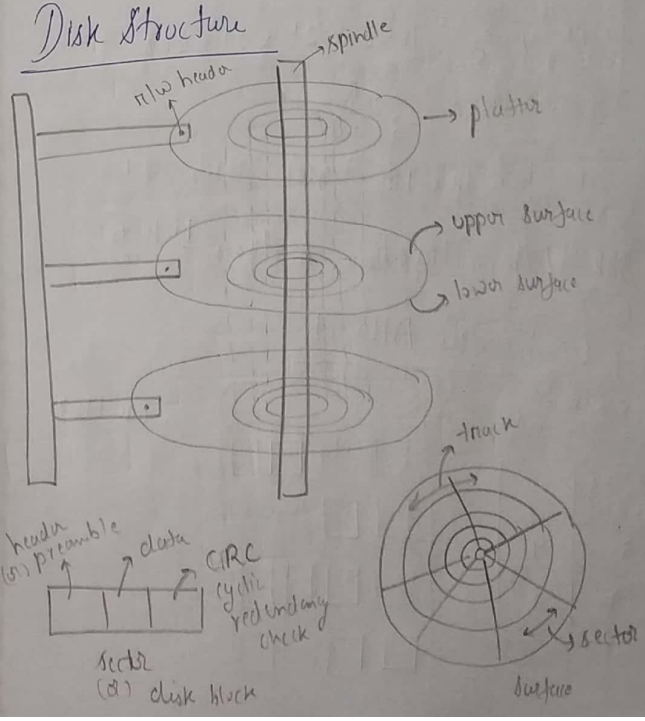
→ The implementation of directory structure is difficult

→ The better classification of files as per the criteria.

→ Searching time for the files will be less

advantage
→ If the same file exists in the 2 different directories then if one file is updated, then other file will be updated automatically by using file sharing concepts.

Disk Structure



Q Consider a disk which has 16 platters and each platter is having 2 surfaces. And every surface is divided into 1K tracks. And every track is further divided into 512 sectors and each sector can store 2 KB data. Then Calculate ~~the~~

i) Capacity of the disk

ii) how many bits are required to identify specific sector of the disk

Sol i) capacity = $16 \times 2 \times 2^{10} \times 2^9 \times 2KB$

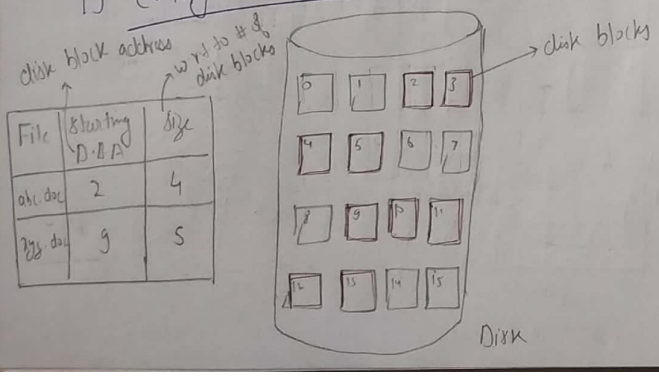
$= 2^{35} B$

$= 32 GB$

ii) bits for sector = ~~9 bits~~. ~~35 bits~~ 24 bits

Disk Space Allocation ~~Methods~~ Methods

i) Contiguous Allocation



→ In the contiguous allocation, disk blocks are allocated in a continuous manner

→ Every file is associated with 2 parameters

i) Starting D.B.A

ii) Size.

→ Increasing the file size may not be possible always

→ It is suffering from external fragmentation.

→ Internal fragmentation may exists in the last disk block of the file.

→ It supports, both sequential & random access of the file.

Starting D.B.A + Offset = Random Access

Ex: 9 + 4 = 13

ii) Linked Allocation / Non-Contiguous Allocation

file	Starting D.B.A	Ending D.B.A
abc.doc	2	1
xyz.doc	9	10

→ In a linked allocation disk blocks are allocated in a non-contiguous manner.

→ Every file is associated with 2 parameters.

i) Starting D.B.A
ii) Ending D.B.A

→ Increasing file size is always possible if the free disk block is available

→ There is no external fragmentation.

→ The internal fragmentation may exists in the last disk block of the file.

Disadv

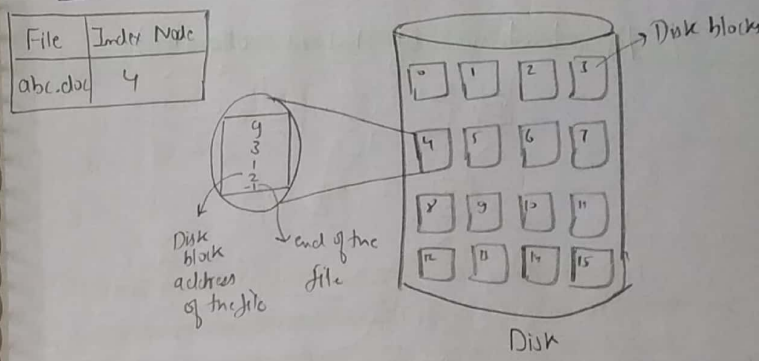
→ There is overhead of maintaining the pointer in every disk block.

→ If the pointer of any disk block is lost then the file will be truncated.

→ It ~~support~~ supports only sequential access of the file

iii) Indexed Allocation

File	Index Node
abc.doc	4



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

→ Index node contains all the disk block addresses of the file.

→ There is no external fragmentation.

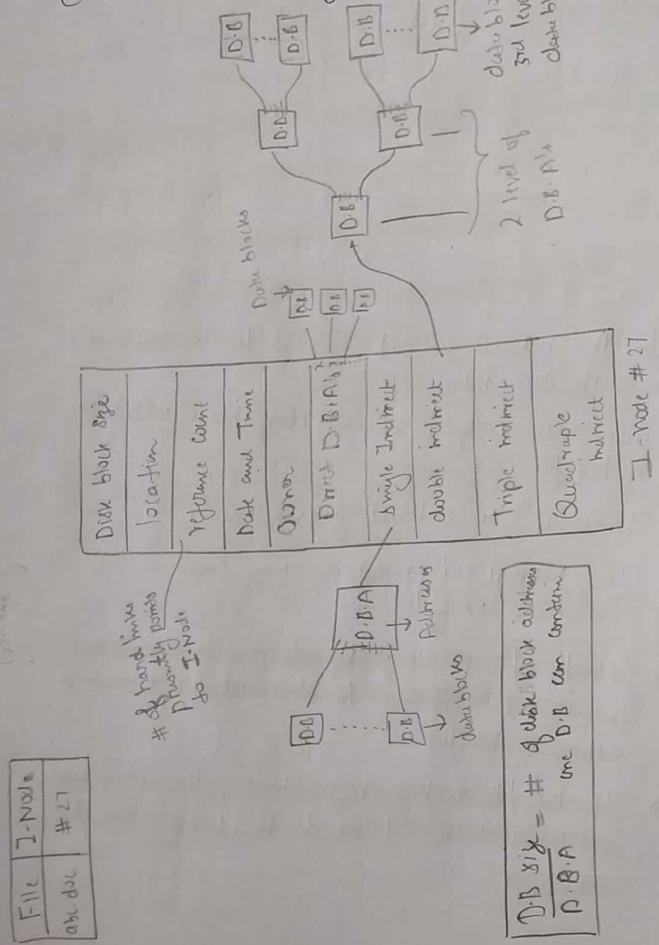
→ The internal fragmentation may exists in the last disk block of the file.

→ If the file is very very large, then 1 disk block may not be sufficient to store all the disk block address of the file.

→ If the file is very very small, then it is waste of using entire 1 disk block, just to store the addresses.

Unix - I-Node Implementation

(An Extension of indexed allocation)



Total size of file system = [# of Direct D.B.A's + (D.B.Size / D.B.A) + (D.B.Size / D.B.A)^2 + ...] * D.B Size

(# single indirect D.B.A's; # double indirect D.B.A's; multiplied by D.B Size)

→ 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.

→ Whenever the file is created, depending on the size of the file, file will be stored only in particular place, may be in the direct D.B.A's, (or) may be in the single indirect D.B.A's, (or) may be in the double indirect and so on.

Q Consider the Unix - I node which maintains 10 Direct D.B.A's, 1 → single indirect, 1 - double indirect and 1 - triple indirect D.B.A's. The D.B size is 1kB and D.B address requires 32 bits, then what is is maximum file size possible

Sol. maximum file size = $\frac{2^{11}}{4} = 2^9 = (512) * 1kB = 2^{27} \times 2^{4}$

$= 2^{38} = 256\,GB$

$= \left(\frac{2^{10}}{2^2}\right)^3 * 1kB = 2^{24} \times 2^{10} = 2^{34} = 16\,GB$

Q 18 P-114

Sol: $\text{Max file size} = \left(\frac{512}{4}\right)^3 \times 512$

$= (2^{21}) \times 2^9$

$= 2^{30}$

$= 1\text{GB}.$

Q 12 P-114

Sol: D.B = 1KB

$(128)^3 \times 1\text{KB}$

$2^{21} \times 1\text{KB}$

2^{31}B

$= 2\text{GB}$

Q13

Sol: max file size

$= \left(\frac{2^{12}}{2}\right)^2 \times 2^{12}$

$= 2^{22} \times 2^{12}$

$= 2^{34}$

$= 16\text{GB}$

Q7

Sol: $\left(\frac{1\text{KB}}{x}\right)^4 \times 1\text{KB} = 4\text{TB}$

$\frac{2^{40}}{x^4} \times 2^{10} = 2^{42}$

$\frac{2^{50}}{x^4} = 2^{42} \Rightarrow 2^8 = x^4$

$2^8 = x^4$

$2^2 = x$

$x = 4 \text{ bytes}$

$x = 32 \text{ bits}.$

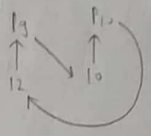
Work book (Discussion) Deadlock

Q14

Sol: b) $n = 21$ (resources) $k = 12$ (processes)

$P_9 \Rightarrow R_{10}^{(2)}, R_{12}^{(1)}$

$P_{10} \Rightarrow R_{10}^{(1)}, R_{12}^{(2)}$



deadlock occur.

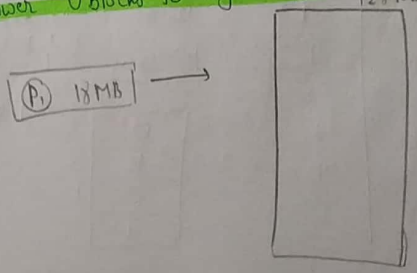
Buddy System

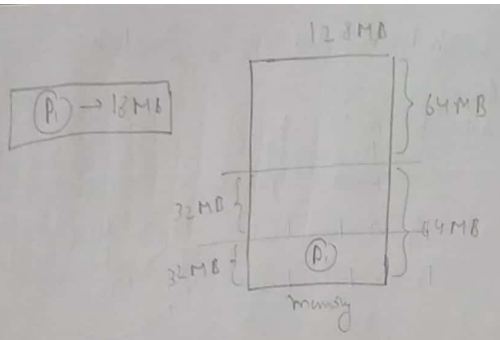
→ In the buddy system initially memory will be single continuous free block.

→ Whenever the request by the process arrives memory will be divided into two half blocks

→ If the request by the process is too small, then the lower block of the memory is further divided into 2 half blocks again.

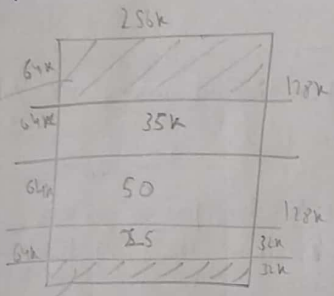
→ In the buddy system memory will be allocated from lower blocks to higher blocks





Q4 · P-100

Ans.



free blocks

Q 28 -108

Page size = 16B

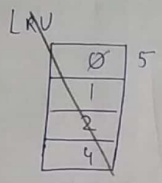
Byte #	Page #
0 - 15	0
16 - 31	1
32 - 47	2
48 - 63	3
64 - 79	4
80 - 95	5

addresses: 0, 4, 8, 20, 24, 36, 44, 12, 68, 72, 80, 84, 28, 32, 88, 92

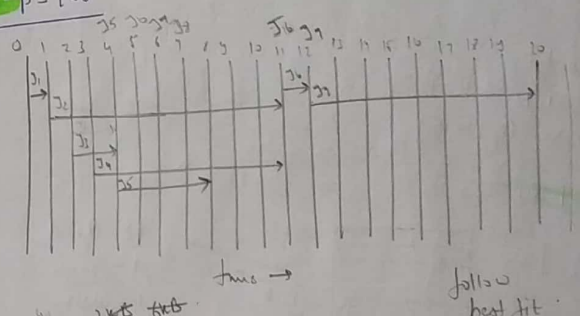
	F	F	F		F	F	F
Req string:	0	1	2	0	4	5	1	2	5

Page faults = 7

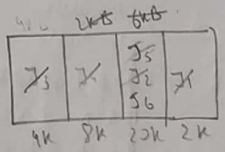
0	2
X	5
2	1
4	



Q 30 P-108



follow best fit.

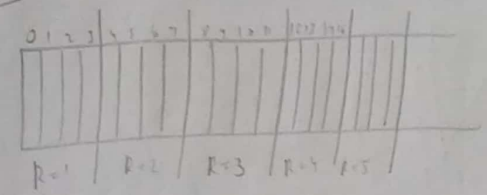


t0	t1	t2	t3	t4	t5	t6	t7
J1	J2	J3	J4	J5	J6	J7	J8
2K	14K	3K	6K	8K	10K	12K	20K
1	10	2	8	4	1	8	6

Ans: 20 (J7 is completed at t = 20

Q5 P-113

Sol:

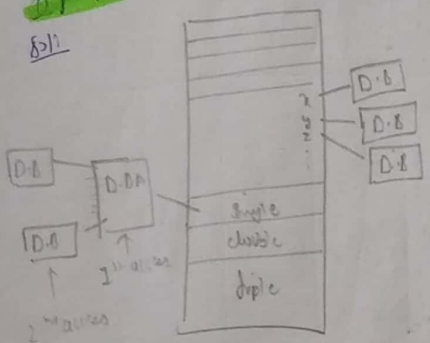


M=4B

First byte of Record N
location = (n-1)*m

Q7 P-119

Sol:



12 - direct
1 - single
1 - double
1 - triple
D.B size = 8kB
D.B. A = 32 bits
Byte # = 1342882 6
Max file size = 16MB

1) max file size possible with direct D.B.A's

12 * 8kB = 96kB
= 96 * 1024
= 98304B

2) max file size possible with ~~triple~~ single indirect

$\left(\frac{8kB}{32bit}\right) * 8kB = 16777216B$

NOTE: for I-Node also disk access required, but it is given that it is present in Main Problem

Q2.18 (GB) P-483

Imp Q24 - P-92

Sol:

$S_1 = 0$
$S_2 = 0$

P_2	P_1
USE R_1	
Signal(S_1)	
	wait(S_1);
	USE R_1
	USE R_2
	Signal(S_2);
wait(S_2);	
USE R_2	
USE R_3	
Signal(S_1)	
	wait(S_1);
	USE R_3
	USE R_4
	Signal(S_2);
wait(S_2);	
USE R_4	

Q6 P-81

NOTE:

If (context switching is disabled then process is not allowed for preemption

Disk Free Space Management

Size of disk = 20 MB

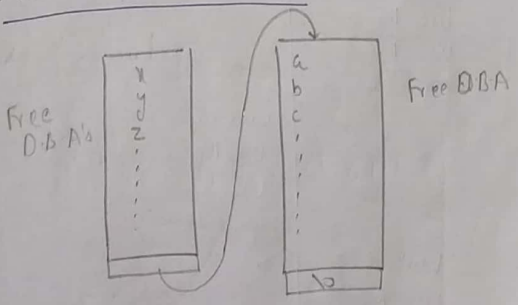
disk block size = 1KB

D.B.A = 2B

# of disk blocks available in the given disk

$= \frac{\text{size of disk}}{\text{disk block size}} = \frac{20MB}{1KB} = 20k$

1) Free disk list Approach :-



In this approach since disk blocks are used just to store the free disk block addresses.

# of disk D.B.A's possible to store in 1 disk block

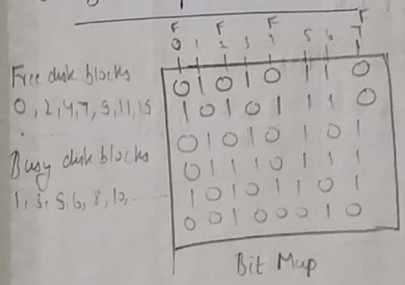
$= \frac{\text{disk block size}}{\text{D.B.A}} = \frac{1KB}{2B} = 512$

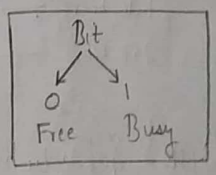
512 D.B.A we can store in 1 D.B

∴ to store 20k D.B.A's, # of D.B required

$= \frac{20k}{512} = 20 \times 2 = 40$ disk block required to store the addresses

2) Bit Map Approach





→ In this approach every disk block is mapped with 1 binary bit.

1) # of disk blocks we can map in 1 disk block = D.B size = 1K × 8 bits = 8k bits

2) 8k D.B ⟶ we can Map in 1 D.B

To map 20k D.B ⟶ ?

$\frac{20k}{8k} = \lceil 2.5 \rceil = 3$

Q17 P-114

Sol: H.D capacity = 40MB

D.B size = 2kB

D.B.A = 3 bytes

D.B in H.D = $\frac{40MB}{2kB}$ = 20k

D.B range we can map in 1 disk block

= 2k * 8 = 16k bits

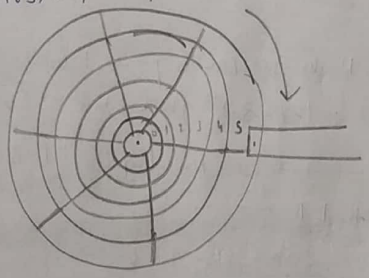
16k bits ⟶ 1 D.B

20k bits ⟶ ? $\frac{20k}{16k} = \lceil 1.25 \rceil = 2$

Disk Scheduling

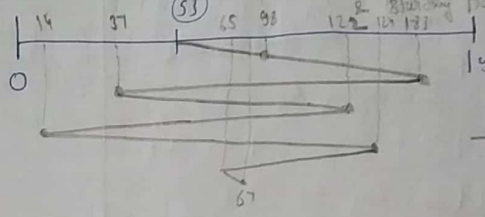
Track Requests

98, 183, 37, 122, 14, 124, 65, 67



First Come First Serve (F.C.F.S)

assume we have 200 tracks & Starting point = 53



→ No starvation

Total track movements made by H/w header i.e. seek time

= ~~Size~~ (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

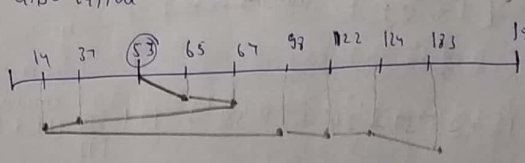
= 640

Shortest Seek time first

also called (Nearest track Next)

→ starvation possible

→ switching direction slows things down

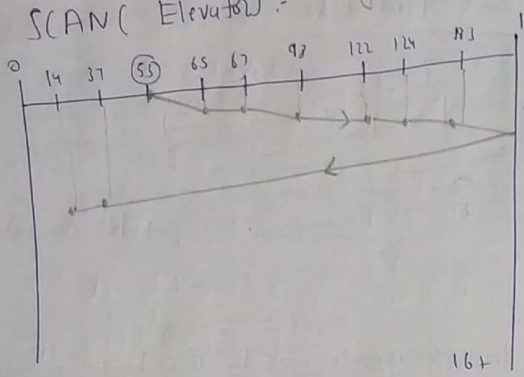


Seek time = 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236

SCAN (Elevator) :-

Initially at 53 moving towards right side



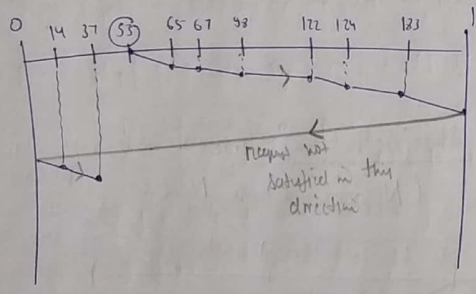
Note
If direction not specified move towards nearest end

16+

Seek time = 12 + 2 + 31 + 24 + 2 + 59 + 162 + 23

= 331

C-SCAN



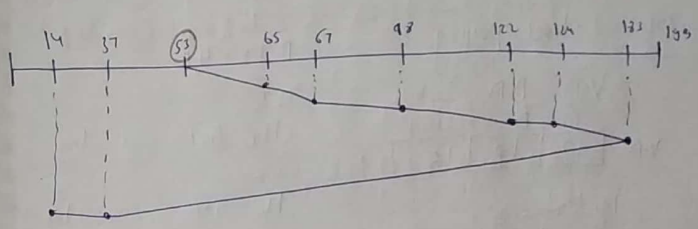
We assume tracks are circular

Seek time = 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 23

= 382

Look

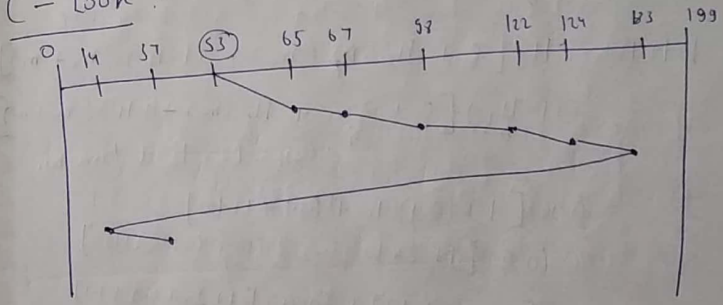
look for the last pending request in the direction of n/w header.



Seek time = 12 + 2 + 31 + 24 + 2 + 59 + 146 + 23

= 299

C - Look :-



Seek time = (183 - 53) + (183 - 14) + (37 - 14)

= 130 + 69 + 23

= 322

Goal: To minimize the avg. seek time of the disk

Important

Q 4.35 P-506

Sol: 2-level paging

P.T.E = 4B

VA = PA = 32 bits

	P1	P2	offset
VA	10	10	12

TLB
H_T = 96% C = 1ns

Cache
H_C = 90% m_C = 1ns

Main memory access time m = 10ns.

Concept: VA ⟶ PA + access the word

Cache contains word/byte. To access word we will refer cache by referring to main memory.

$EMAT = H_T[C + H_C(m_C) + (1-H_C)(m_C + m)] + (1-H_T)[C + 2m + H_C(m_C) + (1-H_C)(m_C + m)]$

↑ access 2 PT's to get frame no.

$= (0.96)[1 + (0.9)(1) + (0.1)(1+10)] + (0.04)[1 + 2*10 + (0.9)(1) + (0.1)(1+10)]$

$= (0.96)[1 + 0.9 + 1.1] + (0.04)[1 + 20 + 0.9 + 1.1]$

$= (0.96)[3] + (0.04)(23)$

$= 2.88 + 0.92$

$= 3.80$

$= 4$ ns, (approx)

(or)

$EMAT = \underbrace{C + (1-H_T)(2*m)}_{VA \to PA} + \underbrace{m_C + (1-H_C)(m)}_{\text{fetching word}}$

$= 1 + (0.04)(2*10) + 1 + (0.1)(10)$

$= 1 + 0.8 + 1 + 1$

$= 3.8$

$= 4$ ns (approx.)

Q 4.43 P-507

Sol

	total size (kB)	# of segments
P_1	195	4
P_2	254	5
P_3	45	3
P_4	364	8

Page size = 1kB P.T.E size = 4B

max segment size = 256kB S.T.E size = 8B

1) Storage Overhead in Paging

$P_1 \to 195$kB # pages = 195 P.T size = 195 × 4 = 780B ≈ 1kB

$P_2 \to 254$kB # pages = 254 P.T size = 254 × 4 = 1016B ≈ 1kB

$P_3 \to 45$kB # pages = 45 P.T size = 45 × 4 = 180B ≈ 1kB

$P_4 \rightarrow$ 364KB # pages = 364 ∴ P.T size = 364 × 4 = 1456

P.T. size > Page size ways for 2nd level.

Level 2

$$P.T\ size = \left(\frac{1456}{1KB}\right) \times 4 = 16B\ (approx) = \sim 1KB$$

∴ ⊗ Storage Overhead (P)

= 1KB + 1KB + 1KB + 1KB

= 4KB

II) Storage Overhead for Segmentation

$P_1 \rightarrow 4$ # Pages = $\frac{256KB}{1KB}$ = 256 P.T size = 256 × 4 = 1KB

∴ S.T will have 4 entries S.T.E size = 8B

S.T size = 4 × 8 = 32B

$P_2 \rightarrow 5$ S.T size = 5 × 8 = 40B

$P_3 \rightarrow 3$ S.T size = 3 × 8 = 24B

$P_4 \rightarrow 8$ S.T size = 8 × 8 = 64B

Storage Overhead (S)

32B + 40B + 24B + 64B = 160B

III) Storage Overhead in Segmented Paging

$P_1 \rightarrow 4$ segment # p = $\frac{256KB}{1KB}$ = 256 PT size = 256 × 4 = 1KB

max segment size = 256KB Total PT's size = 4 × 1KB = 4KB

S.T size = 4 × 8 = 32B

$P_2 \rightarrow 5$ # p = 256 P.T size = 1KB

S.T size = 5 × 8 = 40B total P.T's size = 5 × 1KB = 5KB

$P_3 \rightarrow 3$

S.T size = 3 × 8 = 24B total P.T's size = 3 × 1KB = 3KB

$P_4 \rightarrow 8$ S.T size = 8 × 8 = 64B

total P.T's size = 8 × 1KB = 8KB

⊗ Storage Overhead (T)

(32B + 4KB) + (40B + 5KB) + (24B + 3KB) + (64B + 8KB)

= 160B + 20KB.

∴ S < P < T

Relation b/w Overheads

Q 60 P-509 (GO)

Sol: Minimum # of page color needed

min # page color bits = set index bits + offset bits - page index bits

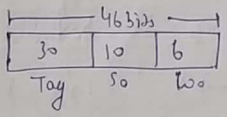
Page size = 8kB (calculated in Q.59)

Cache size = 1MB block size = 64B

16-way set associative

lines = $\frac{1MB}{64B} = 2^{14}$

sets = $\frac{2^{14}}{16} = 2^{10}$



min # page color bits = (10 + 6) - 15 = 3

∴ with 3 color bits we can have 8 page colors.

Q 73 P-511

Sol: Random page replacement algorithm (where a page chosen at random is replaced) suffers from ~~Bel~~ Belady's Anomaly. (True)

bcz pages that are being randomly replaced might follow FIFO policy.

Q 4.37 P-506

Sol: OS is capable of loading and executing a single sequential user process at a time

Hence all the request to disk scheduler will be one by one.

If disk scheduler had known all the disk requests priority, then scheduling algorithm could have optimized the service time. but here requests are one by one therefore there is no use of disk scheduling algorithm.

Any scheduling algorithm in this case will give same ~~service~~ i/p sequence, hence there is no improvement in Eff. I/O.

Q 4.39 P-507

Sol: 2-level paging m = 150ns

Pf service time = 8ms ~~= 8000ns~~

Avg mem time = 100 ns + 2 MA

TLB H_{TLB} = 90% $p_{f.r} = \frac{1}{10,000}$ mem

(EET)avg = ?

(EET)avg = 100ns + 2 MA

MA = (VA ~~*~~ → PA) + accessing word (containing mem)

MA = [C + (1 − H_IB)(2 × m)] + m + p·f·π (P.S-T)

= [0 + (0.1)(2 × 150)] + [150 + 1/10^4 × 8ms/8000]

= (0.1)(300) + [150 + 1/10^4 × 8 × 10^-5]

= 30 + 150 + 800

= 980 ns.

(I.E.T)avg = 100 + 2(980)

= 100 + 1960

= 2060

Correction ⇒ 2060 − 800 = 1260 ns.

Even we need 2 mm. access, for one inst page fault service time need to be included only once.. ~~as for second memory access~~ as for 2nd mm access same inst is being accessed which is deemed to be present in memory on 2nd access.

4.19 P-504

Sol Locality of Reference implies that the page reference being made by a ~~not~~ process is likely to be, to one of the pages used in the last few page references

Locality of reference means, same data value or related storage locations are frequently accessed

Q 5.24 P-523

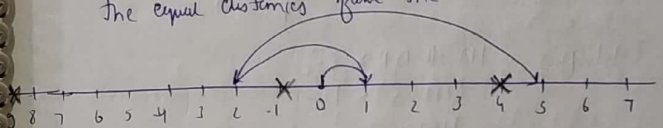
Sol: maximum cardinality of the request set

We need to satisfy 2 conditions

i) Alternating directions with SSTF

ii) Maximize no of requests.

To satisfy i) we should not have 2 requests in the equal distances from the current location.



To satisfy ii) requests must be located as compact as possible. We can place the request next position g/n (cross (x) marked position.

∴ Request seq is 1, 3, 7, 15, ...

$2^1-1, 2^2-1, 2^3-1, 2^4-1, \ldots$

We have 2048 tracks,

Maximum seek length can be $\boxed{2^{11}-1} = 2047$

Q 5.21 P-523

Sol: F1 → 11050B F2 → 4990B F3 → 5170B

F4 → 12640B

for each block book keeping Information take 4B

I-Case

100 B disk block used

F1 → 11050B # D.B = $\frac{11050}{100}$ = 111

Blk I = 111×4 = 444

D.B = $\frac{444}{100}$ = 5

Total space = # DB × Size

= (111+5) × 100

= 11600

F2 → 4990B # D.B = $\frac{4990}{100}$ = 50

Blk I = 200 # D.B = 2

total size = 5200

F3 → 5170B # D.B = 52

Blk I = 208B # D.B = 3

total size = 5500

F4 → 12640 # D.B = 127

Blk I = 508B # D.B = 6

total size = 13300

total space required for storing files

= 11600 + 13300 + 5200 + 5500

= 35600B

II - Case 200 B disk block used

F1 → 11050B # D.B = $\frac{11050}{200}$ = $\frac{552.5}{10}$ = 55.25 = 56

Blk I = 56×4 = 224B # D.B = 2

total size = (58)×200 = 11600B

F2 → 4990 B # D.B = $\frac{4990}{200}$ = $\frac{249.5}{10}$ = 24.95 = 25

Blk I = 25×4 = 100B # D.B = 1

total size = 26×200 = 5200B

F3 → 5170 B # D.B = $\frac{5170}{200}$ = $\frac{258.5}{10}$ = 25.85 = 26

Blk I = 104 B # D = 1

total size = 27×200 = 5400 B

F4 .

F4 → 12640 B # D.B = $\frac{12640}{20 \times 4} = \frac{632}{10} = 64$

BKI = 256 B # D = $\frac{128}{100} = 2$

total size = 66 × 200 = 13200 B

Total space required for storing file

= 11600 + 5200 + 5400 + 13200

= 35400 B

Q 5.35

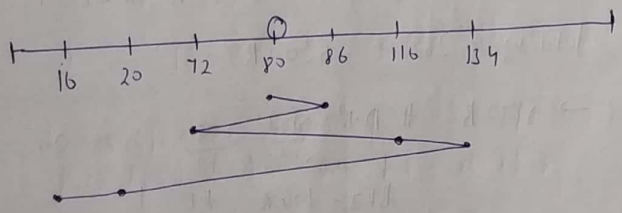
Sol: current → 80 < 100, 80, ✓ not given →

Cylinder req :

[120, 72, 2], [180, 134, 1], [60, 20, 0],

[212, 86, 3], [56, 116, 2], [118, 16, 1]

Cylinder req → 72, 134, 20, 86, 116, 16



total movement = 6 + 14 + 62 + 118

= 200

Power dissipation for 100 cylinder = 20 mW

∴ for moving over 200 cylinder,

power dissipation = 20 mW × $\frac{200}{100}$ = 40 mW

For reversing direction power dissipation = 15 mW

# of times direction reversed = 3

power dissipation = 3 × 15 = 45 mW

Total power dissipation = 40 + 45 = 85 mW.

Q 3.16 P-497

Sol: n resource R_0 R_1 R_2 .... R_{n-1}

k processes P_0 P_1 P_2 .... P_{k-1}

Resource request logic for process P_i

```
if (i%2 == 0) {
    if (i<n) request Ri
    if (i+2 < n) request Ri+2
}
else {
    if (i<n) request Rn-i
    if (i+2 < n) request Rn-i-2
}
```

From resource logic we can clearly see that even numbered processes request even numbered resources.
And they share not more than 1 resource

If odd numbered processes requests only for odd numbered resources, without a cycle then there will be no deadlock.

now we want situation of deadlock
if we go to else part if "i" is odd.
now if n is odd. then

R_{n-i} → will be even

R_{n-i-2} → will be even

Which means only odd numbered processes will be requesting even numbered resources

Hence option a) & c) are eliminated.

If 2 processes requests for same resources R_i & R_j then there will be a deadlock.

Analyzing Option D

n = 41

$P_0 \rightarrow R_0$, $R_0 \rightarrow R_2$

$P_2 \rightarrow R_2$, $R_2 \rightarrow R_4$

$R_4 \rightarrow R_4$, $R_4 \rightarrow R_6$

$P_{16} \rightarrow R_{16}$, $R_{16} \rightarrow R_{18}$

$P_{18} \rightarrow R_{18}$, $R_{18} \rightarrow R_{20}$

k = 19

$P_1 \rightarrow R_{40}$, $R_{40} \rightarrow R_{38}$

$P_3 \rightarrow R_{38}$, $R_{38} \rightarrow R_{36}$

$P_5 \rightarrow R_{36}$, $R_{36} \rightarrow R_{34}$

$P_{15} \rightarrow 26$, $26 \rightarrow 24$

$P_{17} \rightarrow 24$, $24 \rightarrow 22$

Hence no deadlock

Analyzing Option B

n = 21

$P_0 \rightarrow R_0$, $R_0 \rightarrow R_2$

$P_8 \rightarrow R_8$, $R_8 \rightarrow R_{10}$

$P_{10} \rightarrow R_{10}$, $R_{10} \rightarrow R_{12}$

k = 12

$P_1 \rightarrow R_{20}$, $R_{20} \rightarrow R_{18}$

$P_9 \rightarrow R_{12}$, $R_{12} \rightarrow R_{10}$

$P_{11} \rightarrow R_{10}$, $R_{10} \rightarrow R_8$

Here we have deadlock

Q 2.36 P-488

Sol:-

```
do
{
  c[i] = 1; t[i] = max(t[0], .... t[n-1]) + 1;
  c[i] = 0;
  for every j ≠ i in {0, .... n-1}
  {
    1 while (c[j]);
    2 while (t[j] != 0 && t[j] < t[i]);
  }
  C.R
  t[i] = 0;
  Remainder section
} while (true);
```

t[0] = t[1] = ... t[n-1] = 0

P_0 P_1 P_2 ..... P_{n-1}

Case I assume c[0] = c[1] = ... c[n-1] = 0

P_0:
① c[0] = 1
t[0] = 1
For loop
1
2
C.S → preempt
⑤ t[0] = 0

P_1:
① c[1] = 1
t[1] = 2
For loop
1
2 2
④ C.S
t[1] = 0

P_{n-1}:
③ c[n-1] = 1
t[n-1] = 1
For loop
1
C.S
⑥ t[n-1] = 0

Case II

P_2:
② c[2] = 1
t[2] = 2
For loop

P_0	P_1	P_2	P_{n-1}
1	1	1	1
2	2	2	2

C.S

Hence there is M.E

option a) is right answer

Bakery Algorithm (Grade - 2016)

Peterson's soln is best to provide synchronization b/w 2 processes

But if we have more than 2 process ie N > 2 we go for Bakery Algorithm

Simplified bakery Algorithm

# of process → N [0 .... N-1]

P_0 P_1 P_2 .... P_{n-1}

Global shared array num[0 .. n-1]. (token)

Each entry corresponds to a process

ie num[0] corresponds to P[0].

ie num[0] stores token of P[0]

```
lock(i) {
  1 num[i] = MAX(num[0], num[1], .... num[n-1]) + 1,
    for every p ≠ i in {0 ... n-1}
    {
      2 while (num[p] != 0 && num[p] < num[i]);
    }
}
```

this is done way to enter into C.S, it must be atomic

token allocation method

if there is any process with non-zero token and that process token value < current process token value

Critical Section

Unlock(i) { num[i] = 0; } ← reset token to 0

Assume 5 process, all try to enter into C.S simultaneously

Initially	P_1	P_2	P_3	P_4	P_5
num [1..5]	0	0	0	0	0

P_3 first

	P_1	P_2	P_3	P_4	P_5
	0	0	1	0	0

$P_3 \rightarrow$ I / $\rightarrow$ preempt

P_4

	P_1	P_2	P_3	P_4	P_5
	0	0	1	2	0

P_5

	P_1	P_2	P_3	P_4	P_5
	0	0	1	2	3

P_2

	P_1	P_2	P_3	P_4	P_5
	0	4	1	2	3

While (num[p] != 0 && num[p] < num[i])

This condition ensure that process with least non-zero num value enter into C.S ie with least token value

C.S	1st	2nd	3rd	4th
	P_3	P_4	P_5	P_2

ii) num[i] = max(num[0], num[1], ... num[n-1]) + 1

is not atomic ← both execute simultaneously

Order of exit P_3 (P_4 P_5) P_2

P_1	P_2	P_3	P_4	P_5
0	3	1	2	2

Here P_4 & P_5 gets same token, hence they both enter into C.S simultaneously. (NO ME)

Actual Bakery Algorithm

Without atomic assumption

Here we use boolean array of size N.

Choosing [N]

Here each entry corresponds to a process.

Choosing [N] = { False, False, ... , False }
0 1 n-1

Here we have 2 global shared arrays.

Choosing [0... N-1] & num [0..... N-1]

lock() { doorway → token allocation method

Choosing[i] = True;
num[i] = max(num[0], num[1], ... num[n-1]) + 1
Choosing[i] = False;

for every $p \neq i$ loop in $\{0, 1, 2, \ldots n-1\}$

{
While (Choosing[p]);
While (num[p] != 0 && (num[p], p) < (num[i], i));
}
}

this condition ensure that if some process is "choosing" value of "num" then current process will have to wait.

If 2 process gets same token, this condition will resolve problem by allowing lower numbered process to enter into C.S. Hence providing M.E

$$(a,b) < (c,d) \equiv \begin{cases} (a < c) \\ \text{(or)} \\ (a == c) \text{ and } (b < d) \end{cases}$$

Special case ⇒ ∄ [(num[p], p) ≤ (num[i], i)]

bcz of equality there will be deadlock
Hence No progress, No Bounded wait.

Q2.32 P-487

Sol:

```
Process X
private i;
for(i=0; i<n; i++)
{
    a[i] = f(i);
    ExitX(R,S);
}
```

```
Process Y
private i
for(i=0; i<n; i++)
{
    EntryY(R,S);
    b[i] = g(a[i]);
}
```

X & Y are concurrent and array a[] is shared data b/w X & Y.

For synchronization 2 binary semaphore S & R are used both initialized to 0.

Analysis

(a)

```
ExitX(R,S)
1 P(R)
2 V(S)
```

```
EntryY(R,S)
1 P(S)
2 V(R)
```

X → 1 ← suspend wait on R

Y → 1 ← suspend wait on S.

Hence deadlock occurs

Option b) Exit X(R,S)
1 V(R)
2 V(S)

Entry Y(R,S)
1 P(R);
2 P(S);

R = ∅ S = ∅
1 1

X → 1
X → 2

b/c of for loop X can generate all a[i] values without giving any chance to Y.

a[0] a[1], ... a[n-1].

R = X0 S = X0

Y → 1
Y → 2

Y won't be able to complete its remaining b/c X already computed all values of a[i] And there is no other process to make R=1 & S=1.

Hence Y interactions are lost.

option d) Exit X(R,S)
1 V(R)
2 P(S)

Entry (R,S)
1 V(S)
2 P(R);

R=0 S=0

a[0] a[0]
X → 1 R = ∅ 1
X → 2 ← suspend and waits on S.

Y → 1 S = ∅ 1 (& wakeup(-X) i.e X goes from block to ready state
Y → 2 R = X0

b[0]
Y → 1 S = X 1
Y → 2 ← suspend and wait on R.

a[1]
X → 1 R = ∅ 1 (wakeup() Y)
X → 2 S = X0

a[2]
X → 1 R = X 1
X → 2 S=0 suspend and wait on S.

X computed a[0] a[1] a[2], but Y just computed b[0], hence some of the sig nal opn on R is lost, as a result, Y will not be able to complete all its iterations.

Hence Write answer is Option c)

Q2.10 P-481

Sol: If processor does not have any stack pointer register ~~pointer~~, then recursive fun are not possible (ie we can not have subroutine call ins^ns) bcz we can't load activation records in stack. and thus unable to achieve recursion

→ Spooler → Software which create job queue.

→ User level threads can be implemented on an OS which does not support threads bcz user level threads are transparent to kernel.

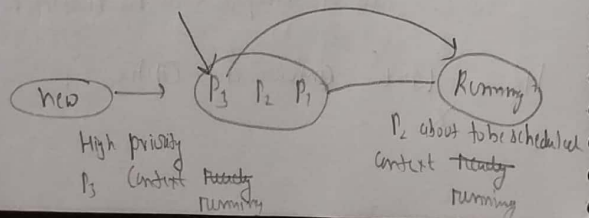
→ For R.R scheduling algorithm we use circular queue

→ ~~FCFS favours~~

→ Dispatch latency means time taken to stop a process and start another process.

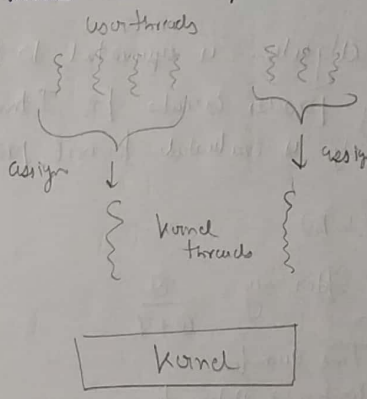
→ Correct sequence of fun

1) Scheduling 2) Dispatching 3) Context saving



→ S/w interrupts are required to get system service which is needed to execute privileged instruction.

→ User level threads are transparent to kernel



Kernel will just see 2 kernel threads asking for execn. It won't be able to see User threads

→ To change mode

non-privileged ⟶ privileged (S/w interrupt required)

privileged ⟶ non-privileged (no S/w interrupt required)
We just need non-privileged instn

→ Kernel threads can share code segment

Q1.51 - P.471

Sol: Scheduler recvaluates the process priorities every T time units, and decides next process to schedule

This algorithm is equivalent to R.R, bcz every process executes for T time, then priority is evaluated for next process.

Q1.1 P-79 (WB)

Sol: CPU efficiency = $\frac{Q}{Q+T}$

Q → Time quantum
T → Context switching time
R → I/O wait Time

Concept :- R is unpredictable.

We can have I/O in 3 ways
1) I/O may occur alone
2) I/O may occur along with CPU
3) multiple I/O may occur simultaneously.