OPERATING SYSTEMS

Handwritten Notes

VIVEKANAND VERNEKAR

GateWallah Parakram C 2024

(Hinglish Weekday)

Other Notes Uploaded Below

https://t.me/pwgatenotes

Introduction

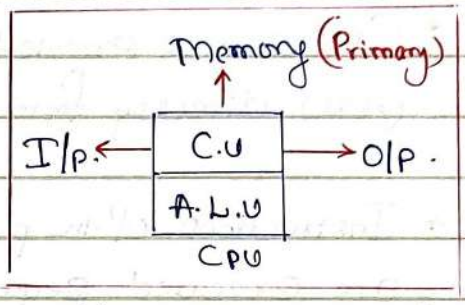
What is an Operating System?

* Is an interface between user and the hardware
* Resource managers
* Set of utilities to simplify the application development
* Set of program(s) / Software
* Operating System acts like a government.

User/Programmers ⟷ OS ⟷ Hardware.

1. Control unit (C.U): It is circuitry that directs operations within a computer's processor.
   * Timing / control signals
   * Sequencing / execution of micro-operations.

   (operations that are carried out on the data, stored in registers (CPU))



VON-NEUMANN Architecture.

2. Arithmetic Logic Unit (ALU): It is a part of CPU that carries out arithmetic and logic operations on the Operands in computer instruction words

Memory:

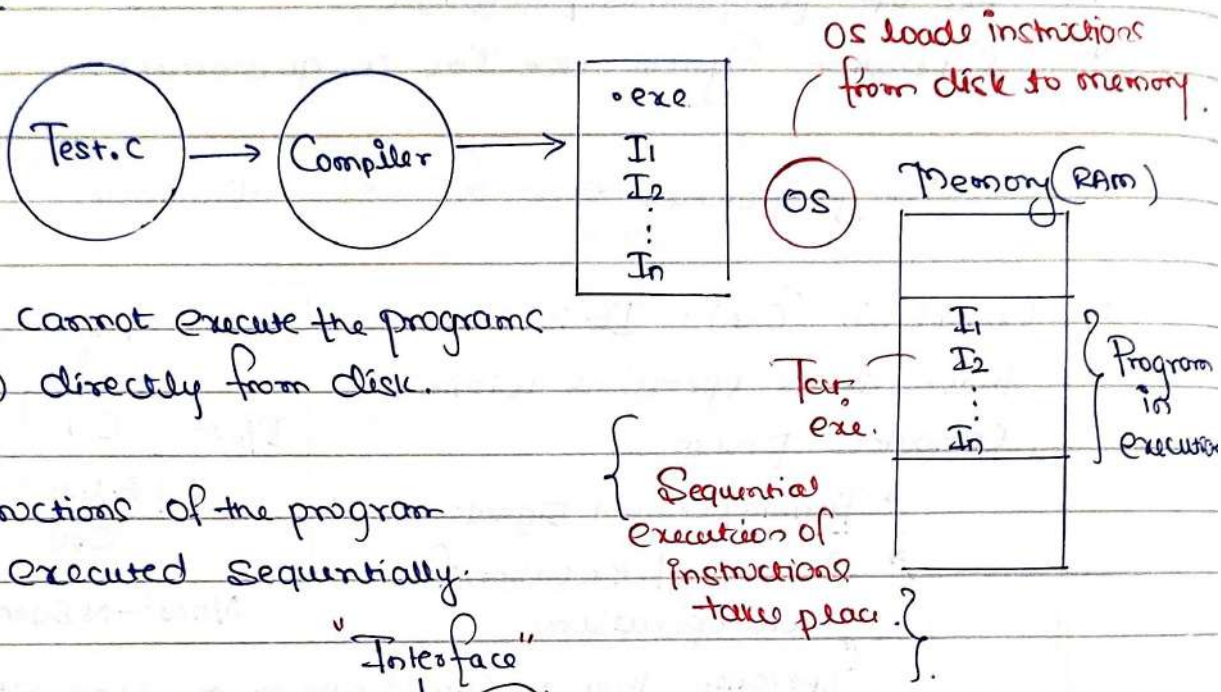
- Primary (also called "Main memory"). eg: Ram, Rom, Cache, registers.
- Secondary ("Auxilary memory"). eg: harddisks, pendrives etc.

Primary memory	Secondary memory
* Volatile (not Rom)	* Non volatile
* faster to access $\times 10^{-9}$ (ns)	* Slower to access (ms) $\times 10^{-3}$.
* Smaller Size	* larger Size.
* more expensive	* less expensive

Note: as per Von Neumann architecture, all Secondary storage devices are part of Ip/op devices.

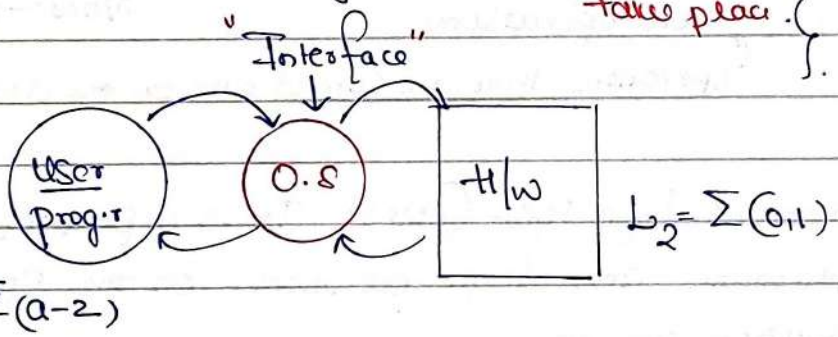
Q How are programs executed in Von Neumann architecture?

→ The programs are executed by the CPU by Stored program concept



* CPU cannot execute the programs (.exe) directly from disk.

* Instructions of the program are executed sequentially.

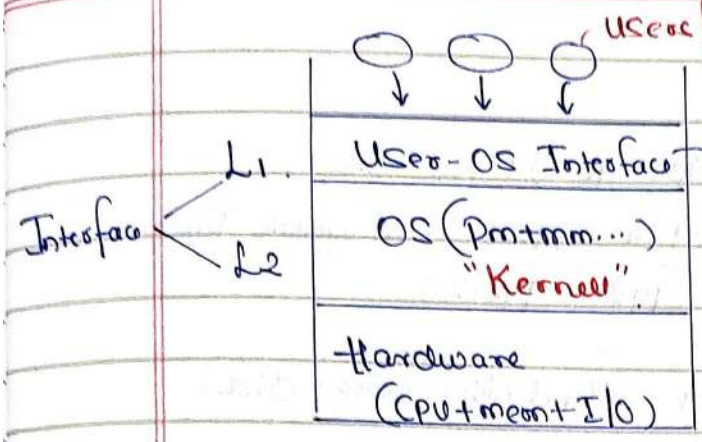


Modules of OS:

- Process manager (CPU)
- Memory manager (memory)
- File manager } Devices
- Devices manager
- Protection manager (Resources)

} Collection is called "Kernel / Nucleus"

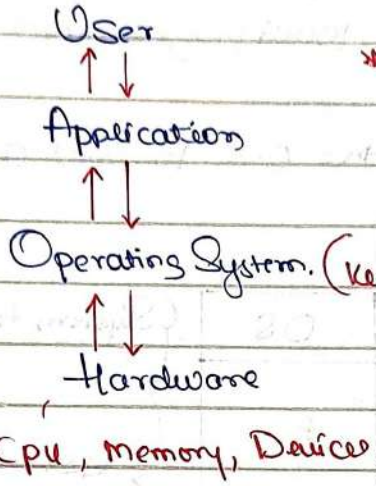
Kernel is a central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time.



Functions of OS:

1. Process management
2. Memory mgmt.
3. Error detection
4. Security
5. File Management.

Functions and Goals:



* function is something that is carried out to achieve the goals.

1. Convinience (ease of use)
2. Efficiency
3. Reliability
4. Robustness
5. Scalability
6. Portability

* Primary (main) goal of an Operating System: Convinience (for personal computing)
* Convinience is not primary goal of all other computing environment

Other Computing environment:

(a) Real time Systems (RTS): + Operate in real time (fixed deadlines)
↓ + Ex: missile, Satellite, nuclear System
main goal: efficiency
control control
← {Hard - Real time System} →

(b) Mobile Operating System main goal: battery efficiency.

Tim Types of OS:

1. Batch OS
2. Multiprogrammed OS
3. Time Sharing OS
4. Real time OS
5. Network OS
6. Distributed OS

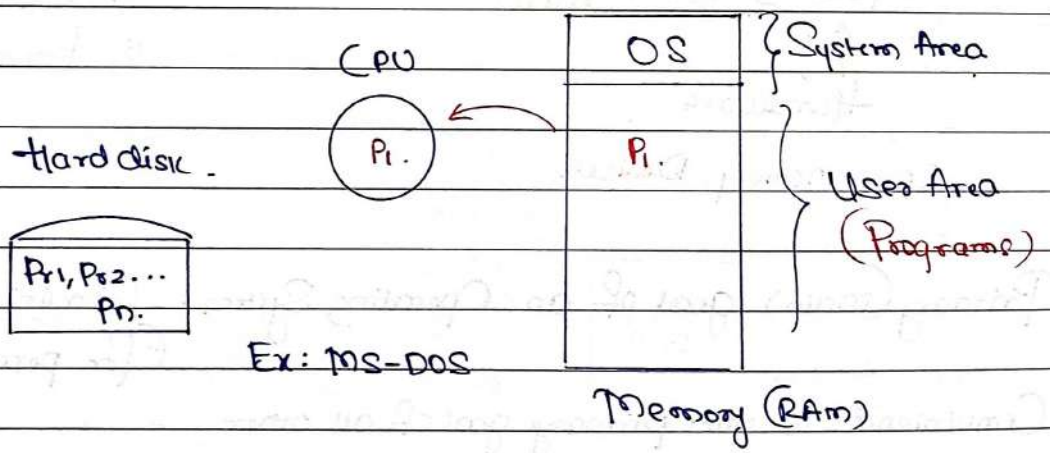
} Advanced

Generations of Computers:

1st Generation of Computers (1930-40s): No O.S
2nd Generation of Computers (1940-1950's): magnetic Tapes
3rd Generation of Computers (1950 - 1960's)
↓
Magnetic Disk (Hard disk, floppy disk)

↙ Uni-programmed ↘ Multiprogrammed

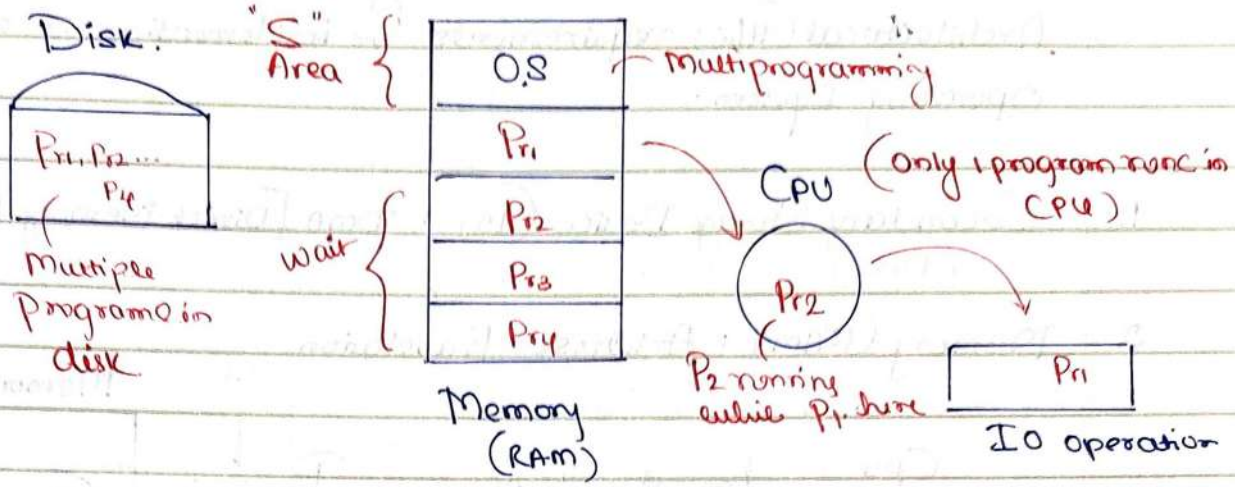
Uni-programming: Ability of the O.S to hold a single program in memory



Drawback:
* Single program in memory is not enough (sufficient) to keep the CPU always busy.
* Throughput and efficiency will be less.
↑ no of programs completed / Unit time.

Multiprogramming: Ability of the operating system to hold (manage) multiple ready to run programs in memory.

Objective: * Maximize CPU utilization
* Maximize throughput.

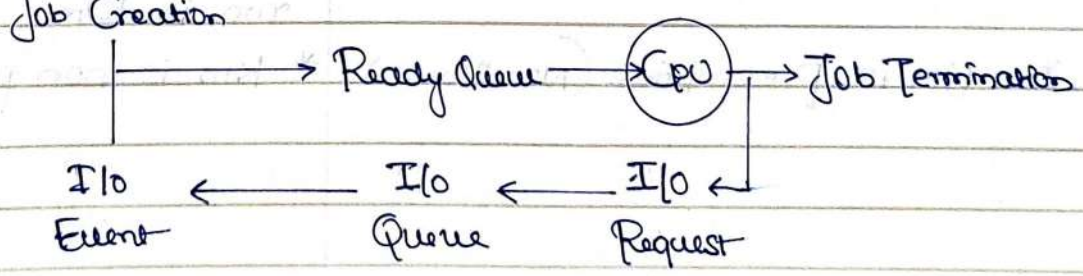


Multiprogramming: Multiplexing of CPU among different programs in memory.
(another definition)

Types of multiprogramming:

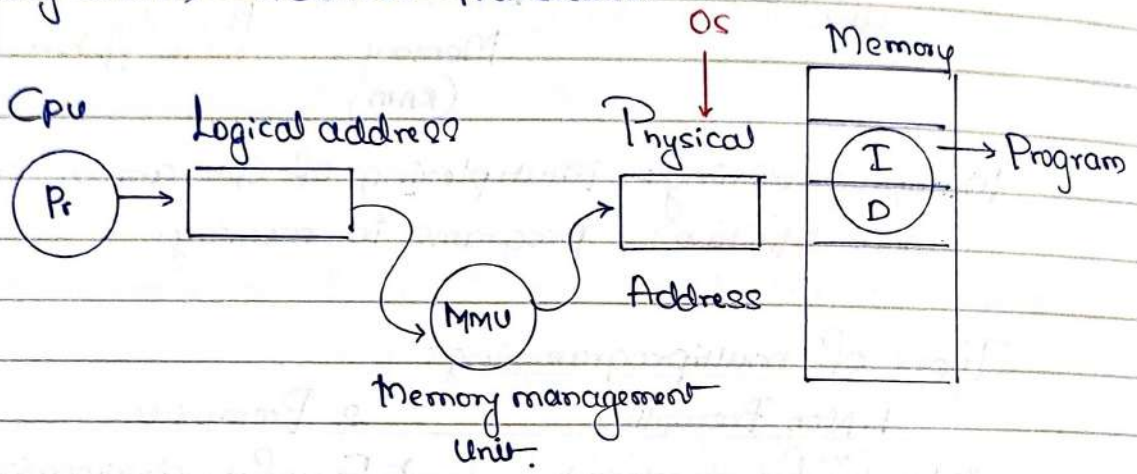
1. Non Premptive	2. Premptive
* No forceful deallocation * Running program on CPU will not be forced to leave CPU. It will release CPU voluntarily: 1. Completed execution 2. needs I/O syst. call. * Drawback: leads to starvation to other waiting programs. - lack of interactiveness, responsiveness.	* Forceful deallocation. * Program running on CPU, can get forcefully deallocated from CPU (Pre Emption), So that waiting programs can get their chance to run on CPU Objective: * improve interactiveness * increase responsiveness Premption is done based on: 1. Priority 2. Time

Multiprogramming (Schematic View)

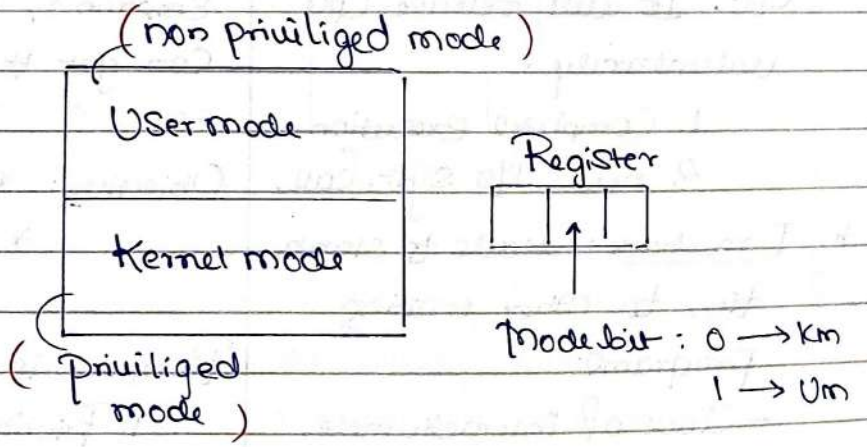


Architectural (H/W) requirements for implementing a multi programmed Operating System :

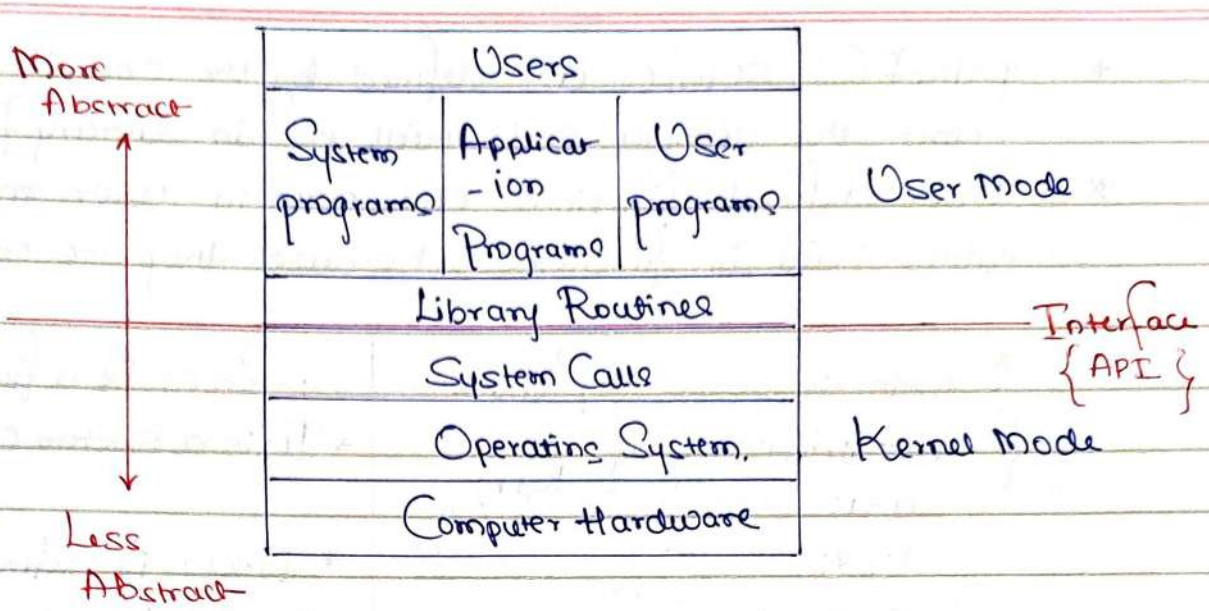
1. Secondary Storage Device (IO) : DMA [Direct Memory Access]
   (Disk)
2. Memory (RAM) : Address Translation



3. CPU (Processor) : Dual mode Operation
   Every CPU of multiprogrammed OS should have two mode of execution.



User mode	Kernel mode
* All user applications run in Um.	* All OS routines programs run in K.m.
* Um is preemptive (Non Atomic)	* Km is non premptive (Atomic)



* Many a times it becomes necessary to shift the mode from user mode and kernel mode and back from kernel mode to user mode.

Mode Shifting: Shifting of mode from user mode to kernel mode and back is called mode shifting.

Mode Shifting (Process) is needed to avail OS services.

API : Application Programmer's Interface / System Call Interface

Implementation of mode shifting process via API/SCI.

*
```
main()
{ int a,b,c;
  b=1;
  c=2;
  a=b+c;
  f(a);
}
```

```
f(int k)
{ k++;
  printf("%d",k);
}
```

predefined fn. - library fun. - built in.

* Implimentation is in library file (.lib)

```
f();
```

'user defined function

* <stdio.h> file contains function prototypes, which is used for type checking

* printf(), scanf() are defined by the compiler implementor and the compile code will be in library file.
* User defined functions are run in user mode along with built in functions because they are written by users.

```
main()                    f(int k)
{ int a,b,c;              { k++; }
  a=1;
  b=2;                    } Um.
  c=a+b;
  f(c);
  fork(); ← Km.
  printf("%d",c);         } Um.
}
```

fork() is a function
* It is a System call } OS routine
* Fork() is defined / implemented in OS Kernel
* Execution of a fork() results in Creating a child process.

```
main()
{ :
  :
  f(); → BSA (priviliged inst.)
  :
  fork(); → SVC
  :
  }.
```

* CT: Compile time
* BSA: branch and save address } Instructions
* SVC: Supervisor Call

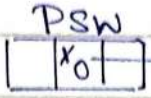
{ All pre defined functions are translated into instructions called BSA (non priviliged instruction) — User mode at runtime. }

{ Software interrupt instructu }.

* fork() is converted into SVC because it is a not user defined — S/w interrupt at runtime.

* Interrupt generated by software instructions are called Software interrupt.
* ISR: Interrupt Servica routine: A program of OS to Servica the interrupt.

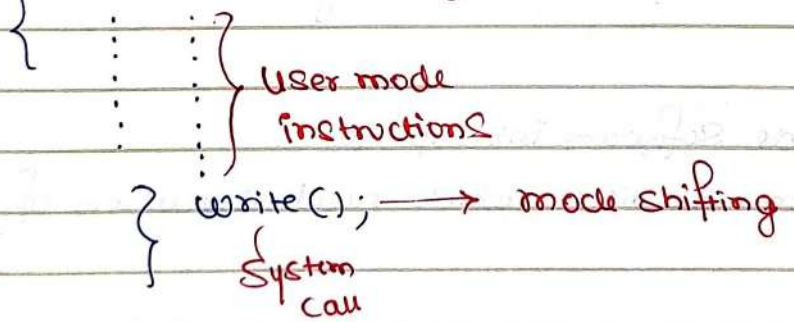
ISR: * ISR will go the register and convert the mode bit from '1' to '0' to shift from user mode to kernel mode and vice versa.



* ISR examines an interrupt and determines how to handle it, executes the handling and then returns a logical interrupt value (returns the address of fork from the dispatch table).

* All the Operating Systems - Windows, Linux, Unix follow the same paradigm
* The fork() will be executed at kernel mode in the dispatch table. All instructions will be executed of the fork and the last instruction will change the mode bit from '0' to '1' after execution and then fork is executed sucessfully at kernel mode and returned to user mode and then remaining code will run.
* To change the mode from kernel mode to user mode then no Software interrupt is required, interrupt is required for changing modes from Um to Km only

printf(): Library file can also use System-call.

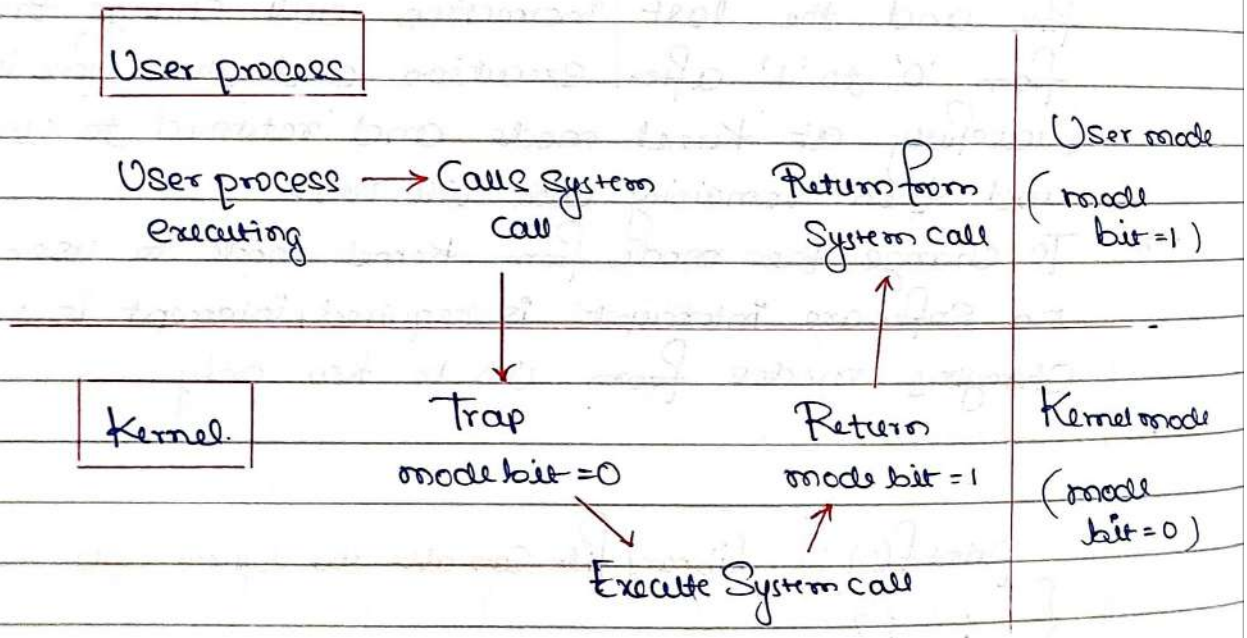


Note: every Priviliged instruction does not generate interrupt

1. Kernel mode: In kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference

any memory address. Kernel mode is generally reserved for lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic, they will halt the entire PC.

2. User mode: In user mode the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.



Q1. A processor needs software interrupt to
: Obtain system service which need execution of priviliged instructions.

Q2 A CPU has 2 modes priviliged & non priviliged. In order to change the mode from priviliged to non priviliged:
: A priviliged instruction (which does not generate an interrupt) is needed.

Process Management.

Q3. System calls are usually invoked by using.
: Software interrupt

Q4. A computer handles several interrupt sources of which of the following are relevant for this question:
* Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
* Interrupt from mouse (raises interrupt is if is moved / button is pressed).
* Interrupt from keyboard (" " key is pressed / released)
* " " Harddisk (" " disk read is completed).

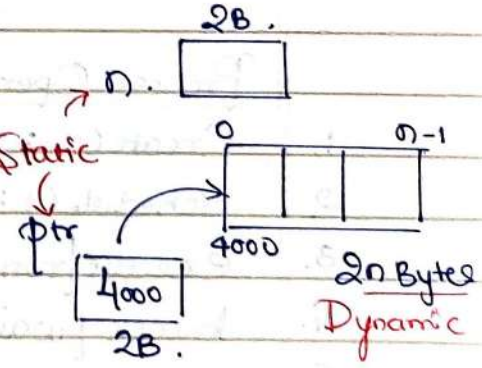
Which one of these will be handled at the highest priority?
: Interrupt from CPU temperature

Program vs Process

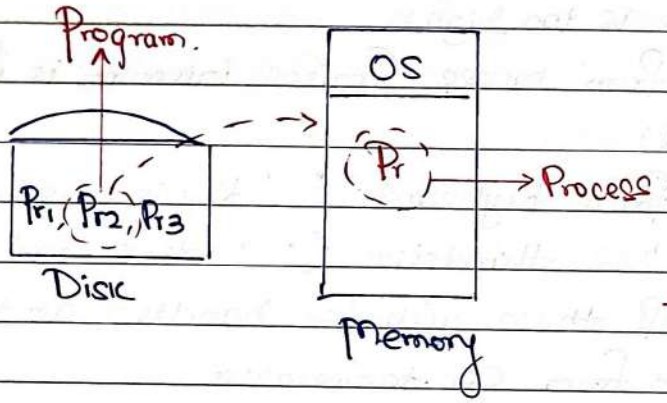
- Program (.exe)
  - Instructions
    - Ex: Load, Store, Add, mov
  - Data (Operands)
    - Static
      - (Fixed / known size)
      - * memory allocated at load time (before RT).
    - Dynamic → * Varying size / fixed size; * memory allocated at run time.
- Process: Program in execution

Static and Dynamic

Ex:
```
int n; int *ptr;
Scanf ("%d", &n);
ptr = (int*) malloc (sizeof(int)*n)
```



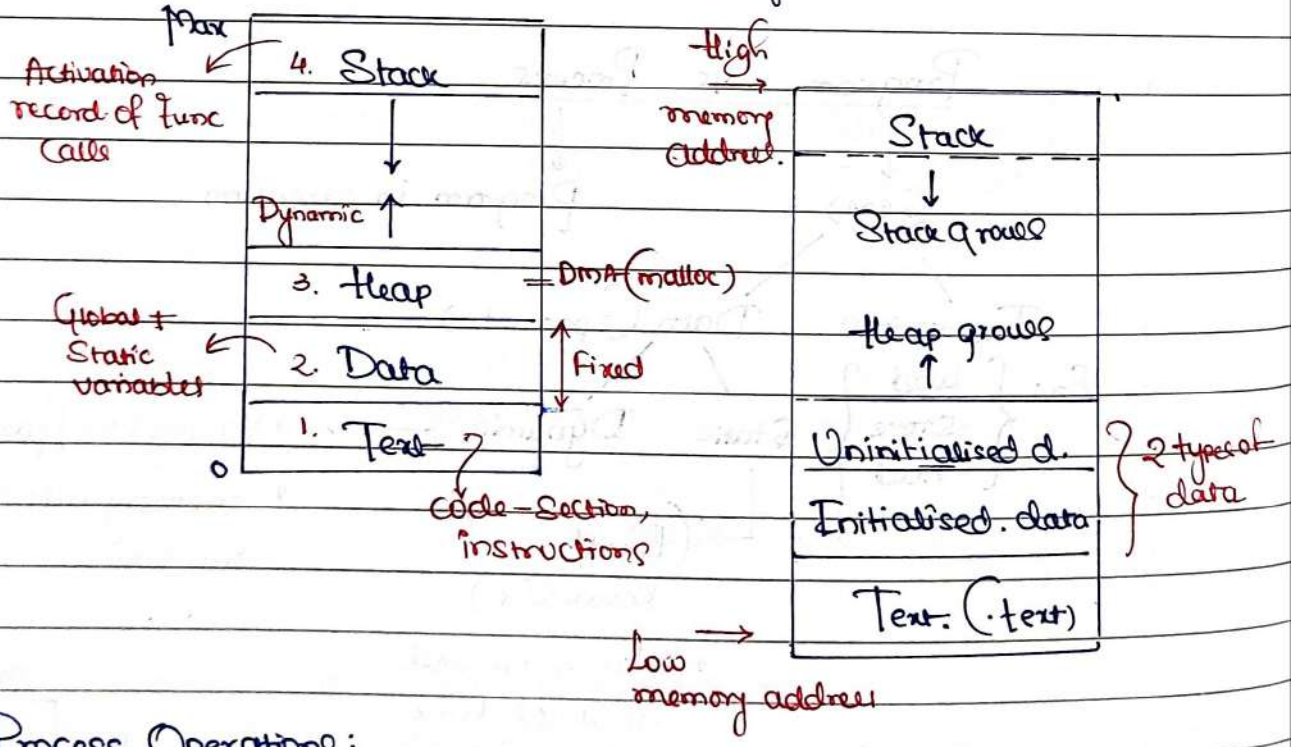
Process:
* Program in Execution
* When program is loaded from Disk to main memory then program is converted to process.
* Instance of a program.
* Locus of control
* Animated Spirit.



* Program is passive.
* Program is always on disk.

★ Process is an abstract data type.

Representation of process in memory.



Process Operations:
1. Create() : resource allocation
2. Schedule() : the act of selecting process to run on CPU
3. execute/run() : the act of executing instruction from code section
4. block/wait : for I/O operation / System call
5. Suspend : act of moving process from memory to disk.

6. Resume : act of moving process from disk to memory
7. Terminate : act of resource deallocation.

Process as an entity is associated with several attributes :

1. Identification : <Pid, gid, Ppid...>
2. CPU related : <Program counter, priority, state, burst time...>
3. Memory related : <Size, limits, ...>
4. File related : <list of files, ...>
5. Accounting : <resources ....>

And many more attributes but these are some important ones.

PCB <Process Control Block> :

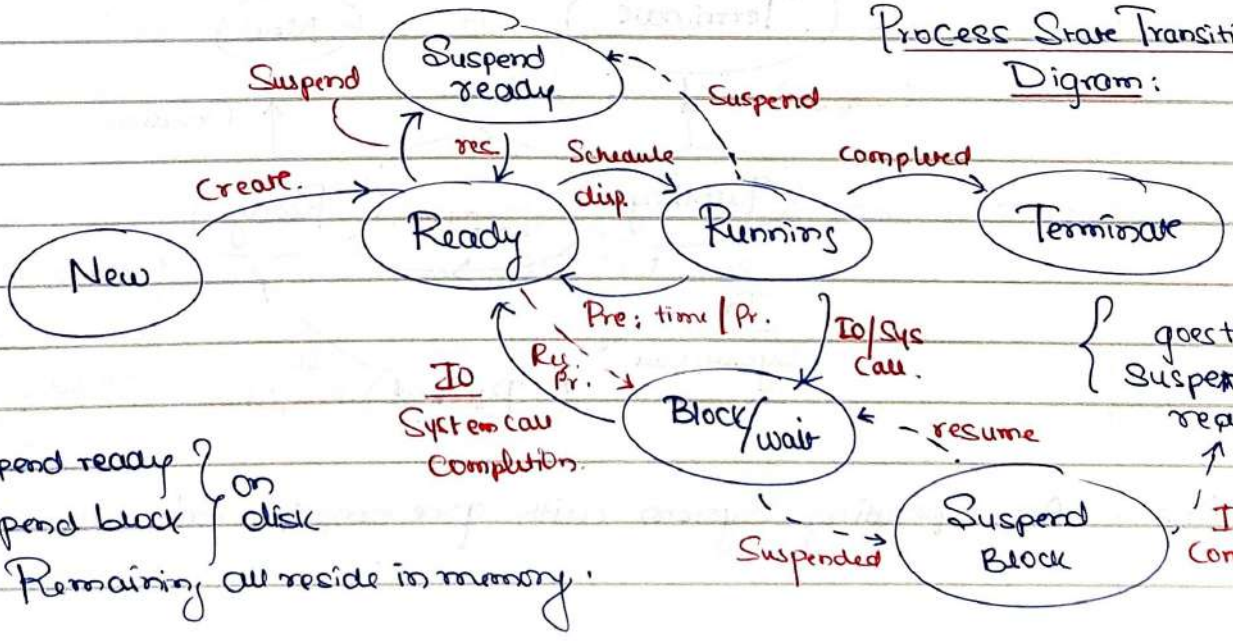
* Contains all attributes of process.
* each process has its own PCB.
* PCB is Stored in memory.

★ Process Context : content of PCB

Process States.

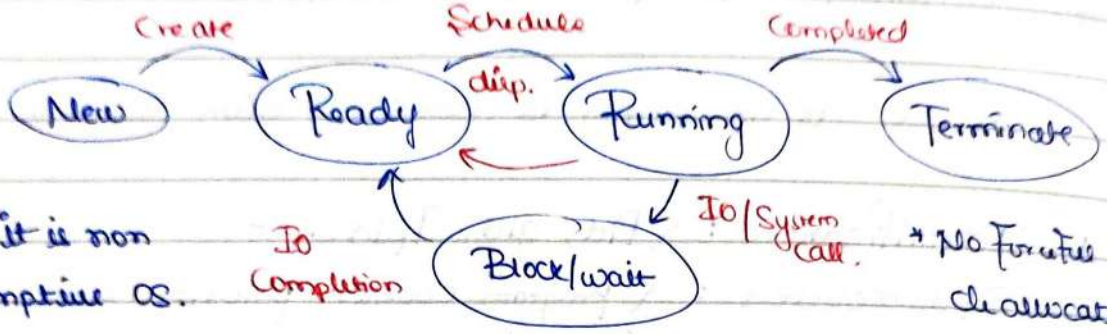
1. new : a process is created and resource is allocated
2. ready : ready to run on CPU
3. running : executing instructions on CPU
4. block/wait : process needs to perform IO/ system call.
5. terminate : resource deallocation.

Process State Transition Digram :



Suspend ready } on disk
Suspend block }
Remaining all reside in memory.

Process State transition diagram of uniprogrammed O.S:

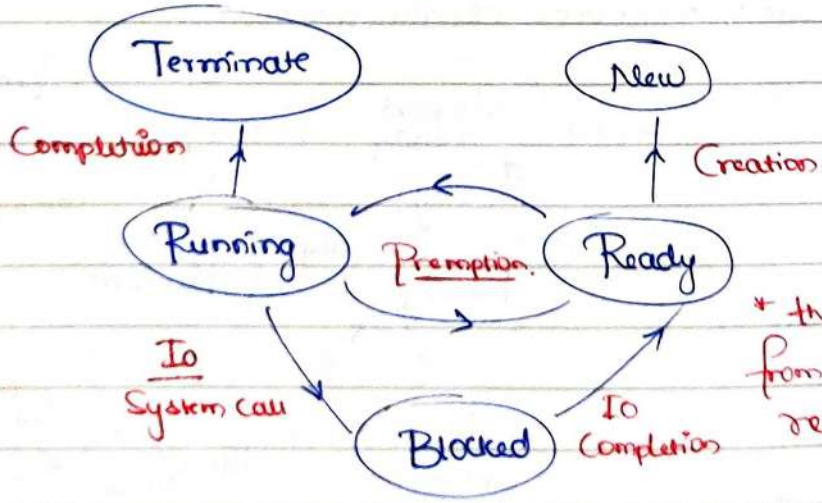


Note: it is non preemptive OS.

* just adding a transition from running to ready state, then the System becomes pre-emptive.

Notes: 1. As long as the process is in ready + running + block states, it is in main memory.
2. There can be many ready, block processes.
3. Maximum number of running process depends on no. of CPUs.
4. The process may go from ready state to block/wait state because of any missing resource.

Q1. The process state transition diagram given below is representation of: Ans: An Operating System with preemptive Scheduler.



Ans An Operating System with pre emptive Scheduler.

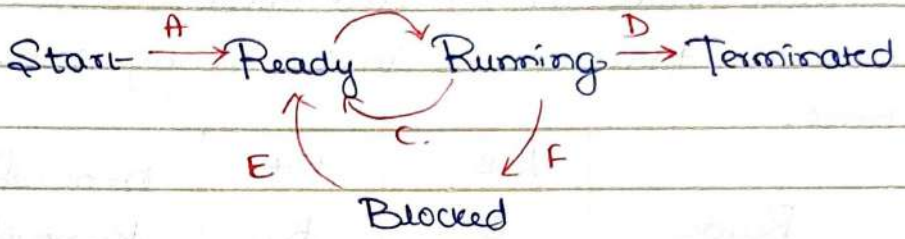
Q2. In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in ready state:

(i) If process makes a transition D, it would result in another process making transition A immediately.

(ii) A process P2 is blocked state can make transition E while another process P1 is in running state.

(iii) The OS uses preemptive scheduling

(iv) The OS uses non preemptive scheduling.



Which of the above statements are true? → II and III.

Q3. Which combination of the following feature will suffice to characterise an OS as multiprogrammed OS?

(a) More than one program may be loaded into main memory at the same time for execution.

Ans: Only 'a'

(b) If a program wait for certain events such as I/O, another program is immediately scheduled for execution

(c) If the execution of program terminates, another program is immediately scheduled for execution.

Q4. The maximum number of processes that can be in ready state for a computer system with 'n' CPUs:

Ans: Independent of n. (theoritically there can be infinite no. of programs in ready state).

* Ans will be n if 'running state' was asked.

Qs. Consider the following Statements about process State transitions for a system using preemptive Scheduling.

I. A running process can move to ready State
II. A ready process can move to running State
III. A blocked process can move to running State
IV. A blocked process can move to ready State.

Which of the above Statements are true? Ans: I, II, IV

Q6. Consider a system having 'n' cpus ($n \geq 1$) and 'k' processes ($k \geq n$). Calculate lower bound and upper bound of the number of processes that can be in the ready, running and blocked states.

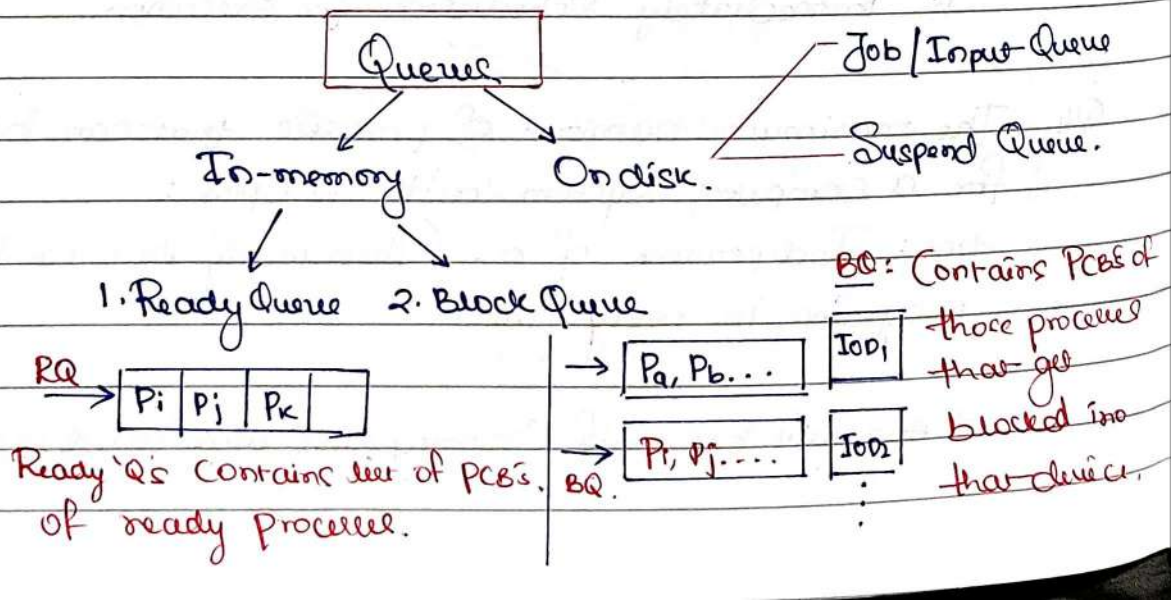
	L.B.	U.P
Ready	0	K
Running	0	n
Block.	0	K

n = no. of cpus ($n \geq 1$)
k = no. of processes. ($k \geq n$)

	L.B.	U.P.
Ready	0	K
Running	0	K
Block	0	K

n = no. of cpus ($n > 1$)
k = no of processes ($k < n$)

Scheduling Queues and State-queuing diagram



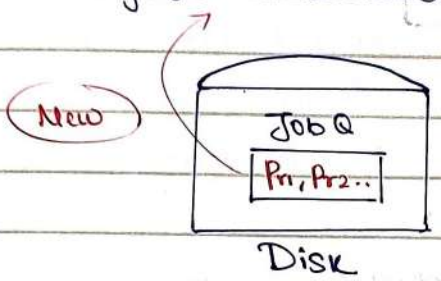
Queue on disk

1). Job Queue
Input Queue

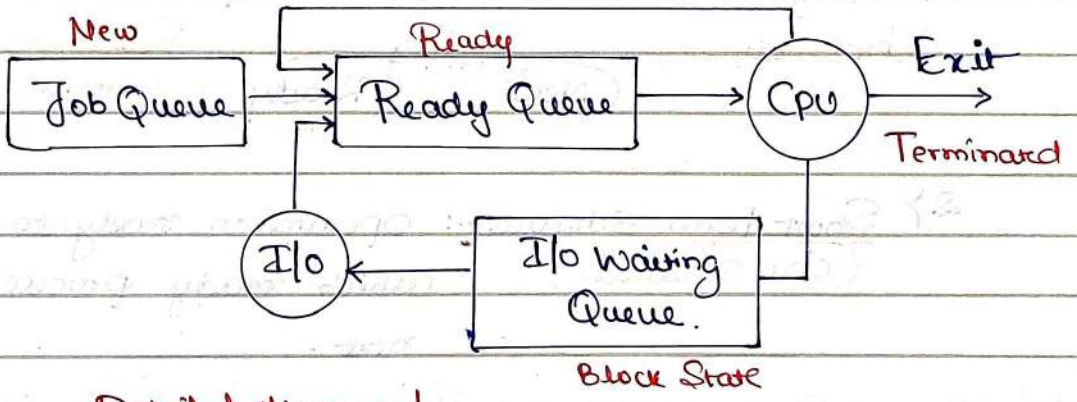
* Contains programs that are ready to be loaded in memory

2.) Suspend Queue

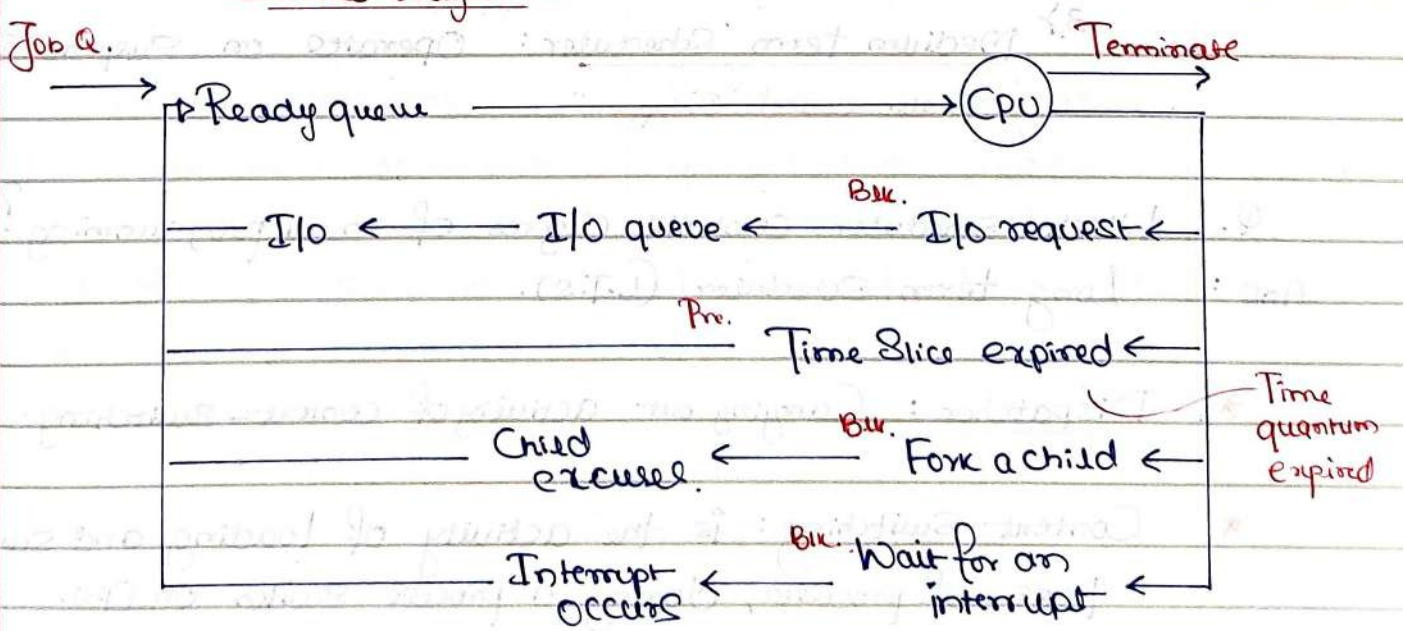
* Contains list of processes that get suspended from memory into disk.



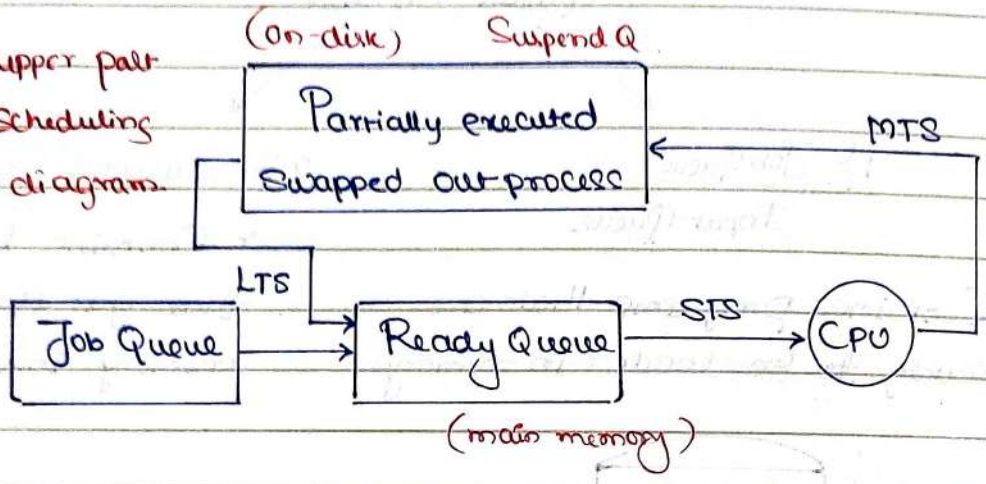
State Queuing Diagram:



Detailed diagram ↓



The upper part of Scheduling queue diagram.



Schedulers and Dispatchers

* Scheduling means making a decision
* Scheduler is a component of O.S that makes decisions.
* There are three types of Schedulers:

1.) Long term scheduler (LTS): operates on Job-Q.
(~~CPU Scheduler~~)

Selects process based on a criteria

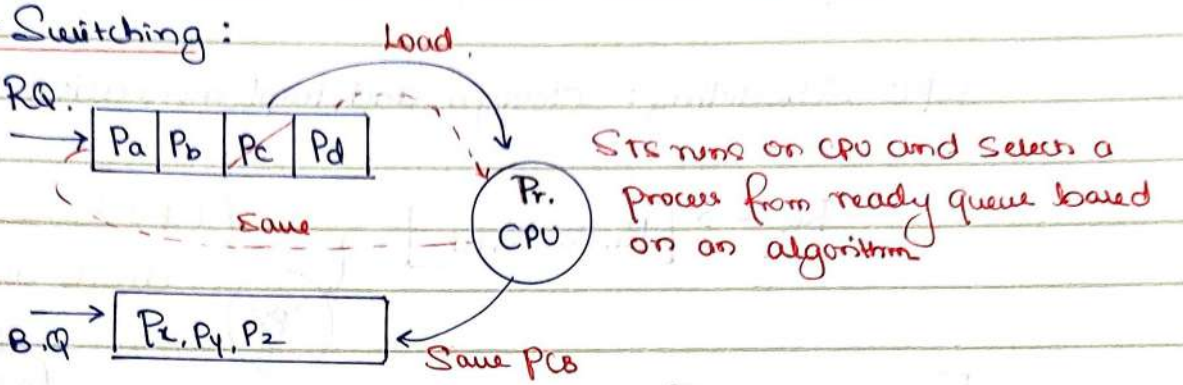
2.) Short term scheduler: operates on ready to Queue, to decide which ready process should run next.
(CPU Scheduler)

3.) Medium term Scheduler: Operates on Suspend Queue.
also called "Swapper".

Q. Which scheduler controls degree of multi programming?
Ans: Long term Scheduler (L.T.S).

* Dispatcher: Carrying out activity of context-switching.
* Context Switching: is the activity of loading and saving the PCBs of process, during a process switch on CPU.

Context Switching :



* Time taken by dispatcher to load and save the PCB's is known as "Context Switching time" or "CPU Scheduling Overhead" or "dispatch latency"

Q1. Let the time taken to switch between user and kernel modes of execution be t_1, while the time taken to switch between two user processes be t_2. Which of the following is true?

Context switching time = t_2
mode shifting time = t_1
$\because t_1 < t_2$.

* process switching involves mode shifting time

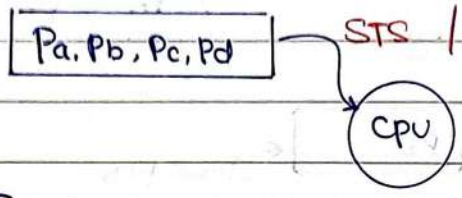
Q2 Dispatch latency is defined as:
The time to switch from one process to another on CPU

Q3. Which of the following is an invalid process state transition?

a. Ready — running ✓
b. Running — terminate ✓
c. Block — ready ✓
d. Ready — suspend ✓
e. Suspend — running ×
f. block — terminate ×
g. suspend — ready ✓
h. block — new ×
i. suspend — terminate ×

CPU Scheduling

CPU Scheduling: design and implementation of S.T.S.



Algorithms (CPU-Scheduling technique)

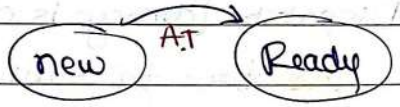
Functions: Select a process from R.Q to run into CPU.
(S.T.S)

Goals (Scheduling Criteria):

* Maximize CPU utilization (throughput)
* Minimise waiting time (WT), turn around time (TAT), response time (RT).

Process Times:

1. Arrival time (AT): first time when the process comes from new to ready state is called arrival time.



2. Waiting time (WT): total time spent by the process in ready queue.

3. Burst time (BT): time spent by the process executing in the CPU is called CPU burst time.

4. I/O Burst time (IOBT): time spent by the process in block queue performing I/O operation is called I/O burst time

5. Completion time: time at which process complete.

6. Turn Around time (TAT) : the time taken by the process from its arrival time to completion time is called turn around time.

$$TAT = C.T - A.T.$$

7. Waiting time (WT) : total time spent by the process in ready queue is called waiting time.

$$WT = TAT - (BT + IOBT)$$

Note:

1. n processes $(P_1 \dots P_n)$
2. $AT(P_i) = A_i$
3. $BT(P_i) = X_i$
4. $IOBT(P_i) = Y_i$
5. $CT(P_i) = C_i$

a) $TAT(P_i) = (C_i - A_i)$

Average $TAT = \dfrac{\sum_{i=1}^{n} (C_i - A_i)}{n}$

b) $WT(P_i) = (C_i - A_i) - (X_i + Y_i)$

Average $WT = \dfrac{\sum_{i=1}^{n} (C_i - A_i) - (X_i + Y_i)}{n}$

Schedule length (L) : Total time taken to complete all 'n' processes as per schedule

* No. of schedules with n-processes = n! (non preemptive)
* No. of schedules with n-processes = ∞ (pre-emptive)

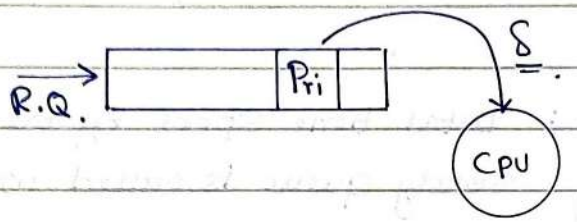
$$L = (\text{completion time of last process}) - (\text{A.T of last process})$$
$$\Rightarrow \max(C_i) - \min(A_i)$$

Throughput = no. of processes completed per unit time.

$$\eta = \frac{n}{L}$$

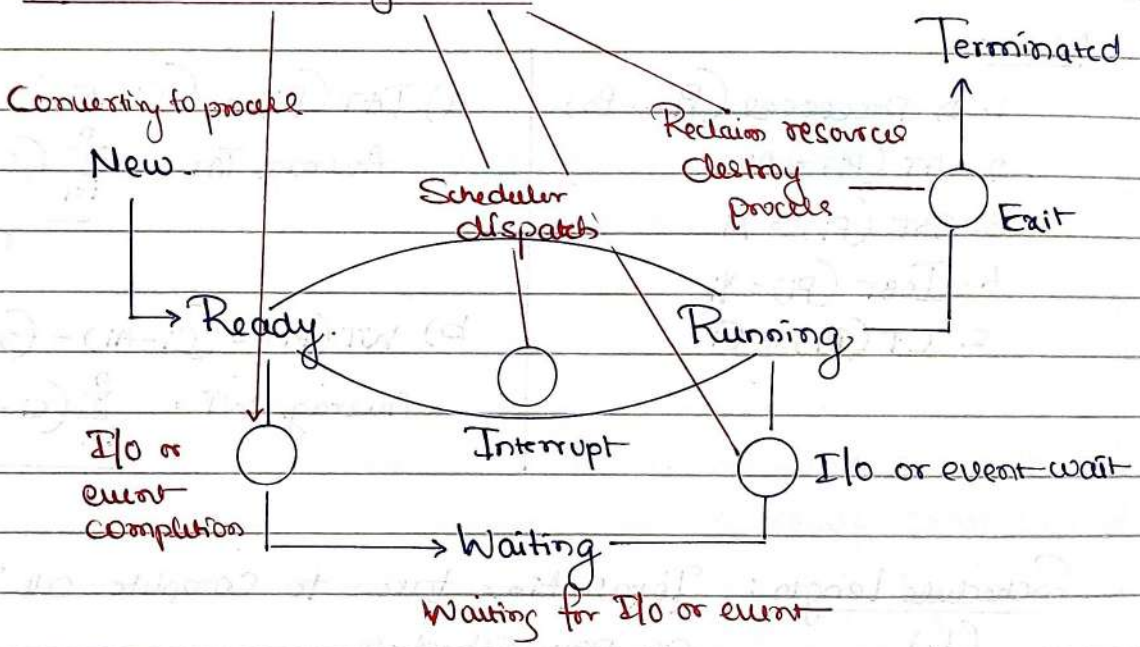
Context-Switch time : CPU Scheduling Overhead

Let CPU Scheduling overhead = "δ"



δ → Time taken to load PCB to process from Ready queue onto CPU.

CPU Scheduling Occurs



Types of CPU Scheduling :

1. Pre emptive
2. Non preemptive.

Scheduling Criteria :

Maximize	Minimize
* CPU utilization	* Turnaround time * Waiting time * Response time

CPU - I/O Bursts:

Process execution consists of a cycle of CPU execution and I/O wait:

* different processes may have distributions of bursts.
* CPU bound processes: performs lots of computations in long bursts, very little I/O.
* I/O bound process: performs lots of I/O followed by short burst of computation.
* ideally, the system admits a mix of CPU bound I/O bound processes to maximise CPU and I/O.

Example:

```
:
load store
add store          } CPU Burst
read from file

wait for I/O       } I/O Burst

store increment    }
  index            } CPU Burst
write of file      }

wait for I/O       } I/O Burst

load store         }
add store          } CPU Burst
read from          }
  file             }

wait for I/O       } I/O Burst.
:
```

1. FCFS (First Come First Serve)

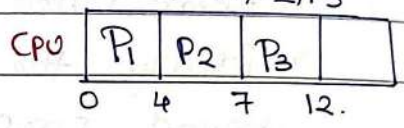
* Selection Criteria: A.T
* Mode of operation: Non pre-emptive.

* Conflict resolution: Lower process id.
* Assumption:
  (i) Time is in clock ticks
  (ii) No IOBTs
  (iii) Scheduling overheads (δ) = '0'.

Ex:1

P.no.	A.T.	B.T	C.T.	TAT	WT.
1	0	4	4	4	0
2	0	3	7	7	4
3.	0	5	12	12	7

RQ → P_1, P_2, P_3



Time

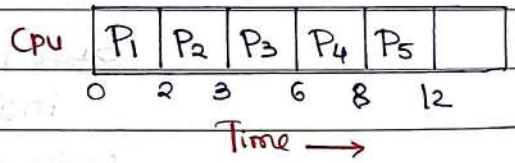
Gantt Chart →

Avg TAT = $\frac{23}{3}$. Avg. WT = $\frac{11}{3}$.

L = 12.

Ex 2

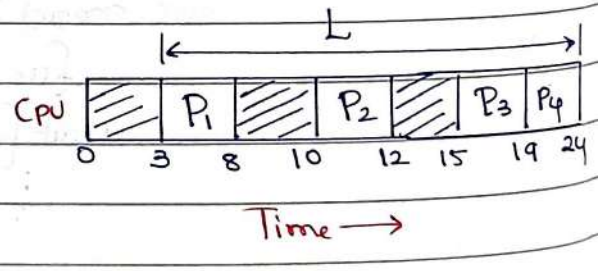
P.no	A.T	B.T	C.T.	TAT.	WT.
1	0	2	2	2	0
2	0	1	3	3	2
3	2	3	6	4	1
4	3	2	8	5	3
5	5	4	12	7	3



Avg. TAT = $\frac{21}{5}$, Avg. WT = $\frac{9}{5}$, L = 12. $\eta = \frac{\Omega}{L} = \frac{5}{12}$.

Ex 3

P.no	A.T	B.T	C.T	TAT	WT
1	3	5	8	5	0
2	10	2	12	2	0
3	15	4	19	4	0
4	18	5	24	6	1

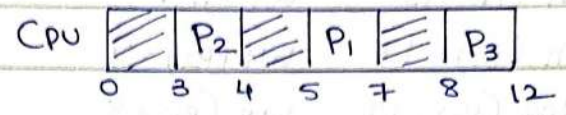


% CPU idleness / L = $\frac{5}{21}$.

Date 22/1/23

Ex.4.

P.no	A.T	B.T
1	5	2
2	3	1
3	8	4

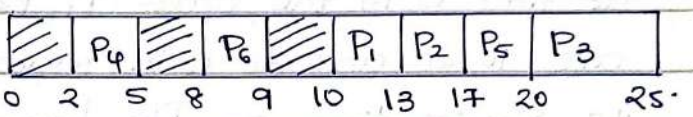


L = 12 - 3 = 9

Ex.5.

P.no	A.T	B.T
1	10	3
2	11	4
3	20	5
4	2	3
5	12	3
6	8	1.

→ RQ : P4 P6 P1 P5 P3.



L = 25 - 2 = 23.

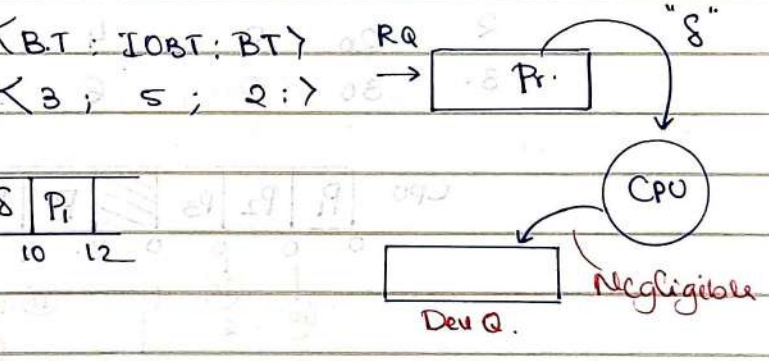
% CPU idleness = 4/23.

CPU Scheduling Techniques / Algorithms

1. FCFS: Scheduling with IO-BT's and CPU Scheduling overheads.

Ex.1: δ = 1

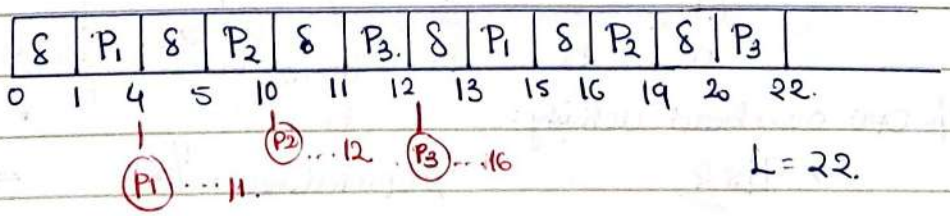
P.no	A.T	⟨B.T : IOBT : BT⟩
1	0	⟨3 ; 5 ; 2 :⟩



Ex 2: δ = 1.

P.no	A.T	⟨B.T	IOBT	BT⟩
1	0	3	7	2
2	2	5	2	3
3	5	1	4	2

System has multiple IO devices.



L = 22.

$TAT(P_1) = 15$ $WT(P_1) = 1$
$TAT(P_2) = 17$ $WT(P_2) = 5$
$TAT(P_3) = 17$ $WT(P_3) = 8$
% CPU Idleness = 0%

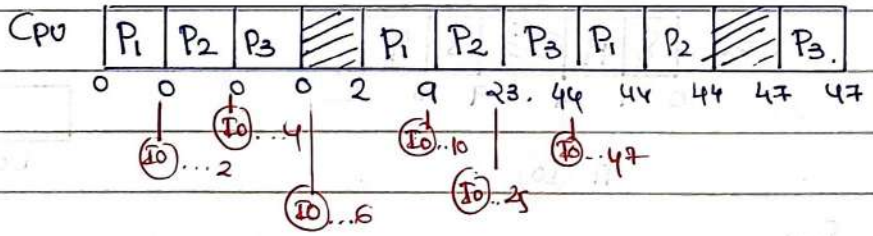
* With 'δ' > 0 the formula for W.T becomes
$WT = TAT - (BT + IOBT + n.\delta)$
* n = no of times process get scheduled into CPU.

Q1. Consider three processes P_1, P_2, P_3 arriving in the ready queue at time 0 in the order P_1, P_2, P_3. Their service time requirements are 10, 20 and 30 units respectively. Each process spends 20% of its service time on I/O followed by 70% of its service time on computation at CPU and last 10% on IO before completion. Assuming concurrent I/O and negligible scheduling overhead, Calculate FCFS scheduling:
(i) Average TAT (ii) % CPU Idleness.

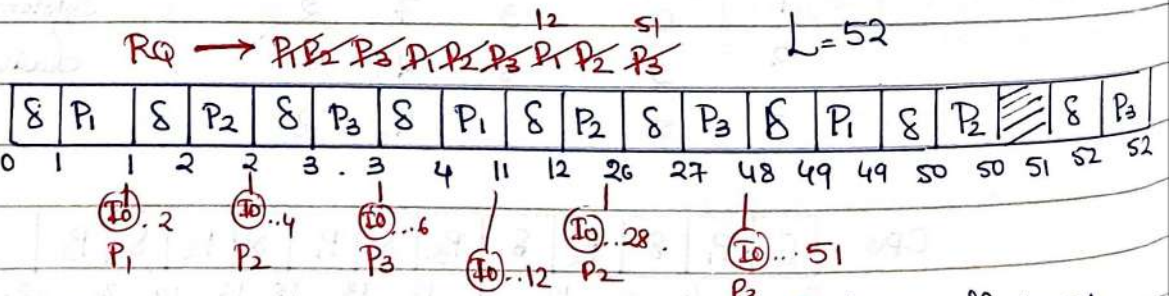
L = 47
% CPU idleness = 5/47
Avg TAT = $\frac{44+44+47}{3} = 44$

Soln:

P.no	S.T	AT	<IOBT ; BT; IOBT;>
1	10	0	2 7 1
2	20	0	4 14 2
3.	30	0	6 21. 3



Above problem with δ = 1.



* % CPU overhead activity: ⇒ 9/52
* % CPU idleness = 1/52
* % CPU efficiency ⇒ $100 - (\frac{9}{52} + \frac{1}{52})$

Homework: Repeat Q 2 and 3 assuming System has only one IO-device.

1) $\delta = 2$

P.no	AT	< BT	IOBT	BT >
1	0	3	4	2
2	5	2	0	3
3	20	1	15	6

2) $\delta = 1$.

P.no	AT	IO.	BT.	IOBT
1	3	5	2	3
2	8	2	10	4
3	12	6	12	1.

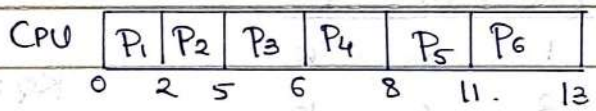
2. Shortest Job First (SJF) / Shortest Process Next (SPN)

* Selection Criteria: Burst Time
* Mode of operation: Non preemptive
* Conflict resolution: Lower process id
  : Among the processes present in RQ Select the one having least B.T and Schedule it on CPU.

Ex.1

P.no	A.T	B.T
1	0	2
2	2	3
3	3	1
4	5	2
5	8	3
6	10	2

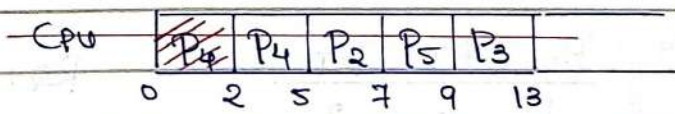
RQ: P1 P2 P3 P4 P5 P6

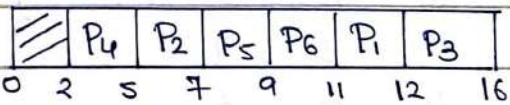


Ex.2

P.no	A.T	B.T
1	10	1
2	4	2
3	6	4
4	2	3
5	5	2
6	3.	2

~~RQ: P4, P2, P5, P3~~





L = 16 - 2 = 14

∴ CPU idleness = 0

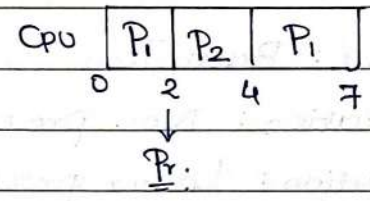
3. Shortest Remaining Time First (SRTF) / Pre emptive SJF

* Selection Criteria: BT
* mode of operation : Pre emptive
* Conflict resolution: Lower process id

→ Pre emption of running process is based on availability of Strictly Shorter process.

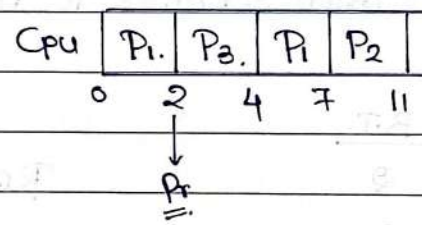
Ex: 1.

P.no	AT	BT
1	0	5
2	2	2



Ex: 2

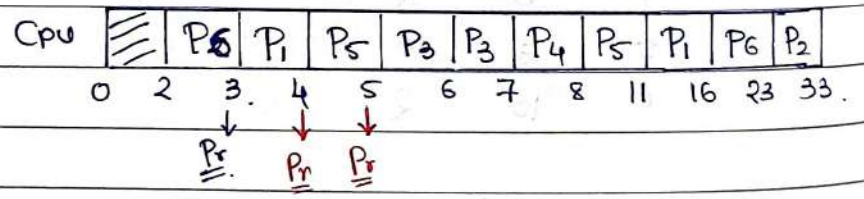
P.no.	A.T.	BT
1	0	5
2	1	4
3	2	2



Ex: 3

P.no	A.T	BT
1	3	6
2	2	10
3	5	2
4	6	1
5	4	4
6	2	8

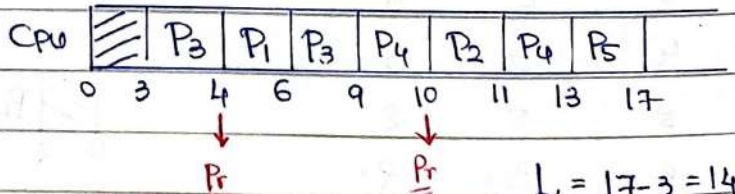
RQ: P2, P6 P1. P5 P3 P4



Ex: 4

P.no	AT.	BT
1	4	2
2	10	1
3	3	4
4	5	3
5	3	4

RQ: P3, P5, P1, P4 P2

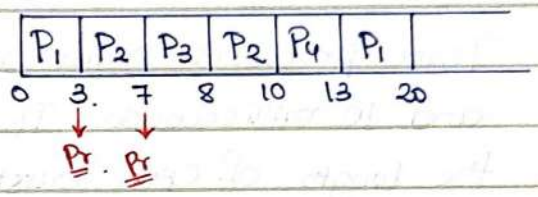


L = 17 - 3 = 14

Q1. Consider the following processes, with AT and length of the CPU burst given in ms. The scheduling algo used is premptive SRTF.

Process	AT.	B.T
P_1	0	10
P_2	3	6
P_3	7	1
P_4	8	3

The average turn around time of these processes is 8.25



Average TAT = $\frac{20+7+1+5}{4}$

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

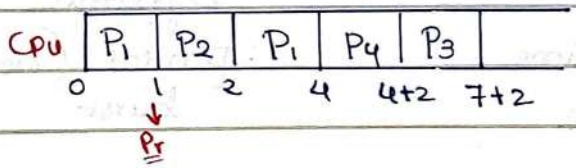
Q2.

Process	P_1	P_2	P_3	P_4
A.T	0	1	3	4
B.T	3	1	3	7 (z)

The processes are run on a single processor using pre-emptive SRTF. If avg WT = 1ms, then value of z = ?

Average WT = 1 ms (SRTF)

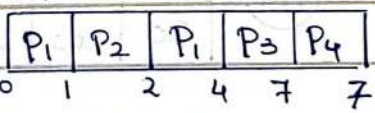
→ Case 1: $z < 3$.



Avg WT = $\frac{1+0+(2+1)+0}{4}$

$1 = \frac{z+2}{4}$ $\therefore z = 2$

Case 2: $z > 3$.



Avg WT = $\frac{1+0+1+3}{4} = \frac{5}{4} = 1.25$.

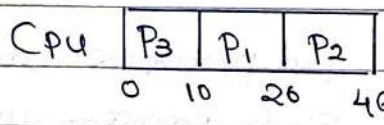
Because Avg WT = 1. So z is less than 3.

Performance of SJF/SRTF:

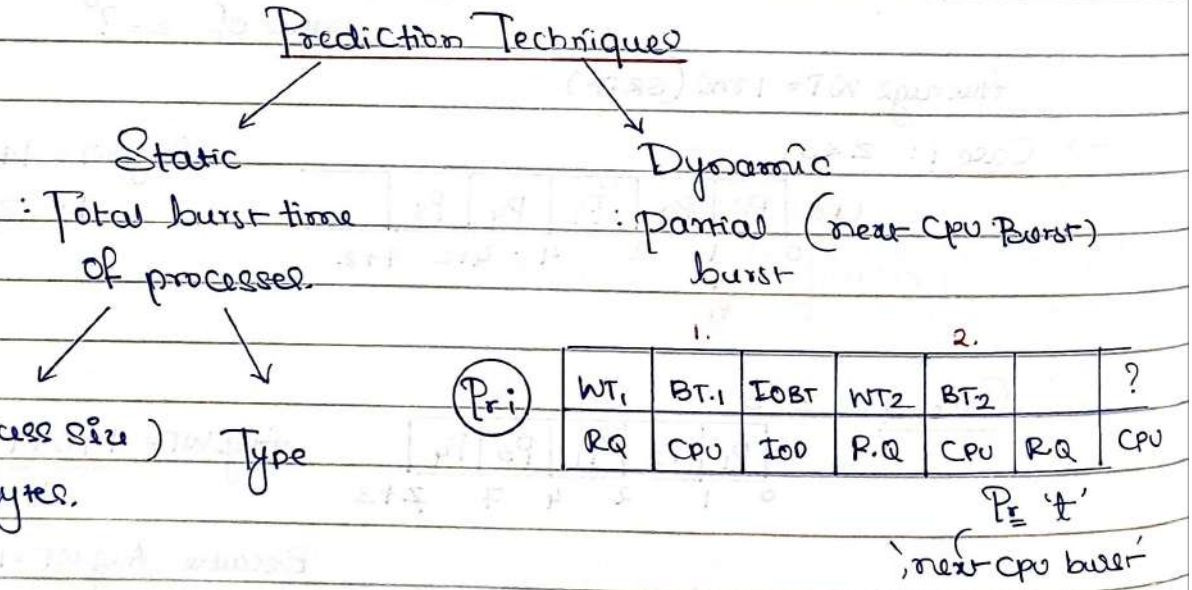
1. It favors shorter process:
   - → Advantages: * Complete more processes, max throughput
     * minimizing of Avg TAT, Avg WT.
   - → Drawback: * Starvation to longer processes.

Note: * SJF/SRTF is considered the "optimal algorithm"
* Since B.T's of processes are not known aprior, SJF/SRTF is "not practically implementable"!
* SJF is used as a benchmark to measure performance of other algorithms.
* SJF can be implemented with predicted burst times.

Q3. Three processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the lengths of CPUs bursts, the min achievable average waiting time for these three processes in non-preemptive scheduler is 12 milliseconds.



Avg WT = $\frac{10+26+0}{3} = \frac{36}{3} = 12$



Exponential Averaging Technique / Aging Algorithm: (To predict next CPU burst)

Let P_i : process
Let t_i : Completed B.T
Let T_i : predicted B.T

Let T_{n+1} denote next predicted B.T

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n$$
$$0 < \alpha < 1$$

Date 24/01/23

Recurrence Relation:

$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ — (I)

$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_n$ — (II)

} Back Substitution

$\tau_{n+1} = \alpha t_n + (1-\alpha)[\alpha t_{n-1} + (1-\alpha)\tau_{n-1}]$

$= \alpha t_n + \alpha(1-\alpha)t_{n-1} + (1-\alpha)^2 \tau_{n-1}$ — (III)

$= \alpha t_n + \alpha(1-\alpha)t_{n-1} + \alpha(1-\alpha)^2 t_{n-2} + (1-\alpha)^3 \tau_{n-2}$ — (IV)

(τ_1 = initial guess)

Given the value of 'α' and τ_1 one can predict next CPU B.T.

Q1. Consider a system using exponential averaging technique to predict next CPU BT. Given $\alpha = 0.5$ and $\tau_1 = 10$. The process B.T's in previous runs are: 4, 8, 12, 10. Predict the next CPU burst time of process.

Soln:- $T_5 = ?$ 4, 8, 12, 10 (t_1, t_2, t_3, t_4)

$T_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ — (I)

$T_5 = 0.5 \times t_4 + 0.5 \times \tau_4$ (9.75)

$= \frac{1}{2} \times (t_4 + \tau_4) = \frac{1}{2}(10 + \tau_4)$

$\tau_4 = \frac{1}{2}(t_3 + \tau_3) = \frac{1}{2}(12 + \tau_3)$ (7)

$= \frac{19.5}{2} = 9.75$

$\tau_3 = \frac{1}{2}(t_2 + \tau_2) = \frac{1}{2}(8 + \tau_2) = 7.5$ (7)

$\tau_2 = \frac{1}{2}(t_1 + \tau_1) = \frac{1}{2}(4+10)$

$\tau_2 = 7.$

$\therefore T_5 = \frac{1}{2}(10 + 9.75) = \frac{19.75}{2} = 9.875$

4 Highest Response Ratio Next (HRRN)

→ : HRRN not only favours shorter process but also limits the waiting time of longer process

* Selection Criteria: Response ration : $\frac{W.T + BT}{BT}$
* Mode of operation: Non pre emp.

Ex 1.

P.no	AT	BT
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

HRRN:

longer shorter

P1	P2	P3	P5	P4

0 3 9 13 15 20

t=13 $R_4 = 7+5/5 = 12/5$

$R_5 = \frac{5+2}{2} = \frac{7}{2}$ (higher)

5. Longest Remaining Time First (LRTF)

→ It is just opposite of ~~longest~~ ~~Smallest remaining time first.~~

* Selection Criteria: Burst Time
* Mode of operation: Pre emptive.

Ex 1.

P.no	AT	BT
1	0	2
2	0	4
3	0	8

CPU (it's pre emp)

P3	P2	P1

0 8 12 14

~~TAT:~~

~~$P_1 = 14-0 = 14$~~

~~$P_2 = 12-0 = 12$~~

~~$P_3 = 8-0 = 8.$~~

CPU

P3	P2	P3	P2	P3	P1	P2	P3	P1	P2	P3

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

P_r P_r P_r P_r P_r P_r P_r P_r

Avg T.AT = $\frac{12+13+14}{3}$

Avg T.A.T. = 13

Q1. Consider 3 processes A, B, C with compute BT.s are 4, 6, 9 units. All the processes arrive at time zero. Consider the longest remaining time first (LRTF). In LRTF ties are broken by giving priority to processes with lowest Pid. Find the avg of completion times of A, B?

RQ: P_1^4 P_2^6 P_3^9

P.no.	AT	BT
A	0	4
B	0	6
C	0	9

P3	P2	P3	P2	P3	P1	P2	P3	P1	P2	P3	P1

0 3 4 5 6 7 8 9 10 11 12 13 14

P2	P3	P4	P2	P3

14 15 16 17 18 19

Avg. CT = $\frac{17+18}{2} = 17.5$

Q2 In a system using single processer, a new process arrives at the rate of 10 processes per minute and each such process require 5 seconds of service time. What is % cpu utilization?

Sol^n :- 60 $ → 10 processes
└→ (5 $ec)

(P1, P2 ... P10)
←——— 60 sec cycle. ———→

% CPU utilization = $\frac{50}{60} \times 100$

= 83.33%

Q3. Six jobs are waiting to be run. The expected running times are 9, 7, 5, 2, 1 and x respectively. Where $5 < x < 7$ and avg CT. is 13. Find the value of x using SJF algorithm? (Assume all jobs arrive at same time = 0).

6. Priority Based Scheduling

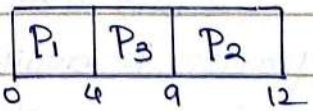
→ Priority based scheduling works exactly like SJF/SRTF except that it looks at priority instead of Burst time.

* Selection Criteria : Priority
* Mode of operation : Non preemptive / preemptive
* Tie breaking : Lower process id.

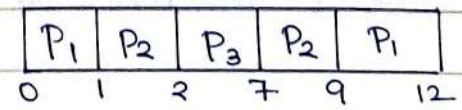
Ex 1.

Prio	P.no	AT	BT
4	1	0	4
5	2	1	3
8	3	2	5

Non pre-emp



Pre emp.

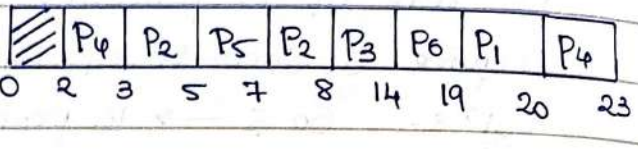


Date 25/01/23

Ex 2.

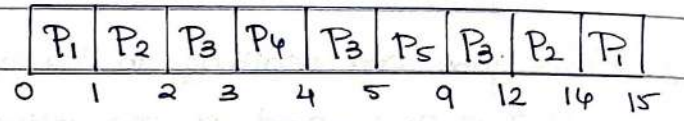
Prio.	P.no	AT	BT
4	1	4	1
6	2	3	3
8	3	8	6
3	4	2	4
7	5	5	2
5	6	3	5

$L = 23 - 2 = 21$.



Ex 3.

Prio	P.no	AT	BT
4	1	0	21
5	2	1	3
6	3	2	5
7	4	3	1
8	5	5	4



Avg TAT = $\frac{15+13+10+1+4}{5} = \frac{43}{5} = 8.6$

Avg WT =

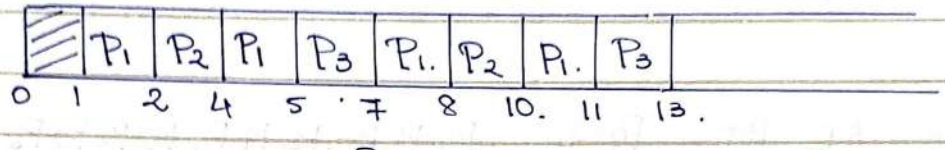
Performance of priority based scheduling:

* Causes Starvation to low priority processes, a solution exists "Dynamic priorities": It increases the priority of the processes at regular intervals of time using an algorithm called "Aging" algorithm.

Q1. Consider a System with Pre-emptive priority based scheduling with processes P_1, P_2, P_3 having infinite instances of them. The instance of these processes arrive at regular intervals of 3, 7 and 20 ms respectively. The priority of these process instances is inverse of their periods. Each of the process instance P_1, P_2, P_3 consumes 1, 2 and 4 ms CPU time respectively. The 1st instance of each process is available at 1ms. What is the completion time of the 1st instance of process P_3?

Prio.	Priod	P.no	AT	BT	<Other instances>
1/3	3	1	1	1	<4, 7, 10, 13, 17...>
1/7	7	2	1	2	< 8, 15, 22, ... >
1/20	20	3	1	4	<21, 41, 61. . . . >

RQ: $P_1, P_2, P_3, P_{1_2}, P_{2_2}$,



Completion time of P_3 first instance is 13ms (at the end of 12)

Q2. Consider a System using preemptive priority based scheduling with dynamically changing priority. On its arrival a process is assigned a priority of zero and running process priority increases at the rate of 'β' and priority of the processes in the ready Q increases at the rate of 'α'. By dynamically changing values of α and β one can achieve different scheduling disciplines among the processes. What discipline will be followed for the following conditions

1. $\beta > \alpha > 0$ 2. $\alpha < \beta < 0$.

Ans: FCFS for first condition and LIFO for second condition.

7. Round Robin Scheduling

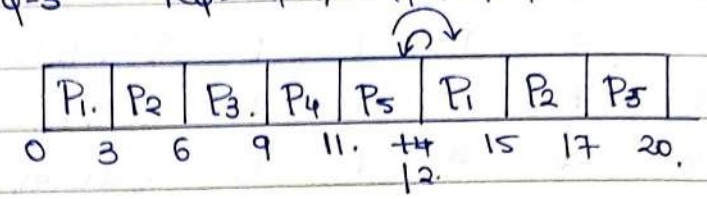
→ It is used in Pre-emptive multi programmed time shared Operating Systems.

* Selection Criteria: Arrival time + Time Quantum
* Mode of operation: Pre emptive
* Working principle: - each process is allotted a fixed time quantum
  - Pre emption is based on completion or expiration of time quantum.

Ex. 1.

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

TQ=3 RQ= 1, 2, 3, 4, 1, 5, 3, 5.

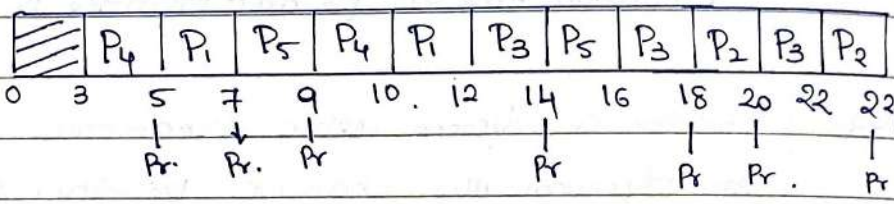


L = 20.

Ex 2.

Pno	AT	BT
1	4	4
2	15	5
3	8	6
4	3	3
5	5	4

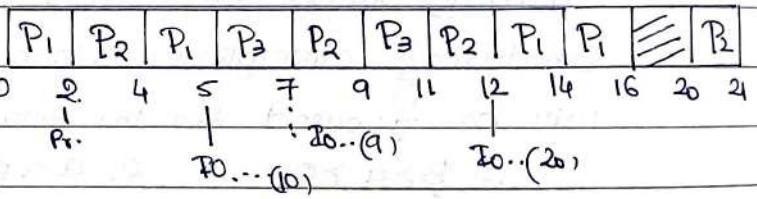
TQ=2 P4 P1 P5 P4 P1 P3 P5 P3 P2 P3 P2



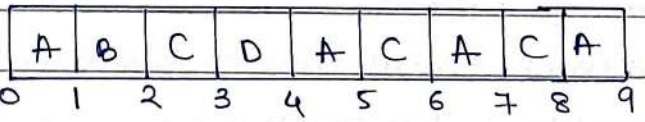
Pno	AT	BT	IOBT	BT
1	0	3	5	4
2	2	5	8	1
3	3	2	2	2

TQ=2

RQ: P1 P2 P1 P3 P2 P3 P2 P1



Q1. Consider a set of 4 processes, ABCD arriving in the order at time 0. Their burst time req are: 4, 1, 8, 1 respectively using RR scheduling with TQ: 1. The completion time of process A is 9.



Q2. Consider a system using RR Scheduling with 10 processes all arriving at time 0. Each process is associated with 20 identical request. Each process request consumes 20 ms of CPU time after which it spends 10 ms of time on I/O there after, initates subsequent request. Assuming scheduling overhead of 2 ms and

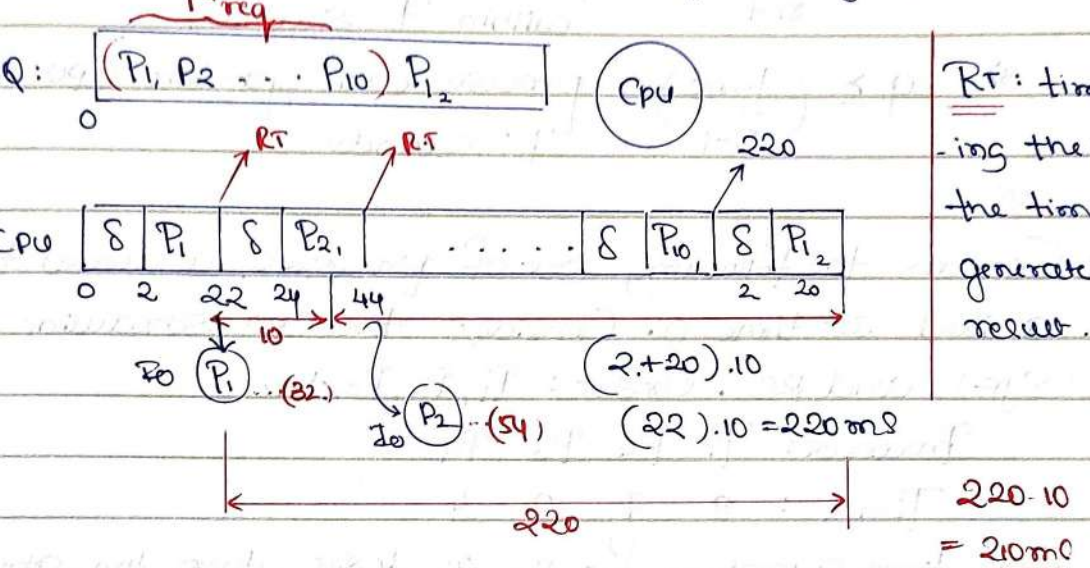
TQ of 20 ms. Calculate:

(i) Response time of 1st request of 1st process. – Ans: 22 ms

(ii) Response time of 1st request of last process – Ans: 220 ms

(iii) Response time of subsequent req. of any process – Ans: 210 ms

Sol:



RT: time of admitting the request to the time it generates into result.

Performance of Round Robin

Time Quantum.

very small: Efficiency = 0

very large: same like FCFS

Small

→ More context Switching (overhead)

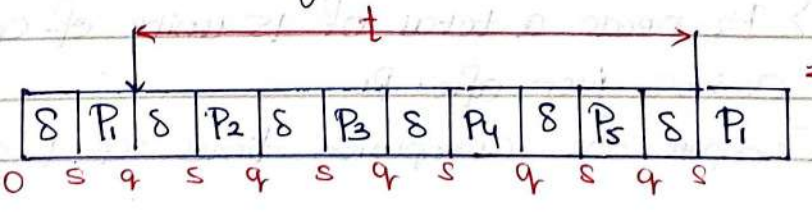
Makes the System better responsive

Large

→ Less context switching (less overhead), Poor interactiveness, responsiveness.

Q1. Consider a system with 'n' processes arriving at time 0+ with substantially large BT. The CPU scheduling overhead is 's' seconds, Time quantum is 'q' seconds. Using RR scheduling what must be value of time quantum 'q' such that each process is guaranteed to get its turn at CPU exactly after 't' seconds in its subsequent run at CPU.

→ $t - ns = (n-1)q$



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

1) $q = \left(\frac{t-ns}{n-1}\right)$: process will get into CPU exactly after 't' seconds.

2) $q < \left(\frac{t-ns}{n-1}\right)$: process will get into CPU atleast once within 't' seconds.

3) $q \geq \left(\frac{t-ns}{n-1}\right)$: process will get into CPU atleast every 't' seconds

HW

Q2. Consider the following set of processes, assumed to have arrived at time 0. Consider the CPU scheduling algorithms (SJF) and RR. Order: P_1, P_2, P_3, P_4.

Processes:	P_1	P_2	P_3	P_4
Times :	8	7	2	4

The time quantum for RR is 4 sec, then the absolute value of average turnaround times in SJF and RR is

SJF

CPU	P_3	P_4	P_2	P_1
	0	2	6	13	21

$$\text{Avg T.AT} = \frac{21+13+2+6}{4}$$

$$= \frac{42}{4} = 10.5$$

RR.

HW

Q3. Consider processes P_1 and P_2 arriving in RQ at time 0 with:

i) P_1 needs a total of 12 units of CPU time and 20 units of I/O time. After every 3 units of CPU time P_1 spends 5 units on I/O.

ii) P_2 needs a total of 15 units of CPU time and no I/O. P_2 arrived just after P_1.

Compute the completion times of P_1 and P_2 using SRTF & RR.

HW

Q4. Consider 4 processes P, Q, R, S scheduled on a CPU as per RR algorithm with TQ = 4 units. Processes arrive in order P, Q, R, S all at time t = 0. There is exactly one context-switch from S to Q, exactly one context switch from R to Q and exactly two context switches from Q to R. There is no context switch from S to P. Switching to a ready process after termination of another process is also considered a context switch.

Q5. Which of the following statements is/are correct in context of CPU Scheduling?

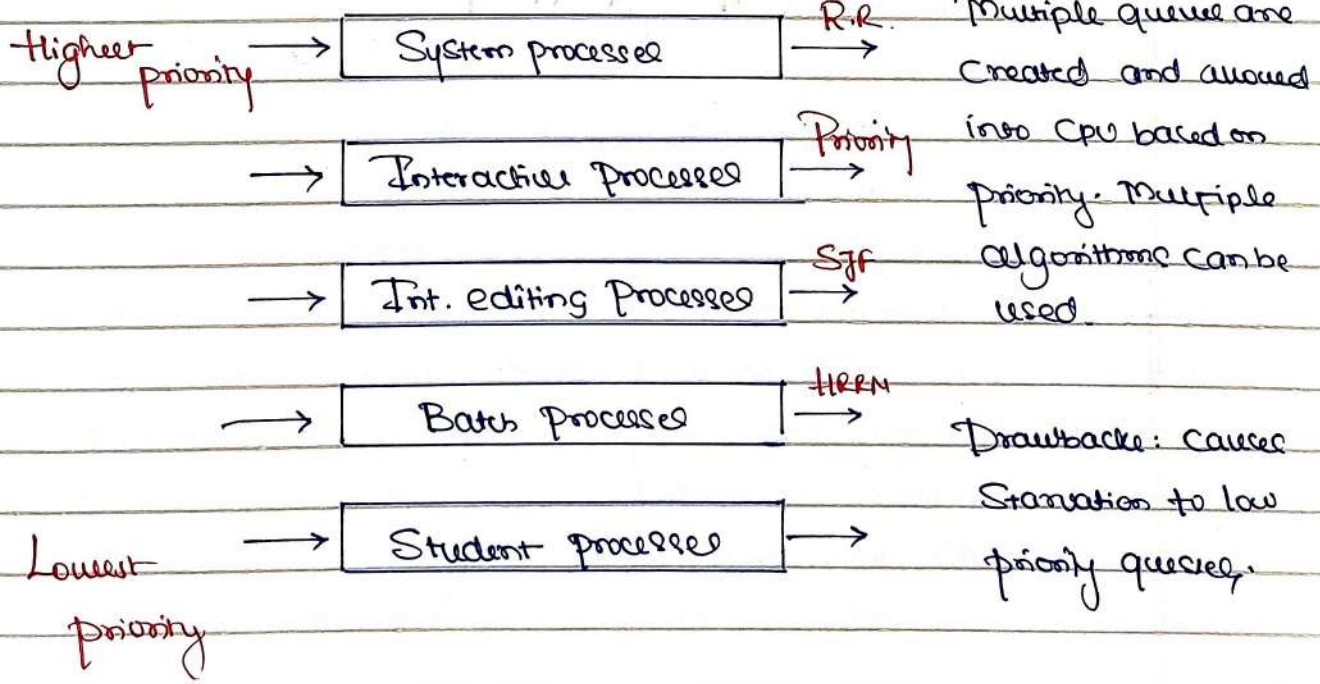
A. The goal is to only maximize CPU utilization and minimize throughput

B. Turnaround time includes waiting time

C. Implementing pre-emptive scheduling needs hardware support

D. RR policy can be used even when the CPU time required by each of the processes is not known apriori

8. Multi Level Queue Scheduling:

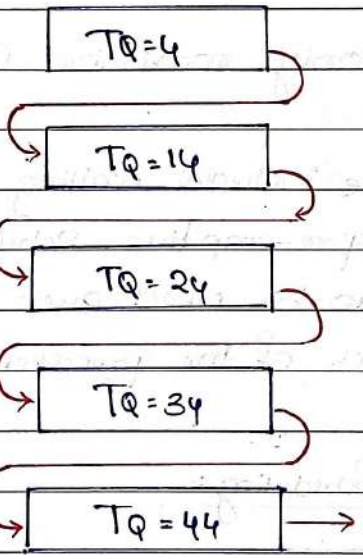


9. Multilevel feedback Queue Scheduling :

* Another way to put a reference on short lived processes, penalize processes that have been running longer.
* Pre emptive.

Q6. Consider a system which has CPU bound process, which require the burst time of 80 seconds, the multilevel feedback queue scheduling algorithm is used and the queue time quantum is '4' sec and in each level it is incremented by '10' seconds. Then how many times the process will be interrupted, and on which queue the process will terminate the execution ?

Ans : 4, 5.



B.T = 80ms.

4 + 14 + 24 + 34 = 80

Synchronization

* **IPC**: Inter process communication is a mechanism that allows processes to communicate with each other and synchronize their actions.

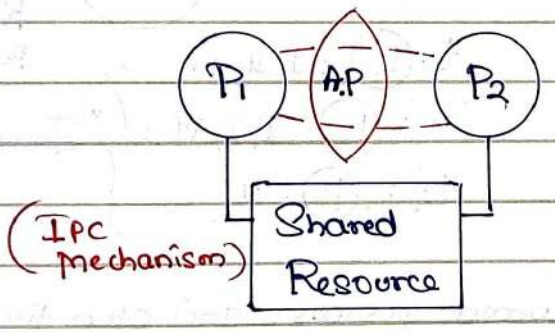
Every communication should be synchronized / co-ordinated

→ Lack of synchronization in IPC environment leads to:

1. Inconsistency (Incorrectness)
2. Loss of data
3. Deadlock (lockup)

Synchronization: It is agreed upon protocol in an IPC environment, to make sure that there is no inconsistency, data loss or deadlock.

IPC Environment:



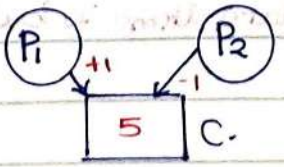
* Synchronization involves the orderly sharing of system resources by processes.
* We can think of this intersection as a system resource that is shared by two resources.

Types of Synchronization

Competitive

Two or more processes are said to be in competitive Sync. if they compete for accessibility of a shared resource.

Ex:

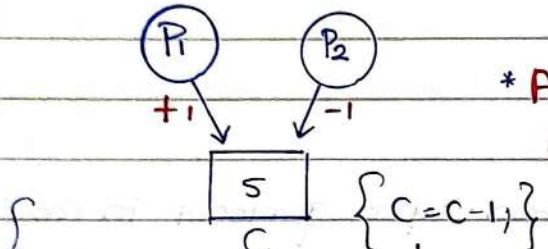


Co-operative

Two or more processes are said to be in Co-operative synchronization if they get affected by each other. ie execution of one process affects the other process.

Ex: Producer-consumer problem

Note: * Lack of synchronization among completing processes may lead to either inconsistency, dataloss.
* Lack of synchronization among co-operating processes may lead to deadlock.
* An application of an IPC environment may involve either competition or cooperation or both type of synchronization



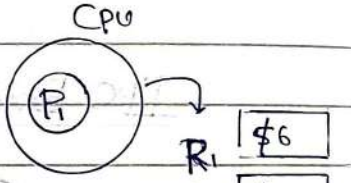
* Any process that executes in user mode can get preempted after any instruction.

I. Load R_1, C;
II. Inc R_1;
III. Store C, R_3;

Load R_2, C;
Dec R_2;
Store C, R_2;

Scenario

P_1, P_2



t: P_1 : I, II, Pr
t_1: P_2 : I, II, III ; } "Inconsistency"
t_2: P_1 : III

Note: * Sometimes we get correct results and other times it is possible to get incorrect results.
* As an end user we want a solution, that always gives correct result weather pre emption take place or not.

Implementation of producer Consumer problem

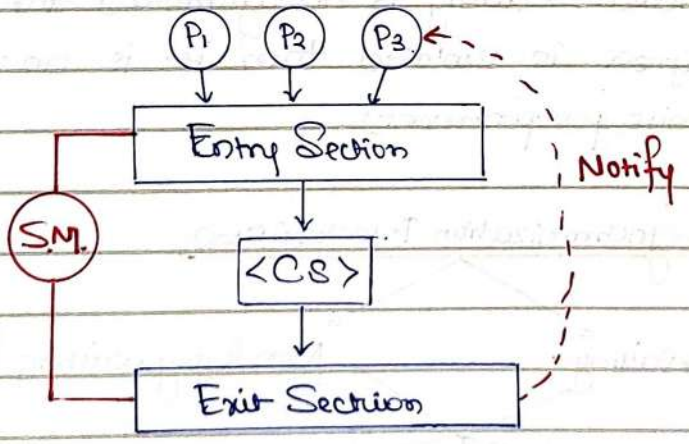
```
#define N 100
int buffer[N];
int count = 0; <No. of data items in buffer>
```

```
Void Producer (void)
{ int itemp; in = 0;
  while (1)
  {
NCS a> itemp = Produce-item();
 CS b> while (count == N); Busy waiting
 CS c> Buffer [in] = itemp;
 CS d> in = (in+1) % N;
 CS e> count = count + 1; }
 }
```

```
Void Consumer
{ int itemC, out = 0;
  while (1)
  {
  a> while (count == 0);   Busy waiting
  b> itemC = Buffer [out];
  c> out = (out + 1) % N;
  d> count = count - 1;
  e> Process-item(); }
 }
```

Necessary conditions for Synchronization Problems

1. Critical Section: It is that part of the program where shared resources are accessed
2. Race conditions: Situation where in processes are trying to access critical section and final result depends on the order at which they finish their update.
3. Pre emption:




```
Do {
  Entry Section
    <Critical Section>
  Exit Section
    <Remainder Sect>
 }
 while (true);
```

Note: Process running in user mode can get prempted any time and any number of times (after completing any instance).

Date 7/02/23

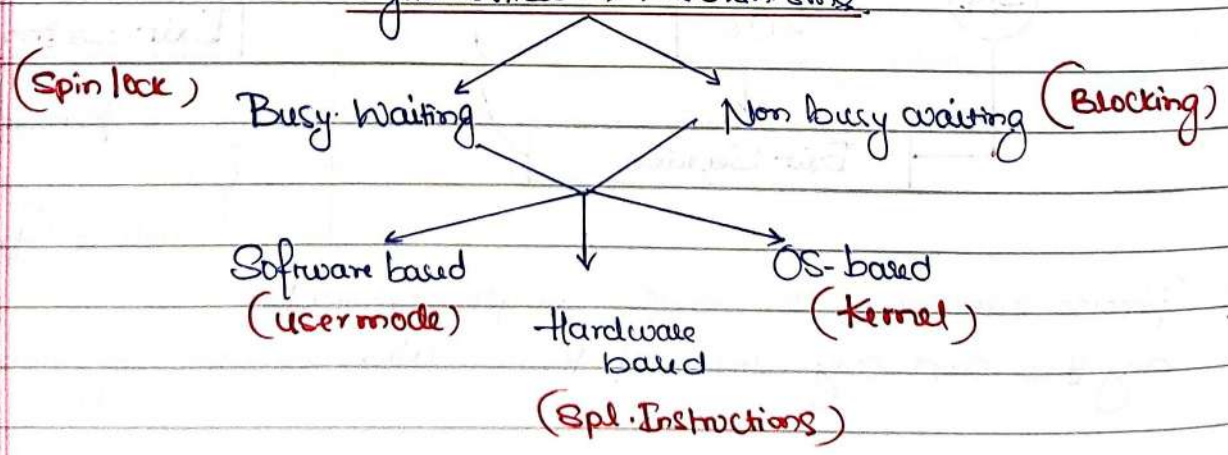
Requirements of C.S. Problems:

1. **Mutual Exclusion (M/E)**: If process Pi is executing in its Critical Section then no other processes can be executing in their Critical Sections.

2. **Progress**: If no process is executing in its critical sections and some processes wish to enter their Critical Sections, then only those processes that are not executing in their remainder Sections can participate in deciding which will enter its Critical Section next and this selection cannot be postponed indefinitely.

3. **Bounded waiting**: There exist a bound, or limit, on the number of times that other processes are allowed to enter their Critical Sections after a process has made a request to enter its Critical section and before that request is granted.

Note: * If mutual exclusion is not guaranteed then inconsistency + data loss may happen.
* If bounded waiting is not guaranteed then starvation.
* If progress is violated then it is an unfair Solution (Indefinite postponement).

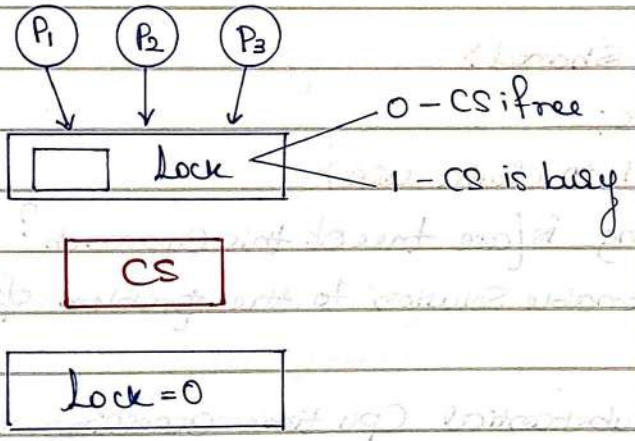


Assumptions:

* Process enters <CS> and come out of it in finite amount of time.
* When a process is in entry section then it means, it is intrested in <CS>.
* A process is said to have left CS only when it has executed its Critical Section.
* A process can get pre empted from CPU while executing entry + <CS> + Exit Section.

1. Lock Variable:

- Busy waiting Solution
- Software Solution implementation at user mode
- Multi process Solution.



Implementation : High L.

```
int lock = 0;
void process (int i)
{ while (1)
  { a> NON-CS
    b> while (lock != 0); } Entry Section
       lock = 1;
    c> CS
    d> lock = 0; } Exit Section.
  }
```

Implementation : Low Level.

```
integer lock = 0;
Process (int i):
a> NON-CS
b> Load Ri, lock;
c> Compare Ri, #0;
d> JNZ Step b;
e> Store lock, #1;
f> <CS>
g> Store lock, #0
```

Entry Section. / Exit Section

* Lock variable does not always gurantee mutual exclusion along with bounded wait.

Date 8/2/23

Note: * Lock variable does not guarantee mutual exclusion always.
* Lock variable guarantees progress always
* Lock variable fails to satisfy bounded wait.

* Because a process can enter CS multiple times successfully while other processes are waiting for their turn to enter CS.
* Lock variable is a busy waiting solution leading to wastage of CPU time/cycles (in the loop).

Q1. Several concurrent processes are attempting to share an I/O device. In an attempt to achieve M/E each process is given the following structure.

```
<code unrelated to device use>
    Repeat
    until busy = false;
    Busy = true;
<code to access shared>
    Busy = false;
<code unrelated to device use>
```

Which of the following is/are true of this approach?

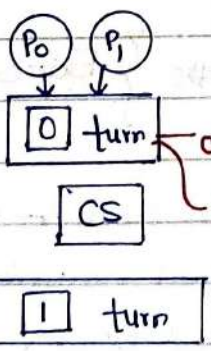
I. It provides a reasonable solution to the problem of guaranteeing M/E

II. It may consume substantial CPU time accessing the Busy variable.

III. It will fail to guarantee mutual exclusion.

2. Strict Alteration:

- Busy waiting
- Software solution implementable at user mode
- Two process solution <Pi/Pj>



"Strictly on alternate basis processes take turn to enter CS."

Implementation: High Level.

```
int turn = rand(0,1);
void process(0)
{
  while(1)
{ a> NON-CS();
  b> while(turn!=0)
  c> CS
  d> turn=1; }
}
```

```
void process(1)
{
  while(1)
{ a> NON-CS();
  b> while(turn!=1);
  c> CS
  d> turn=0; }
}
```

* Strict alteration always guarantee mutual exclusion.
* Strict alteration does not guarantee Progress.
* Strict alteration always guarantee bounded waiting.
* Like lock variable this is a busy waiting solution which leads to wastage of CPU time/cycles.

Generalised Implementation of Strict Alternation:

```
int turn = rand(i,j);
void process(int i)
{ int j= NOT(i);
  while(1)
{ a> NON-CS();
  b> while(turn!=i)
  c> CS
  d> turn=j; }
}
```

Q2 Consider the following two-process synchronization solution.

Process 0

Entry: loop while (turn == 1);

(Critical section)

Exit: turn = 1;

Process 1

Entry: loop while (turn == 0);

(Critical section)

Exit: turn = 0;

The shared variable turn is initialized to zero. Which of the following is true?

Ans: This solution violates progress requirement.

Q3. Code Snipped:

```
int turn = 0;
void Process(0)
{ while (1)
  { a> NON-CS();
    b> while (turn == 1);
    c> CS
    d> turn = 1; }
}
```

```
void process(1)
{ while (1)
  { a> NON-CS();
    b> while (turn == 1);
    c> CS
    d> turn = 1; }
}
```

Does the above code guarantees mutual exclusion?

Ans No.

* If turn = 1; then it will result in deadlock as no process can enter in the Critical Section.

Q4.

```
int turn = rand(i,j);
Void Process (int i)
{ int j = NOT (i);
  while (1)
  { a> NON-CS();
    b> while (turn != i);
       turn = j;
    c> CS
    d> turn = j; }
}
```

i) Does this guarantee M/E?
M/E is not guaranteed.

ii) Progress?

iii) Bounded Wait?

3. Peterson's Solution:

- Busy-waiting
- Software Solution implemented at user mode
- Two process Solution
- Combination of lock & Strict alternation.

```
#define N 2
#define True 1
#define False 0
int flag[N] = {false};
int turn;
```

```
Void Process (int i)
{ int j = NOT(i);
  while(1)
  { a> NON-CS();
    b. flag[i] = True;
    c. turn = i;
    d. while (flag[j] = True and turn == i);
    e. CS
    f. flag[i] = False; }
}
```

Note: Peterson's Solution guarantees M/E always.

* It guarantees progress
* It guarantees bounded waiting.

Date 9/2/23

Q5.

```
void Dekkers-Algorithm(int i)
{
  int j = !(i);
  while(1)
{ a> NON-CS();
  b> flag[i] = True;
  c> while (flag[j] == True):
    { if (turn == j)
        { flag[i] = False;
          while (turn == j);
          flag[i] = True }
    }
  d> CS
  e> flag[i] = False;
    turn = j;
}.
```

* Deckers algorithm is a variation of petrison's algorithm and is a correct solution.

Analyse the solution.

Q6 Process P_1 and P_2 hold/use Critical flag in the following routine to achieve M/E. Assume that Critical-flag is initialised to false in main program.

```
get-exclusive-access()
{  if (Critical-flag == False)
    { Critical-flag = True;
      Critical-region();
      Critical flag = False; }
}
```

Consider the following statements

I. It is possible for both P_1 and P_2 to access critical region concurrently.

II This may lead to a deadlock

Ans I. is true and II is false..

Q7. Two processes P_1 and P_2 need to access a critical section of the code. Consider the following synchronization construct used by the process.

```
/P1/
while (true)
{ wants1 = true;
while (wants2 == true);
/CS1/
wants2 = false; }
/Remainder Section/
```

```
/P2/
while (true)
{ wants2 = true;
while (wants1 == true);
/CS1/
wants2 = false; }
/Remainder Section/
```

Here wants1 and wants2 are shared variables which are initialised to false. Which of the statements are true.

I. It does not ensure M/E.

II. It does not ensure bounded waiting.

III. It requires that processes enter critical section in strict alternation

IV. It does not prevent deadlock but ensures M/E.

Q8. Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both.

```
Process X
while (true)
{ var P = true;
while (varQ = true)
{ /CS1/
varP = false; }
}
```

```
Process Y
while (true)
{ varQ = true;
while (varP == true).
{ /CS1/
varQ = false; }
}
```

Here varP and varQ are shared variables and both initialised to false. Which of statements are true?

I. The proposed Solution prevents deadlock but fails to gurantee M/E.
II. It gurantees M/E but fails to prevent deadlock.
III. It gurantees M/E and prevents deadlock.
IV. It fails to prevent deadlock and M/E.

Synchronization Hardware:

Each processer supports Some Support instructions (H/W) that are atomic <Executive non preemptively>
a> TSL
b> SWAP.

→ Busy waiting
→ Can be used at user mode as Spl. instructions
→ Hardware Category
→ Multi process Solution
→ Lock based solution.

a> TSL : Test and Set Lock

TSL (&Lock); < Process executing TSL, returns the current values of lock and sets the value of lock always to True >

```
Bool TSL (Bool *target)
{
    Bool rv;
    rv = *target;
    *target = True;
    return (rv); }
```

Completly runs on Kernel mode (Atomic)

Solving CS problem through TSL

```
Bool lock = false;
void Process (int i)
{  while (1)
   {  a> NON-CS();
      b> while (TSL(&Lock) == True);
      c> <CS>
      d> Lock = F; }
   }
```

* This solution gurantee mutual exclusion.
* Does not gurantee bounded wait.

b> SWAP Instruction <ATOMIC>

```
SWAP (Bool *a, Bool *b)
{  Bool t;
   t = *a;
   *a = *b;
   *b = t; }
```

* Gurantees mutual exclusion
* Gurantees progress
* Gurantees bounded waiting.

User process :

```
Bool lock = False;
void Process (int i)
{ Bool key = True;
  while (1)
  { a>. NON-CS();
    b>. while (key == True)
        SWAP (&lock, &Key);
    c> CS
    d> Lock = False; }
}
```

Q9. The enter-CS(); and leave-CS() functions to implement Critical Section of a process and are realised using test-and-set instructions as follows:

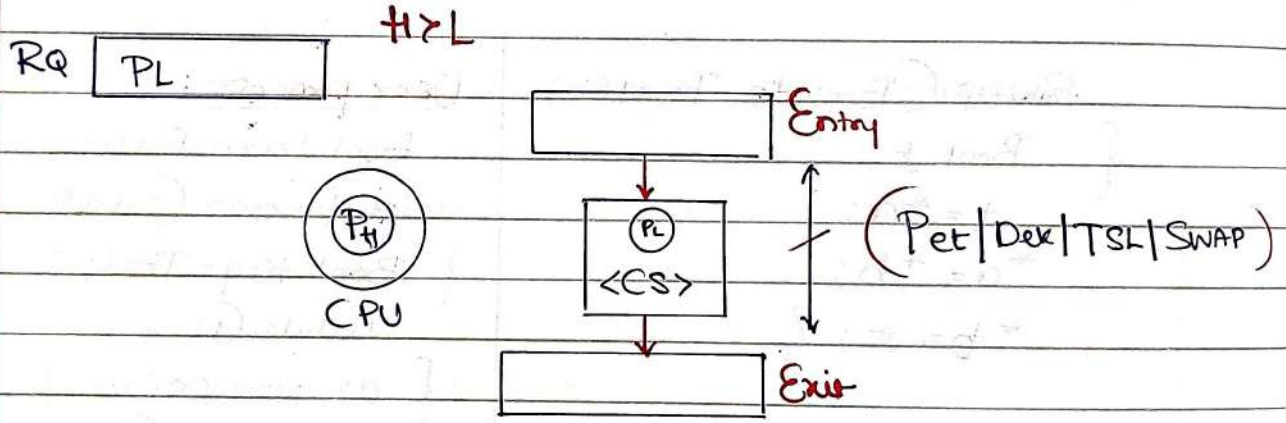
```
void enter_CS(x)
{ while (test_and_set(x));
void leave_CS(x)
  { x=0; }
```

In the above solution, x is a memory location associated with the CS and is initialised to 0. Now consider:

I. The above solution to CS problem is deadlock free.
II. Solution is starvation free.
III. The process enters CS in FIFO order
IV. More than one process can enter CS at same time

Priority Inversion Problem

All correct busy waiting software solutions suffers from priority inversion problem. This problem happens on pre emptive priority based scheduling



* If P_H is not interested in CS then there is no problem
* If P_H is also interested in CS then it results in "Deadlock"

imp

Q10. Fetch-And-Add (x,i) is a atomic Read-modify-Write instruction that reads the value of memory location x, increments it by the value i and returns the old value of x. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialised to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock(L)
{ while (Fetch-And-Add(L,1))
    L=1; }
ReleaseLock(L)
  { L=0 }
```

This implementation:

I. fails as L can overflow

II. fails as L can take on a non-zero value when lock is actually available

III. works correctly but may starve some process.

IV. works correctly without Starvation.

Analasys:

RQ. $P_1 P_2 P_3$

ϕX_2 Lock

P_1.

Cs

```
int lock=0

Entry  while (Fetch-And-Add(&lock,1))
         L=1;
       <CS>
Exit   L=0;
```

* If MSQ asked, I. also correct in rare cases

I:

t_1 : P_1 : F_A_A : → 0

t_2 : P_2 : F_A_A : → 01

II:

t_1 : P_1 : f_A_A : 0 : <CS> ... P_1

t_2 : P_2 : F_A_A : 1. P_2 <L=1>

t_3 : P_1 : <CS>

t_4 : P_2 : L=1, FAA=1

Blocking Mechanisms / Non busy waiting

- to avoid wastage of CPU time in the form of loops.
- all blocking mechanisms are implemented using "if-else-then".

There are three mechanisms: 1. Sleep-wakeup
2. Semaphores
3. Monitors.

1. Sleep() and Wakeup():

- Blocking construct
- Multi process solution
- Are OS primitives
- When process executes Sleep() - gets blocked till some other process wakes it up (wakeup())

Producer - Consumer using Sleep() and Wakeup()

```
#define N 100
int buffer[N];
int count = 0;

void Producer(void)
{
    int itemp, in=0;
    while(1)
    {
    a>. itemp = produce_item();
    b>. if (count == N) Sleep();
    c>. Buffer[N] = itemp;
    d>. in = (in+1) % N;
    e>. count = count + 1;
    f>. if (count == 1) wakeup(consumer);}
    }

void Consumer(void)
{   int itemC, out=0;
    while(1)
    {   a> if (count == 0) Sleep();
        b> itemC = Buffer[out];
        c> out = (out+1) % N;
        d> count = count - 1;
        e> if (count == (N-1)) wakeup(Producer);
        f> Process_item(itemC); }
}
```

* When producer and consumer both try to update the value of count, leads to "inconsistency"

Two problems with this implementation
i. Inconsistency
ii. Deadlock.

Semaphores

* Blocking construct
* OS based <is an OS resource (S/W)>.
* General purpose utility
  { - <CS>
    - Requirements of classical IPC problems
    - Concurrency mechanisms }.
* It was proposed by E. Dijkstra
* Semaphore is implemented as an abstract data type.

Definition: Semaphore is a variable <ADT: SEM> that takes only integer values.

Operations
- down() / wait() / P()
- up() / Signal() / V()

Types.

1. Binary (mutex) <0, 1>
2. Counting (General) <-∞ to ∞>

Counting Semaphore <-∞ to ∞)

```
Typedef
    Struct
{   int values;
    QueueType L;
} CSME;
```

List of PCBs of those processes that get blocked while performing down operation unsuccessfully.

UM:

```
CSEM S;
S.value = 1;
(S=1);
DOWN(S);
<Si>
```

Next Statement.

Km:

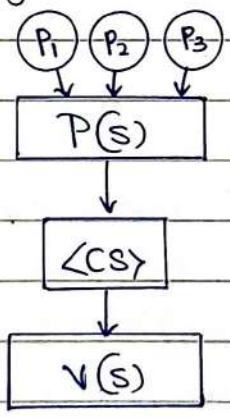
```
DOWN(CSEM S)
{
 1. S.value = value - 1;
 2. if (S.value < 0) // unsucess
    { put this process PCB
      in the S.L(Q) and block
      the process (Sleep); }
    else
      return; } // success.
```

eg: CSEM S;
S = 5;
How many DOWN operations can be carried out by process sucessfully?

S = 4 3 2 1 0 -1.
1 1 1 1 1 | 1 1.

Sucessful down opn's. — after this process get blocked

* Positive value of semaphore indicate that those many down operations can be carried out sucessfully.
* After carrying out a series of down operations, if the value of semaphore becomes negative, then the magnitude of the negative value indicate the number of blocked processes.




```
CSEM S;
S = 1;
```

```
UP(CSEM S)
{
 1. S.value = S.value + 1;
 2. if (S.value < 0)
    { Select a process from
      S.L(Q) and wakeup(); }
    else
      return; }
```

Block Queue.
~~P2 P3~~.

S = 1; 0 -1 -2 -1 0 1.
blocked (down) . (up)

* If the value of S is less than or equal to '0' then there are blocked processes in queue.

Q1. CSEM S = 8;

Operations : 10P, 1V, 15P, 2V, 6P, 3V What is current value?
-2 -1 -16 -14 -20 -17.

Current value of S = 17.

Q2. CSEM S = ?

Operations : 12P, 4V, 6P, 3V, 8P, 1V, Final value is -6.

What should be initial value of S?

Initial value of S should be 12.

Q3. Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 20 P(S) operations and 12 V(S) operations are issued in some order. The largest initial value of S for which atleast one P(S) operation will remain blocked 7

Binary Semaphore <0,1>

```
Typedef
    Struct
{ enum value (0,1);
  QueueType L;
} BSEm;
```

List of PCBs that gets blocked while performing down operation unsuccessfully.

VM:

```
BSEm S;
S=1;
DOWN(S);
<Nxt Strnt>
```

```
DOWN (BSEm S)
{ if (S.value == 1)
  { S.value = 0; // Sucess
    return; }
else
  { put this process (PCB)
    in S.L(Q) and
    block it (sleep();); }
}
```

* In binary semaphore we don't get to know the no. of blocked processes because no negative numbers after down operations.

```
UP(BSEM S)
{
  if (S.L(Q) is NotEmpty)
  {
    Select a process from
    S.L(Q) and wakeup(); }

  else
      S.value = 1; }
```

Cases of Binary Semaphores.

1.
```
BSEM S;
S=1;
P(S);
```
S=0
Status = Sucess.

2.
```
BSEM S;
S=0;
P(S);
```
S=0
Status = Unsuccessful.

3.
```
BSEM S;
S=0;
V(S);
```
S → 0 : if 'Q' not empty
S → 1 : if 'Q' empty.
Status = Sucess.

4.
```
BSEM S;
S=1;
V(S);
```
S=1
Status = Sucess.

5.
```
BSEM S;
S=0;
V(S); // First operation.
```
S = 1.
Status = Sucessful.

Note:

As long as there is a process in "CS" & processes in block 'Q' then value of BSEM must be '0'.

Q4 BSEM S=1;
Operations: 10P, 2V, 18P, 3V, 4P, 5V;
(i) Final value of S?
(ii) Size of 'Q'[L] ?

S = ~~X~~ 0

10 P, 2V, 18V, 3V, 4P, 5V;

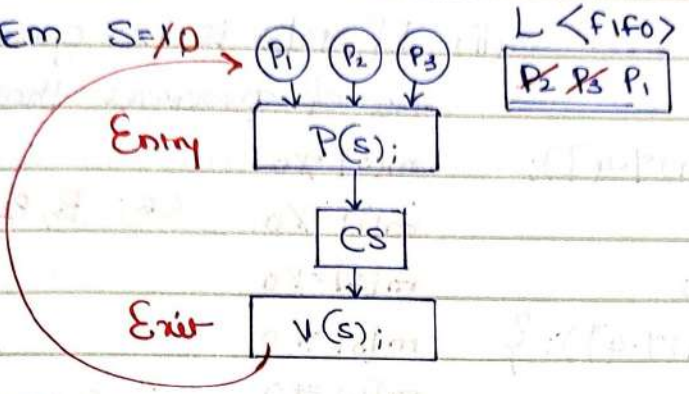
L → ~~9 7 2 5 2 3 2 6~~ 21

Final value of S = 0
and 21 blocked processes.

Date 10/2/23

Both binary and counting semaphore can be used to solve the problem of CS.

eg: BSEM S=1



L <FIFO>
~~P2~~ ~~P3~~ P1

1) Mutual Exclusion ✓
2) Progress ✓
3) Bounded wait ✓

Q1. BSEM S=1, T=1;

```
Pr(i)                    Pr(j)
{ while(1)               { while(1)
  { P(S);                  { P(T);
    P(T);  <- To prevent deadlock
    <CS>                     P(S);
    V(T);                    <CS>
    V(S); }                  V(S);
  }                          V(T); }
                           }
```

Does it gurantee ME? and deadlock?

Yes it gurantees ME but it is not deadlock free.

To prevent it from deadlock Swap down operations in one process. Both processes down opn. should be same.

Q2. BSEM S=1, T=0

```
Pr(i)                    Pr(j)
{ while(1)               { while(1)
  { P(T);                  { P(S);
    Print('1');              Print('0');
    Print('1');              Print('0');
    V(S); }                  V(T); }
  }                        }
```

What will be output?

00110011....

Does it gurantee
1. ME ✓
2. Progress ✗
3. Bounded wait ✓

This is a implementation of "strict alternation" using semaphores

Q3. BSEM m[0...4] = {1};

```
Pr(i) i=0,4
{
  while(1)
  { P(m[i]);
    P(m[(i+1)%4]);
    <CS>
    V(m[i]);
    V(m[(i+1)%4]); }
}
```

(i) Does it guarantee m/E?

No

(ii) If m/E is not guaranteed then max no. of processes that can be in CS?

m[0] = X0
m[1] = X0
m[2] = X0
m[3] = X0
m[4] = X0

CS: P0, P2. P1, P3, P4 } blocked

∴ 2 processes in CS

* Deadlock can occur.

Q4. BSEM = 1, T=0, Z=0

```
Pr(i)
{ while(1)
  { P(S);
    Print(*);
    V(T);
    V(Z); }
}

Pr(j)
{ P(T);
  V(S); }

Pr(k)
{ P(Z);
  V(S); }
```

What is min and max no of * that get printed?

Case 1: Pri, Prj, Prk, Pri blkd

I: <* *> : minimum → 2

Case II: Pri, Prj, Pri, Prk, Pri

II: <* * *> maximum → 3.

Variation: BSEM S=1, T=0, Z=0;

```
Pri
{ while(1)
  { P(S);
    Print('*');
    V(T); }
}

Prj
{ P(T);
  V(Z); }

Prk
{ P(Z);
  V(S); }
```

In this variation the min and max no of stars to be printed = 2;

Qs. Consider the following processes T_1, T_2, T_3 executing on a single processor, synchronized using three binary semaphore variables S_1, S_2 and S_3, operated upon using standard wait() and signal(). The threads can be context switched in any order and at any time. Which.

Which initialization of semaphores would print sequences

BCABCABCA... ?

T_1

```
while (true)
{ wait(S3);
  Print("C");
  Signal(S2); }
```

T_2

```
while (true)
{ wait(S1);
  Print("B");
  Signal(S3); }
```

T_3

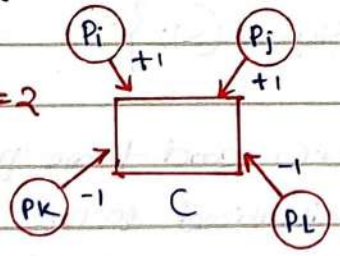
```
while (true)
{ wait(S2);
  Print("A");
  Signal(S1); }
```

Ans: $S_1 = 1$, $S_2 = 0$, $S_3 = 0$

$T_2 \rightarrow T_1 \rightarrow T_3$

HW

Q6 Consider a counting semaphore initialized to 2, there are 4 concurrent processes P_i, P_j, P_k and P_L, the processes P_i and P_j desire to increment the current value of variable C by 1 whereas P_k and P_L desire to decrement the value of C by 1. All processes perform their update on C under semaphore control. What can be the min and max value of C after all processes finish their update?



Assuming initial value of C != 0

```
Process(x)
{ P(S);
  <C = C ± 1>
  V(S); }
```

x = i, j, k, l

+1 +1 -1 -1

Q7. Consider three processes using four binary semaphores a,b,c,d in the order shown below.
Which is sequence is a deadlock free sequence?

X : P(b); P(a); P(c);
Y : P(b); P(c); P(d);
Z : P(a); P(c); P(d);

a=b=c=d=1. } Initialized

Q8. Each of a set of n processes executes the following code using two semaphores a and b initialized to 1 and 0 resp. Assume that count is a shared variable initialized to 0 and not used in code section P.

```
<Code Section P>
    wait(a);
    count = count + 1;
  if (count == n) signal(b);
  signal(a); wait(b); signal(b);
<Code Section Q>
```

What does code achieve?
It ensures that no process executes code section Q before every process has finished code section P.

HW

Q9. Consider two functions Incr() and Decr() GATE 2023

```
Incr()                    Decr()
{ wait(s);      } 5       { wait(s);      } 3.
  x = x + 1;              x = x - 1;
  signal(s); }            signal(s); }
```

Five processes invoke Incr() and three processes invoke Decr()
x is a shared variable initialized to 10. (10+5-3)=12

I1 : S value is 1 (Bin. Sem.) max/min = 12
I2 : S value is 2 (counting. Sem.)

Let v1 and v2 be min possible values of implementation of I1 and I2 then choose the value of x for v1 and v2.

Date 11/2/23

Q1. Which of the following may result in a process getting blocked?

a) DOWN c) System-call

b) UP. d) Scheduler dispatch.

Classical IPC Problems

1. Producer Consumer Problem:

```
#define N 100
int Buffer[N];
CSEM Empty = N;
<No of empty slots>
CSEM full = 0;
<No. of full slots (data-items)>
BSEM mutex = 1;
<used b/w P and C to
ensure M/E on buffer>
```

```
void Producer (void)
{ int temp, in = 0;
  while (1)
  { a). itemp = produce-itemC();
Entry { b). DOWN (Empty);
      { c). DOWN (mutex);
    < d). Buffer[in] = itemp;
      e). in = (in+1) % N; >
Exit { f). UP (mutex);
     { g). UP (full);
```

```
void Consumer (void)
{ int itemC, out = 0;
  while (1)
  { a). DOWN (full);
    b). DOWN (mutex);
    c). itemC = Buffer[out];
    d). out = (out+1) % N;
    e). UP (mutex);
    f). UP (empty);
    g). Process-item (itemC); }
  }
```

Q1. Consider the following solution to the P-C Sync. problem. Shared buffer size is N. Three Semaphores empty, full and mutex are declined with respective initial values of 0, N and 1. Semaphore empty denotes the no. of available slots in the buffer, for Consumer to read from. Semaphore full denotes no of available slots in the buffer, for producer to write to. the placeholder variables denoted by P, Q, R, and S in code below can be assigned either empty or full.

```
Producer:                          Consumer:
  do{                                do{
     wait(P);                           wait(R);
     wait(mutex);                       wait(mutex);
    // add item - buffer               // Consume item - buffer
     signal(mutex);                     signal(mutex);
     signal(Q); }                       signal(S); }
  while(1);                          while(1);
```

Which of the assignment of P, Q, R, and S yeild correct solution?

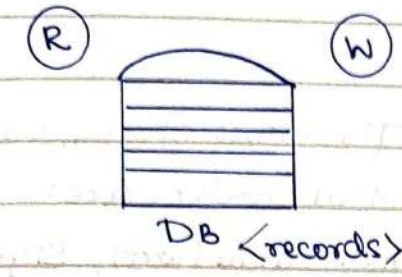
Ans P: full, Q: empty, R: empty, S: full

Q2. Consider the procedure below for P-C problem uses Semaphores:

```
Semaphore n=0;
Semaphore s=1;
void Producer()                    void Consumer()
{  while(true)                     {  while(true)
   {  produce();                      {
      SemWait(s);                       SemWait(s);
      addToBuffer();                    SemWait(n);
      SemSignal(s);                  removeFromBuffer();
      SemSignal(n); }                   SemSignal(s);
   }                                    consume(); }
}                                  }
```

Ans. Deadlock occurs when/if the Consumer succeeds in acquiring Semaphores when buffer is empty.

2. Reader Writer Problem.




```
BSEM mutex=1;
{
  P(mutex)
    <DB>
  V(mutex); }
```

First Reader Writer Problem.

t_1 : r_1 and w_1 : Anyone
t_2 : r_1 ; ✓ w_1 (wait)
t_3 : r_2 ; ✓
t_4 : r_3 ; ✓

r_1 r_2 r_3

t_1 : r_1 and w_1 : Anyone
t_2 : w_1 ; ✓ r_1 (wait)
t_3 : w_2 ; ✗ (wait)

w_1

Problem: Starvation to writers.

Implementation of First R-W using Semaphore

```
int rc=0;
BSEM mutex=1;
<used by Rs to update>
BSEM db=1;
<used by Rs and Ws to
   access <DB> as CS>

void writer (void)
{
  while (1)
  { a>. DOWN(db);
    b>. <DB-write>
    c>. UP (db); }
  }
```

```
void Reader (void)
{ while (1)
  {
    a>. DOWN(mutex);
    b>. rc = rc+1;
    c>. if (rc==1) DOWN(db);
    d>. UP (mutex);
    e>. <DB-Read>
    f>. DOWN(mutex);
    g>. rc = rc-1;
    h>. if (rc==0) up(db);
    i>. UP(mutex); }
  }
```

Q3. Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores

mutex and wrt are used to obtain sync.

```
wait(wrt)
// waiting is performed
signal(wrt)
wait(mutex)
readCount = readCount + 1
if readCount == 1, then "S1"
   "S2"
// reading is performed
   "S3"
readCount = readCount - 1
if readCount = 0 then "S4"
signal(mutex)
```

The values of S1, S2, S3, S4 in that order are:

Ans: wait(wrt), signal(mutex), wait(mutex), signal(wrt).

3. Dining Philosophers Problem

'N' - Philosophers ($N \geq 2$)

If all become hungry causes "Deadlock"

```
#define N 5
void Philosopher(int i)
{  while(1)
   {  a> think(i);
      b> take-fork(i);
      c> take-fork((i+1)%N);
      d> eat(i);
      e> put-fork(i);
      f. put-fork((i+1)%N); }
}
```

Without deadlock, for N=5 what is the max no. of philosophers that can be eating?

P_0 : f_0, f_1 ✓

P_1 : × Ans: 2

P_2 : f_2, f_3 ✓

P_3 : f_4

P_4 : f_4, f_0 ×

Date 13/2/23

Prevent/Avoid Deadlock in Dining Philosophers

Semaphore

→ put semaphore control on taking and releasing the forks.

Non Semaphore based

→ out of 'N' philosophers,
Let (N-1) → L-R
and 1 → R-L

→ Even numbered : L-R
Odd numbered : R-L

Q4. A solution to the dining philosophers problem which avoids deadlock is:

Ensure that one particular philosopher picks up the left fork before right fork, and all other philosophers pick up right before left fork.

Q5. Let m[0...4] be mutexes and $P_0 - P_4$ be processes. Suppose each process P[i] executes the following

```
wait(m[i]); wait(m[i+1]%4)
<CS>
release(m[i]; release(m[i-1]%4)
```

Which situation could occur?

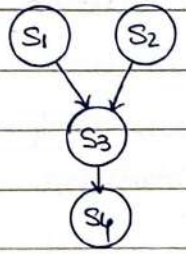
Ans: Deadlock.

Concurrency vs Sequentiality

```
{ S1: a = b+c;
  S2: d = e*f;
  S3: k = a+d;
  S4: l = k*10; }
```

I_1: Load R_1, b
I_2: Load R_2, c
I_3: Add R_1 R_2
I_4: Store a, R_1

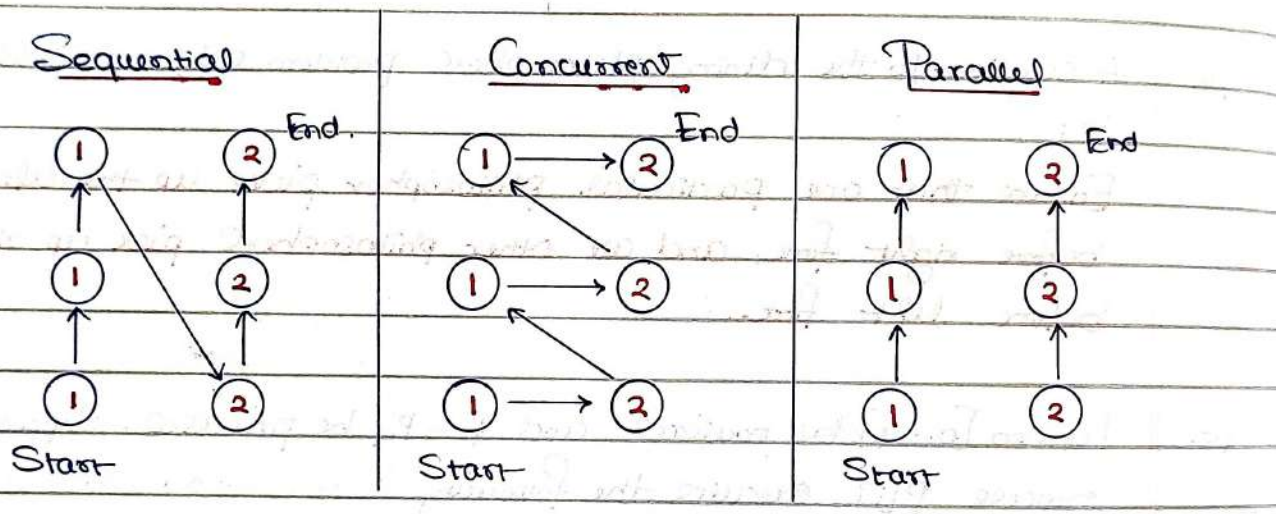
* Only S1 and S2 can perform concurrently (no dependency)



Precedence Graph (directed)

Si : High level language statement.

Concurrency	Parallelism
* A system is said to be concurrent if it can support two or more actions in progress at the same time	A system is said to be parallel if it can support two or more actions executing simultaneously
* Concurrency is about dealing with lots of things at once.	Parallelism is about doing lots of things at once.



An Analogy:

Concurrent : Two queues and one coffee machine

Parallel : Two queues and two coffee machines

Note:

* In Concurrency process will execute with preemption.
* Parallelism is possible with multiple CPUs / Cores.
* Concurrency is about structure, parallelism is about execution.
* Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

* The modern world parallel it has:
  1. Multi Cores
  2. Networks.
  3. Cloud of CPUs
  4. Loads of users.
* Concurrency makes parallelism easy.

Types of Concurrency

1. Pseudo (one CPU) → Interleaved execution among processes/applications.
2. Real/Physical (Parallelism) → Multiple processing units

Concurrency Conditions.

→ Si = a=b+c; Sj = d=e*f;

* Output of one statement should not serve as input to another

S — R(S) : Read set
  — W(S) : Write set

eg: (I) S : a=b+c
R(S) → b,c; W(S) → a.

(II) S : a+= ++b * --c;
R(S) and W(S) → a,b,c.

(III) S : scanf("%d", &x);
R(S) → ϕ
W(S) → x

(IV) printf(x);
R(S) → x
W(S) → ϕ

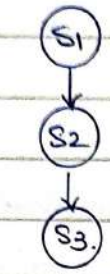
Bernstein's Concurrency Conditions: I. $R(S_i) \cap W(S_j) = \phi$
II. $W(S_i) \cap R(S_j) = \phi$
III. $W(S_i) \cap W(S_j) = \phi$
IV. $R(S_i) \cap R(S_j)$ = may/may not ϕ

Concurrency mechanisms

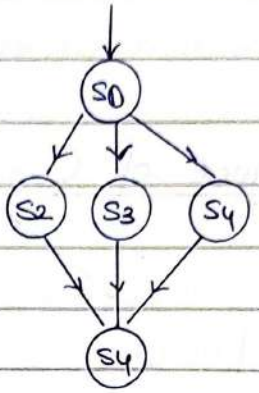
1. Parbegin - Parend / Cobegin - Coend :

```
{ begin
    S1;
    S2;
    S3;
  end }
```

Sequential

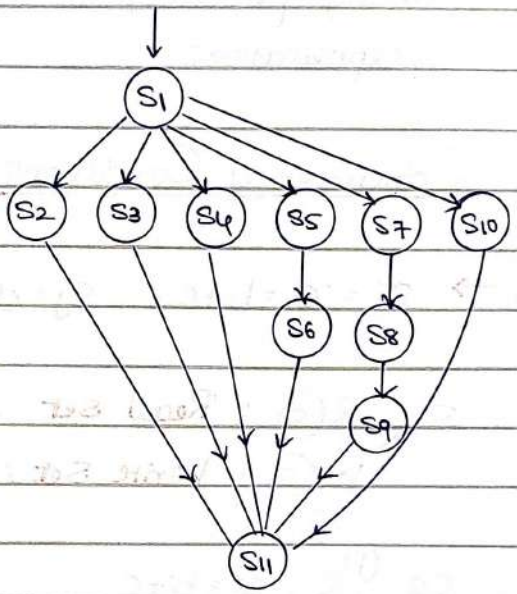



```
S0;
Parbegin
    S1;
    S2;
    S3;
Parend
    S4;
```

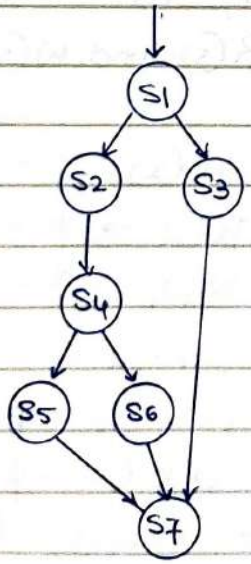


eg:

```
S1;
Parbegin
    S2; S3;
    S4;
    begin S5; S6; end
    begin S7; S8; S9; end
    S10;
Parend
    S11;
```



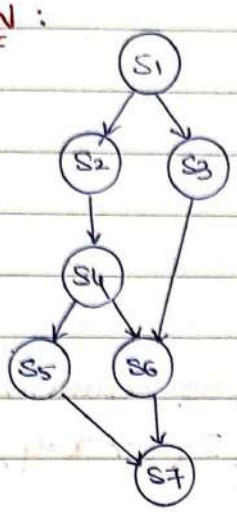
eg:




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

How many times parallel construct to be used?

HW :



S1; BSEM a,b,c,d,e,f,g = {0}

Cobegin

begin S1; V(a); V(b); end

begin P(a); S2; S4; V(c); V(d); end

begin P(b); S3; V(e); end

begin P(c); S5; V(f); end

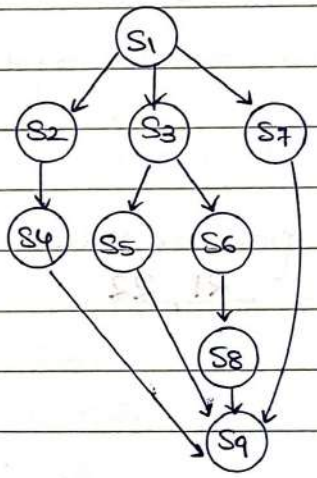
begin P(d); P(e); S6; V(g); end

begin P(f); P(g); S7; end

Coend

Q1. Draw the precedence graph for the Concurrent Program.

```
S1;
Parbegin
    begin
        S2; S4; end
    S3;
    Parbegin S5;
        begin S6; S8; end
    Parend
    end.
    S7;
Parend;
S9;
```



Q2.

```
void P(void)        void Q(void)        void main()
{ A;                { D;                { Parbegin
  B;                  E; }                  P(); Q();
  C; }                                    Parend; }
```

Valid output Sequences: ABCDE

DEABC, ADBEC.

Q3. int x=0; y=0;

Cobegin

begin

S1: x=1;

S2: y=y+x;

end

begin

S3: y=2;

S4: x=x+3;

end

Coend.

Final values of x and y

I. x=1, y=2

II. x=1, y=3

III. x=4, y=6

Statements S3, S4, S1, S2 results in x=1, y=3.

Statements S1, S3, S4, S2 → x=4, y=6

Q4. int x=0, y=20;

Bsem mx=1; my=1;

Final possible values of x {21, 22}

```
Cobegin
  begin
    P(mx);
    x=x+1;
    V(mx);
  end;
  begin
    P(mx);
    x=y+1;
    V(mx);
  end
Coend
```

HW

Q5. integer B=2;

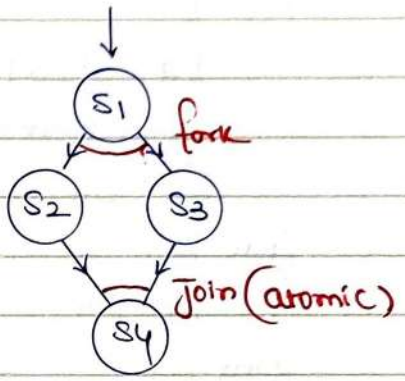
```
P1()
{ C=B-1;
  B=2*C; }

P2()
{ D=2*B;
  B=D-1; }

main()
{ Parbegin
    P1()
    P2() }
  Parend.
```

The no. of distinct values of B are _____

2. Fork join

Syntax of fork : fork L;

→ Execution of fork results in starting 2 computation concurrently
I. Which is immediately after fork
II. Which is at Label 'L'.

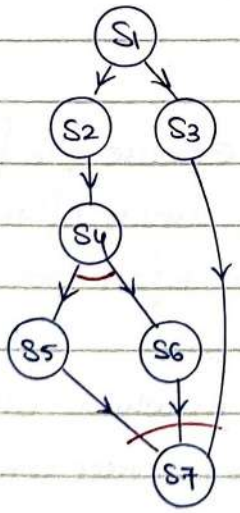


eg:

```
integer count=2;
S1;
fork L;
S2;
goto X;
L: S3;
X: Join (count);
S4;
```

```
Join (int count)
{ count = count-1;
  if (count ≠ 0) then exit;
  else
    return; }
```

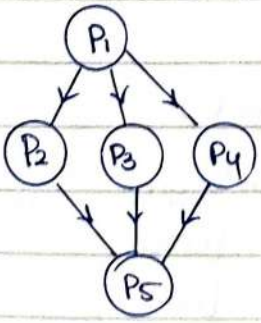
eg:




```
int count=3;
S1;
fork L;
S2; S4;
fork X;
S5;
goto Z;
```

```
L: S3;
   goto Z;
X: S6;
Z: Join (count)
   S7;
```

eg

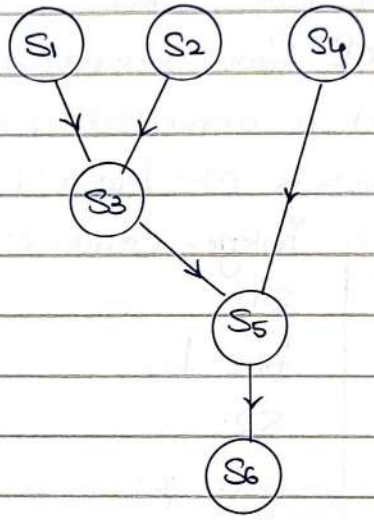



```
P1;
fork L;
P2;
L: fork X
   P3;
   goto Z;
```

```
X: P4
Z: Join (count);
   P5;
```

eg. N=2, M=2 Draw the precedence graph.

```
fork L3, L4 :
S1 ;
    L1 : Join N
S3; &
    L2 : Join M
S5 : goto next ;
    L3 : S2 ;
goto L1 ;
    L4 : S4 ;
goto L2 ;
next : S6 ;
```



Q1. Consider the following pseudo code where S is a semaphore initialized to five and counter is a shared variable initialized to 0 in line 1. Assume that increment operation in line #7 is not atomic.

```
1  int Counter = 0
   Semaphore S = init(5);
   void parop(void)
   {  wait(S);
      wait(S);
7     Counter ++;
      Signal(S);
      Signal(S);
   }
```

If five processes execute the function parop concurrently, which of the following program behaviour is/are possible?

~~I.~~ there is a deadlock involving all processes.
~~II.~~ Counter = 5 after all processes comp- parop
~~III.~~ counter = 1 " " "
IV. counter = 0 " " "

Q2.

```
int count=0; void test()
        {int i, n=5;
         for(int j=1; j<=n; ++i)
             count = count+1; }
```

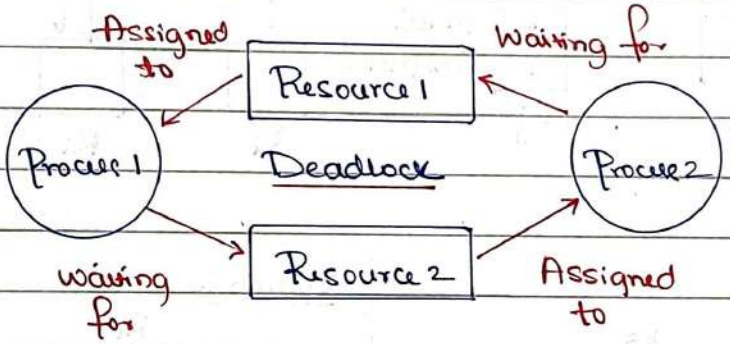
```
main()
{ Parbegin
    test();
    test();
  Parend }
```

Date 16/2/23

Deadlock

Deadlock: Two or more processes are said to be in deadlock if they wait for the happening of an event which will never happen.

Consequences:
* Throughput/Efficiency drops
* Ineffective utilization of resources

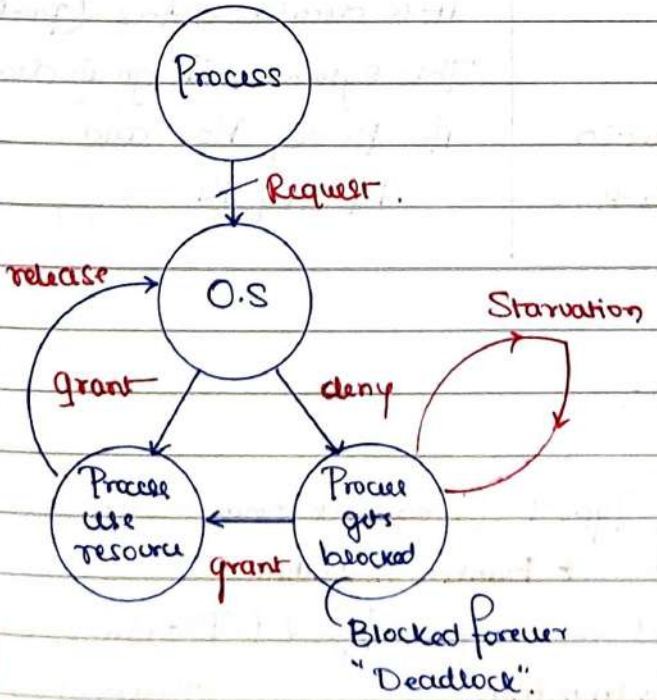


System Model:

→ n: no of processes $<P_1 \dots P_n>$

→ 'm': no. of resources $<R_1 \dots R_m>$

- Single instance
- Multi Instance

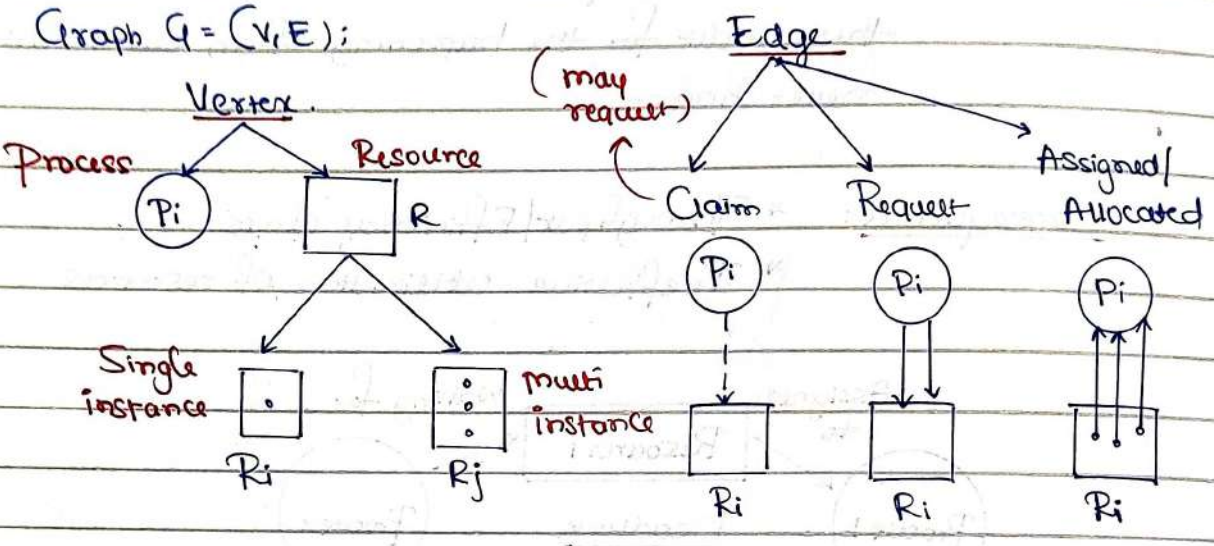


Necessary Conditions for Deadlock:

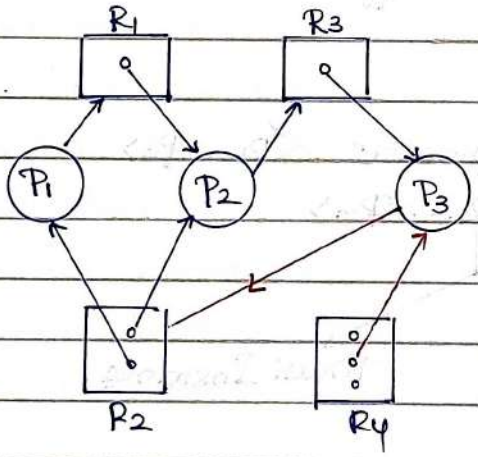
1. Mutual Exclusion
   → Shared resource <CS>
2. Hold and wait
   Every H&W is not deadlock but every deadlock will have H&W.
3. No pre-emption of resources
4. Circular wait

Resource Allocation Graph (RAG)

Graph G = (V, E):

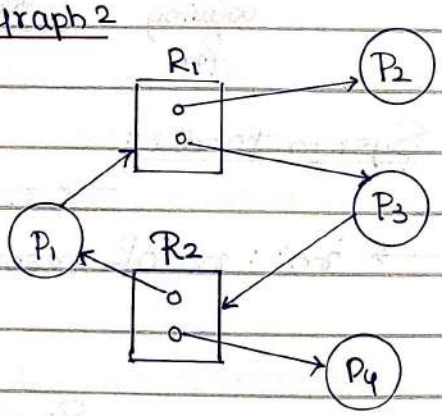


Graph 1.



It is deadlock free.
After changes it will result in deadlock (P_1-R_1-P_2-R_3-P_3-R_2-P_1) cycle

Graph 2



It is deadlock free (P_1-R_1-P_3-R_2-P_1)
This system will go in deadlock if P_3 req R_2 and P_4 req R_1

Deadlock Handling Strategies

1. Deadlock Prevention } Type 1 (Deadlock never occurs)
2. Deadlock avoidance → Banker's Algorithm.
3. Deadlock detection and recovery } Type 2 (Deadlock definitely occurs)
4. Deadlock Ignorance → Ostrich Algo

I. Deadlock Ignorance (Ostrich Algorithm)

* Pretend there is no problem (Ignorant Strategy of ostrich)
* Reasonable if : (i) deadlock occur very rarely
  (ii) cost of prevention is high.
* UNIX and Windows take this approach
* It is a trade-off between convinience and correctness.

II. Deadlock Prevention

Deadlock Prevention can be done by dissatisfying or negating the necessary conditions of deadlock.

a) mutual exclusion :→ Since every multi programmed Operating System has atleast one shared resource ∴ M/E is non dissatisfyable.

b) Hold and Wait ⟶ Hold or Wait implementation

(i) Process must request and be allocated all resources prior to its start.

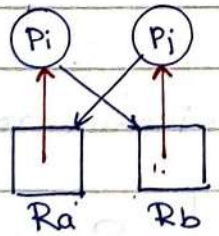
Drawbacks: Causes/leads to Starvation, inefficiency

(ii) Process must release all resources before making a fresh/new request.

Drawback: Suffer from Starvation

(iii)

c) No preemption: ⟶ pre emption of resources from processes.



forced (Running process should not block)

Self (allows other processes to Complete first)

Drawbacks: Causes Starvation.

d) Circular-wait :⟶ is prevented by a total order relation among all processes and resources.

(i) Assign unique numbers/Ids to each resource

(ii) Never allow a process to request a lower numbered resource than the last one allocated.

Drawback: Starvation Possible

Note :

(1.) If the R.A.G. has resources of only multi instance type, then cycle is only a necessary condition

(2.) If R.A.G has resource of both Single and multi instance, then presence of cycle is only a necessary condition for deadlock.

(3.) If the R.A.G has resources of only Single instance type, then cycle in a necessary and sufficient condition for deadlock.

III. Deadlock Avoidance

1. Single instance resource
<Resource allocation Graph algorithm>

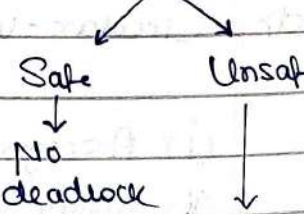
2. Multi instance of Resource (SI+M.I)
Banker's Algorithm
(i) Safety algorithm
(ii) Resource request algorithm

Note: Both the algorithms are based on Apriori Knowledge/ information.

1. Resource-Allocation Graph Algorithm <Single instance resource>

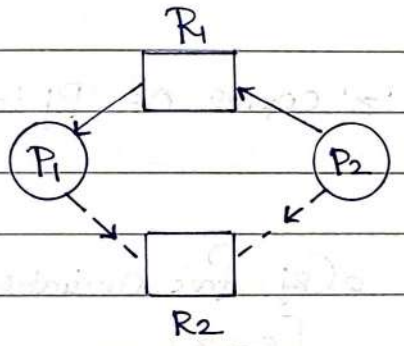
* Resources are claimed a priori in the system.
* If process Pi starts executing then all claim edges must appear in R.A.G.
* If Pi's request resource Rj, then the request is granted only if, converting the request edge to ~~claim~~ assigned edge does not lead to a cycle in R.A.G otherwise

System State



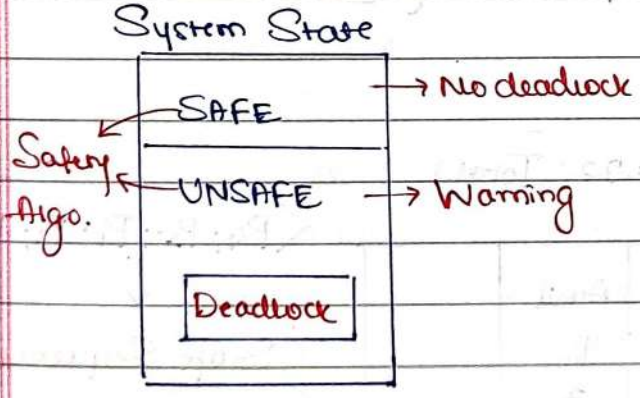
the process is blocked
<No cycle implies Safe State and formation of cycle implies Unsafe state>

* The basic objective of R.A.G algorithm (S.I) is to always operate the system is in SAFE State.
* System is said to be safe, if the conversion of request edge to assigned edge does not lead to cycle in R.A.G. Otherwise if it is leading cycle then it is unsafe mode.



→ Current State: Safe (no cycle)

→ What if P2 also request R2?



Banker's Algorithm (m. Instances)

(i) Safety algorithm (ii) Recource request algorithm

Data Structures (System state)

1. n : no of processes (P1 ... Pn)
2. m : no. of resources (R1 ... Rm)
3. maximum [1...n, 1...m]nxm :

max [i,j] = k

Pi ⟶ k (Rj)

Process Pi demands (apriori) 'k' copies of Rj'.

4. Allocation $[1..n, 1..m]_{n \times m}$: $Alloc[i,j] = a$
$P_i \leftarrow a(R_j)$ $[a \leq k]$

5. Need $[1..n, 1..m]_{n \times m}$: Need $[i,j] = b \rightarrow$ Need = max − allocation
$P_i \leftrightarrow b(R_j)$ $[b = k - a]$

6. Request $[1..n, 1..m]_{n \times m}$: $P_i \longrightarrow c(R_j)$ @ time 't'.
$Req[i,j] = c$. $[c \leq b]$

7. Total $[1...m]$
Total $[j]$ = 'z' → There are 'z' copies of R_j in the system.

8. Available $[1...m]$
Avail $[j]$ = 'e' There are e(R_j) free available @ time 't'.
$Avail = Total - \sum_{i=1}^{n} Alloc_i$ $[e \leq z]$

Example : n = 5, m = 1, R = 22 (Total)

Pid.	Max R	Alloc R	Need R	Avail R	
P_1	10	5	5	3	
P_2	8	4	4	5	
P_3	12	5	7	8	
P_4	5	2	3	13	
P_5	6	3	3	17	

<22>

<P_4; P_5; P_1; P_2; P_3;>
↓
"Safe Sequence"
(We can have multiple safe sequences)

Need = Max − Alloc

Safety Algorithm:

System is said to be "safe" if the need of all processes can be satisfied with available resources in some order, otherwise it is "unsafe."

Example 2: $\langle A, B, C \rangle = \langle 10, 5, 7 \rangle$ $\langle P_1; P_3; P_4; P_0; P_2 \rangle$

	Allocation A B C	Max A B C	Available A B C	Need A B C	
P_0	0 1 0	7 5 3	3 3 2	7 4 3	"System is Safe."
P_1	2 0 0	3 2 2	<5 3 2>	1 2 2	
P_2	3 0 2	9 0 2	<7 4 3>	6 0 0	
P_3	2 1 1	2 2 2	<7 4 5>	0 1 1	
P_4	0 0 2	4 3 3	<7 5 5>	4 3 1	
			<10 5 7>		

(ii) Resource request algorithm

Algorithm Res-request(Pi, Reqi, Alloci, Needi, Avail)

{

1. $Req_i \leq Need_i$
2. $Req_i \leq Alloc_i$
3. [Assume to have satisfied the Reqi]
   a. $Avail = Avail - Req_i$
   b. $Need_i = Need_i - Req_i$
   c. $Alloc_i = Alloc_i + Req_i$
4. Run Safety algorithm
5. If System is SAFE then grant the Reqi
   else deny the Reqi and block the process Pi; }

HW

Processes	Allocation A B C D	Max A B C D	Available A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

Q1. An OS uses the banker's algorithm for deadlock avoidance when managing the allocation of three resources types x, y and z to the processes P_0, P_1 and P_2. The table given below presents the current system state. Here the allocation matrix shows the current no. of resources of each type allocated to each process and max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	x	y	z	x	y	z
P_0	0	0	1	8	4	3
P_1	3	2	0	6	2	0
P_2	2	1	1	3	3	3

There are 3 units of type x, 2 units of type y and 2 units of z still available. System is currently in safe state. Consider the following independent requests for additional resources in current state.

Req 1: P_0 requests 0 units of x, 0 units of y and 2 units of z

Req 2: P_1 requests 2 units of x, 0 units of y and 0 of z.

	Max	Alloc	Need	Avail
	x y z	x y z	x y z	x y z
P_0	8 4 3	0 0 1	8 4 2	3 2 2
P_1	6 2 0	3 2 0	3 0 0	$\langle 1\ 2\ 2 \rangle$
P_2	3 3 3	2 1 1	1 2 2	$\langle 6\ 4\ 2 \rangle$
				$\langle 8\ 5\ 3 \rangle$
				$\langle 8\ 5\ 4 \rangle$

T_0: P_0 : $\rightarrow \langle 0,0,2 \rangle$

$\langle P_1, \ldots \rangle$

System is unsafe & Req 1 is not granted.

T_1: P_1 : $\langle 2,0,0 \rangle$

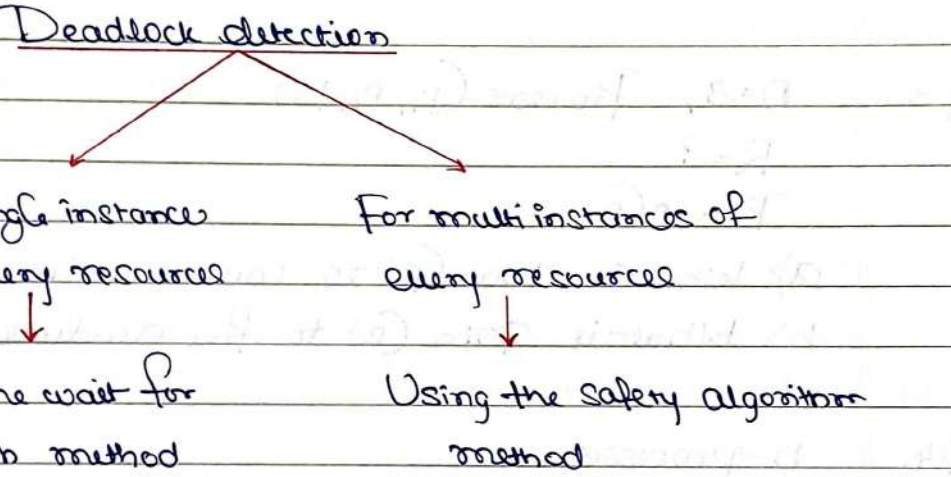
Request 2 : $\langle P_1, P_2, P_0 \rangle$

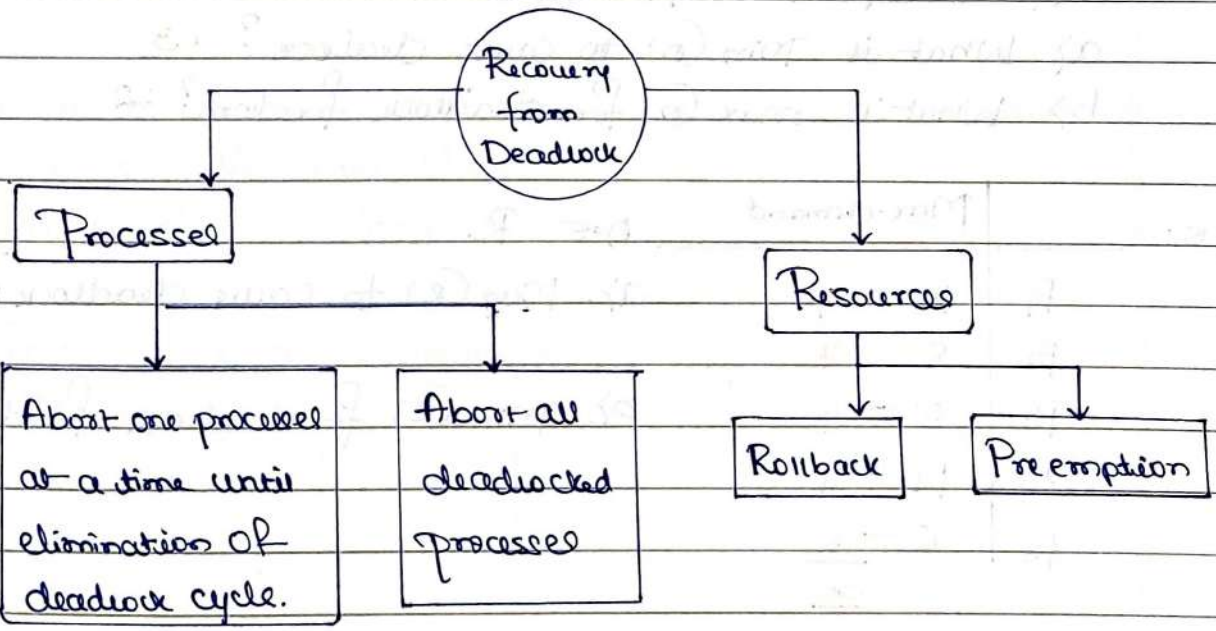
$\therefore$ System is safe ~~but~~ and request is granted

Only Request 2 is granted and Request 1 is not granted

(iv) Deadlock detection and recovery:

When to activity/activate (apply detection algo.)?
→ Under utilization of cpu
— majority of processes are blocked.





1. Prevention } Class 1
2. Avoidance }
3. Detection and Recovery } Class 2
4. Ignorance. } Class 3.

Conceptual problems.

Q2. Consider a system having 'n' processes and a single 'R' having '6' copies. Each process need 2 copies of R to complete.
a) What is min (n) to cause deadlock? : → 6
b) What is max (n) to deadlock freedom? : → 5

Q3 n=3, Processes (P_1, P_2, P_3)
R = ?
$P_i = 2(R)$
a) What is Max(R) to cause deadlock? → 3
b) What is Min (R) to free deadlock? → 4

Q4. n-processes
R = 6 copies. P_i requires : 3(R)
a) What is Min (n) to cause deadlock? : 3
b) What is max (n) for deadlock freedom? : 2

Q5.

	Max-demand	n=5, R.
P_1	10 — 9	a). Max(R) to cause deadlock : 36
P_2	8 — 7	
P_3	5 — 4	b). Min (R) for deadlock freedom : 37
P_4	12 — 11	
P_5	6 — 5	
	36	

Q6. Which of them is not a valid deadlock prevention scheme?
Ans Never request a resource after releasing any resource

Q7. Which is not true of deadlock prevention and deadlock avoidance?
Ans In deadlock prevention the request for resources is always granted if the resulting state is safe.

Q8. An OS implements a policy that requires a process to release all resources before making a request for another resource. Select true statement.

Ans. Starvation can occur but deadlock cannot occur.

Q9. A computer has 6 tape drives, with n processes competing for them. Each process may need two drives. What is the max value of 'n' for deadlock freedom?

Ans 5

Q10 Consider a system having m resources of same type. These resources are shared by 3 processes A, B, C which have a peak demand 3, 4, 6 resp. What value of 'm' deadlock will not occur?

Ans Value should be greater than 10.

Q11. A system shares 9 tape drives. The current allocation and max req. of tape drives for:
Which of them best describes the current state of system?

Process	Current Allocation	Max. Req.
P_1	3	7
P_2	1	6
P_3	3	5

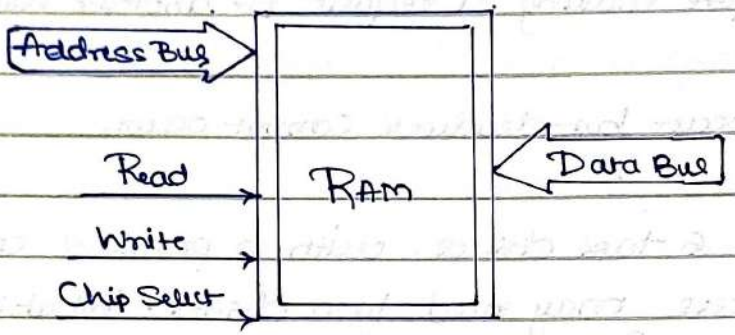
Ans: Safe, Not deadlocked

Q12 Which of following statements is/are true w.r.t deadlocks?

Ans. a. Circular wait is a necessary condition for formation of deadlock.

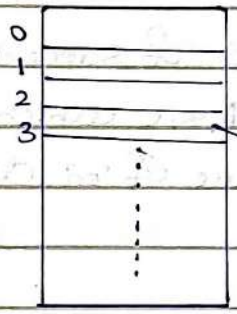
d. In the resource allocation graph of a system, if every edge is an assignment edge, then system is not in deadlock state.

Memory Management

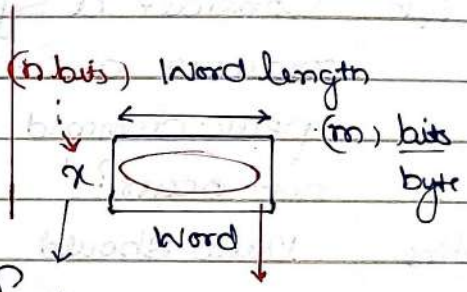


Block diagram - Primary Memory

Linear dimensional view of Memory
(Abstract view)



Words / Locations / Cells



Total no. of words
= 'N' Memory Capacity

fixed length address

You can store instructions + data

Memory: Memory is specified as number of words × width of words
$(N \times m)$

n: Size of address in bit
m: word length / width of word
N: total number of words

I. **Relation b/w N & n:**

→ Binary bits: (0, 1)
→ Using 3 binary bit 8 combinations are possible
→ 'n' bits :→ 2^n combinations possible

eg:

$N = 8$ words

$n = 3$ bits

$2^3 = 8W$

$N = 2^n$ words
$n = \log_2 N$ bits

Conversions:

$2^{10} = 1024 = 1K$ words
$2^{20} = 2^{0}$ 1m. words
$2^{30} = 1G$ words
$2^{40} = 1T$ words.
$2^{50} = 1P$ words

eg. * $N = 256$

$n = \log_2 256 = 8$ bits

* $n = 6$ bits

$N = 2^6 = 64$ words.

* $n = \log_2 z$ bits

$N = z^{(\log_2 z)}$

$= z^{(\log_2 z)}$ $\therefore$ $N = z$ Bytes

I. $N \propto n \propto m$.

1. $n = 13$ bits
$m = 8$ bits
$N = 8KW$
$= 8KB$.

2. $n = 18$ bits
$m = 16$ bits
$N = 256$ KW Word view
$= 512$ KB Byte view

3. $n = 33$ bits
$m = 64$ bits
$N = 8GW$
$= 64GB$

4. $N = 64$ KB
$m = 4$ B
$n = \frac{64KB}{4B} = 16K$ Words.

5. $n = 23$ bits
$m = 32$ bits
$N = 2^{23} = 2^3 \times 2^{20} = 8m$ words
$N = 8m \times 4 = 32m$ Bytes.

6. $N = 256$ MB
$m = 128$ B
$N = \frac{256MB}{128B} = 2m$ words.

$n = 21$ bits (W)
$n = 28$ bits (B)

7. $N = 8G$ bits (Twist)
$m = 64$ bits
$N = 128m$ words
$N = 1G$ Bytes.

Loading vs Linking

Loading: Loading refers to the activity of loading the program or bringing the program from disk to memory. The component of the operating system that does this activity is called "Loader".

The two approaches of loading are: 1. Static loading
2. Dynamic loading

1. **Static loading**: The whole program is loaded in memory before starting the execution.
   * Advantage: faster execution (time efficient)
   * Drawback: possible wastage of space (ineffective utilization of memory)

2. **Dynamic loading**: The process of loading the program on demand at runtime is called Dynamic loading.
   * Advantage: no wastage of memory
   * Drawback: slow execution (time inefficient)

Linking: Linking is the process of resolving external references in the program

```
#include <stdio.h>
   :
main()
{  :
   fun();   BSA
   :
BSA
  scanf();
}
```

```
fun()
{ ...
  ...
  ... }
```

Note: Compilation always starts with 1 from first line of code

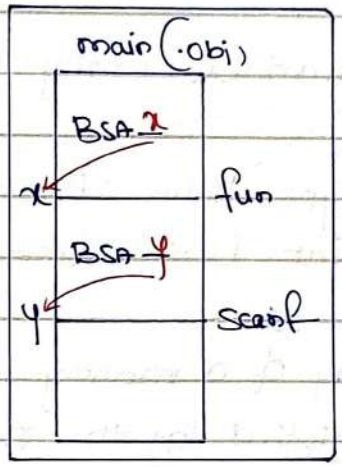
* Execution of program always start with main()

BSA: Branch and save address

The two ways of linking are: 1. Static linking
2. Dynamic linking

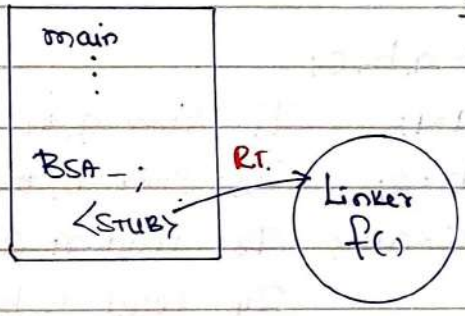
Static Linking: Linking of program before execution. Space inefficiency is the drawback of static linking.

```
main()          | fun()
{ ....          | { :
  ....          |
BSA_ fun();     |   Scanf();
     :          |   :
   }            |   }
```



Dynamic Linking: Linking at runtime is called dynamic linking. It is space efficient but time inefficient.

```
main()          | f()
{ ....          | { :
  ....          |
BSA_ f();       |   Scanf();
<STUB>  :       |     BSA_
     :          |     :
   }            |   }
```



STUB: a small piece of code that gives the address to BSA.

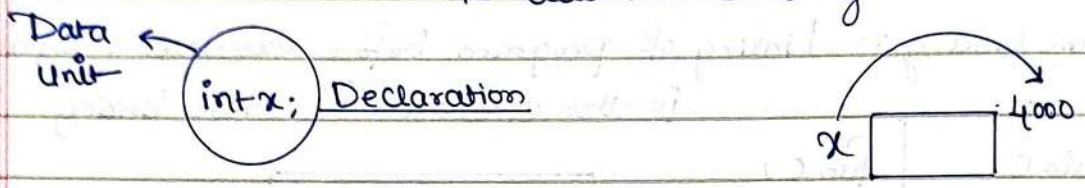
.dll: (Dynamic link library). The libraries linked at runtime using principle of dynamic linking is called DLL

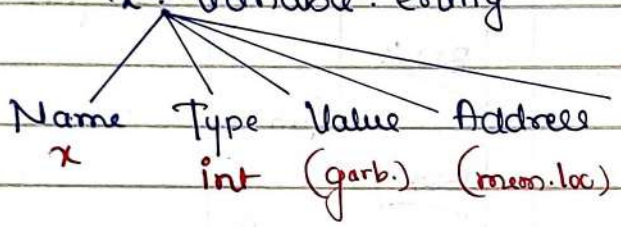
Advantages:
* Space efficiency
* Reusability
* Modification (flexibility) is easy.

Drawbacks:
* Time inefficiency
* Less security (Insecure)

Note: Static linking is more secure

Address Binding: Association of program instructions and data units to memory locations (address) is address binding.

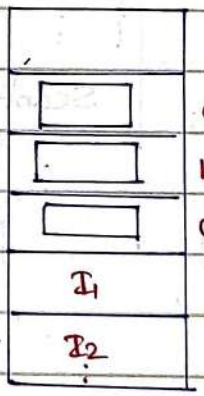




Q1. The capacity of a memory unit is defined by the number of words multiplied by the number of bits/word. How many seperate address and data lines are needed for a memory of 4K x 16?

Ans $n = \log_2 4K = 12$ bits, $m = 16$ bits $= 2B$ = Dataline/bus

```
int a, b, c;
a=1;      I1: Store a, #1
b=2;      I2: Store b, #2
c=a+b;    I3: load r1, a
          I4: load r2, b
          I5: Add r1, r2
          I6: Store c, r1
```



Binding time: Time at which the binding takes place is called as binding time.

```
int (x);
x=1;
```

Type : at compile time (static binding)
Address : at load time (St. b)
Value : run time (address binding)
Size : Compile time (St. b)

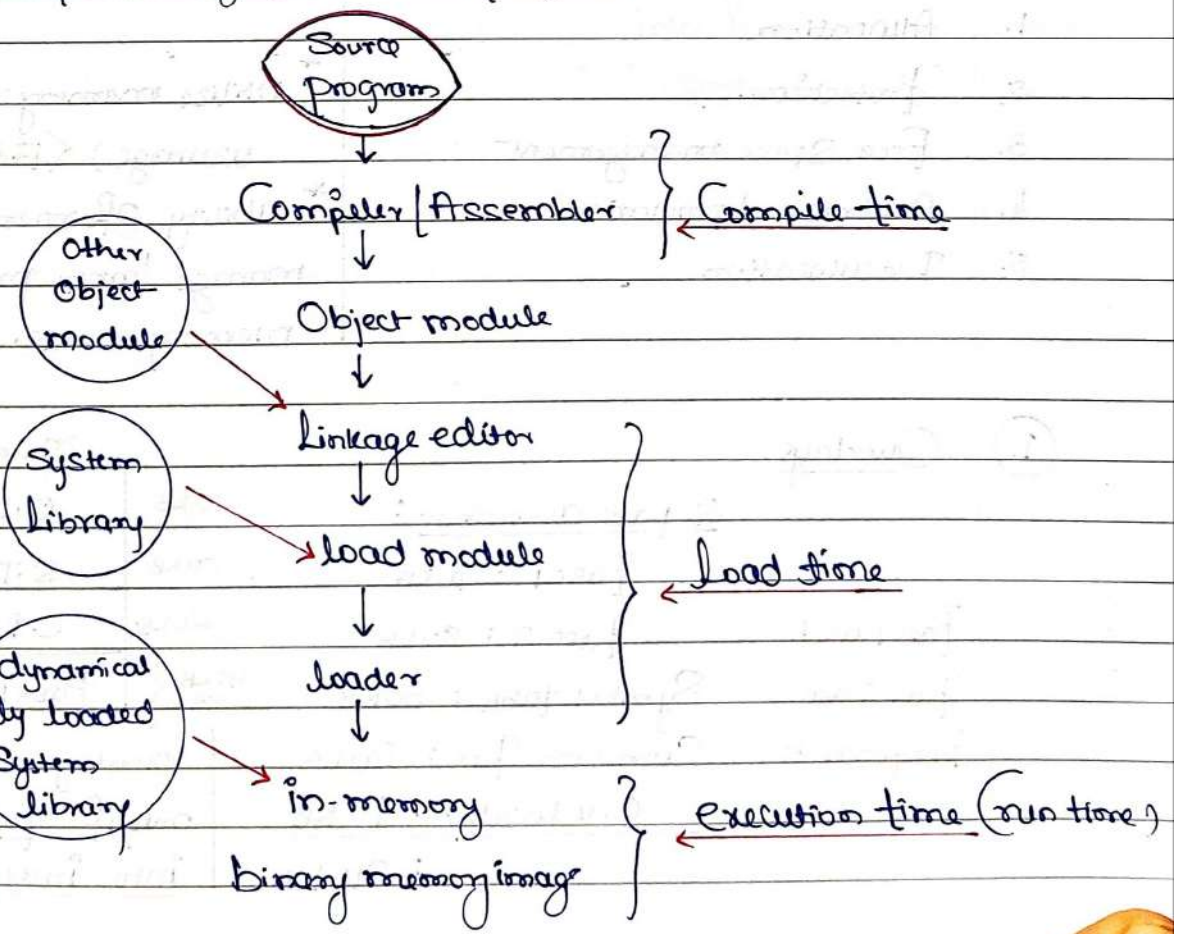
Types of binding: 1. Static binding (cannot change)
2. Dynamic binding (can change)

Address Binding

(Static) Compile time	(Static) Load time	(dynamic) Runtime address binding
	Loader.	Loader.
Compiler: Address	→ Compiler will not generate/associate address	→ Relocation happens in runtime (relocatable)
I1: 4000		→ Flexibility is advantage
I2: 4004		
I3: 4008	→ Loader will assoc.	

I1: 0, I2: 4, I3: 8 } offset — 6000 Base (0.5) address

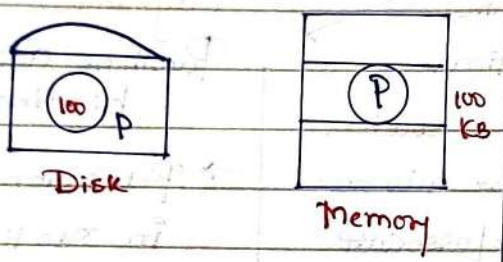
Multistep processing of a user program



Date 23/2/23

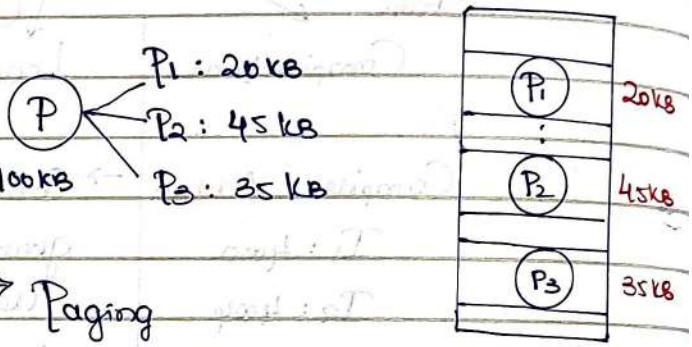
Memory Management Technique.

Contigous <centralized>



Technique →
- Overlaye
- partitions → fixed, variable
- buddy System.

Non-Contigous <decentralized>



- Paging
- Segmentation
- Paging and Segm.
- Demand paging.

Function of Memory Manager:

1. Allocation
2. Protection.
3. Free Space management
4. Address translation
5. Deallocation

Goals of memory manager:

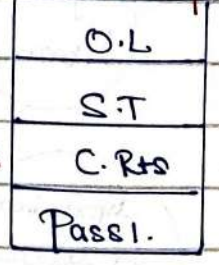
* Utilize memory efficiently (min wastage) <Fragmentation>
* ability of memory manager to manage larger program in small memory areas.

(1.) Overlays.

2 pass assembler :
- Pass 1 : 70 KB
- Pass 2 : 80 KB
- Symbol table : 30 KB
- Common Rts : 20 KB
- O.V loader : 10 KB

: 210 KB.

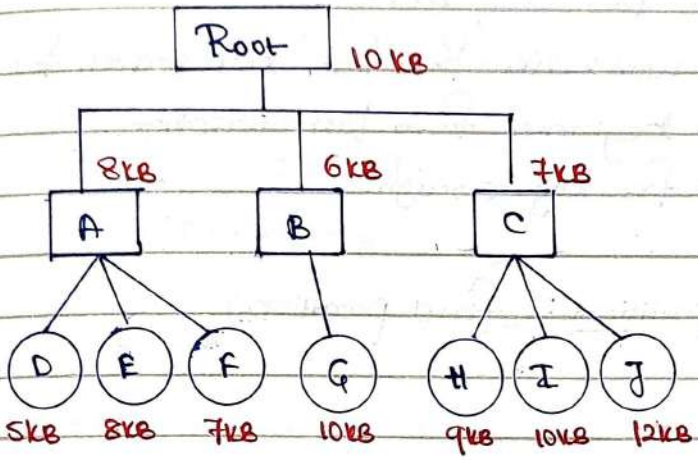
Pass 1 and Pass 2 are independent.



{overlay = replace}

Overlaye concept is possible only if "program divisible into independent modules."

Consider the following program expressed as an overlay.



Prog-Size : 92KB (Condicy)

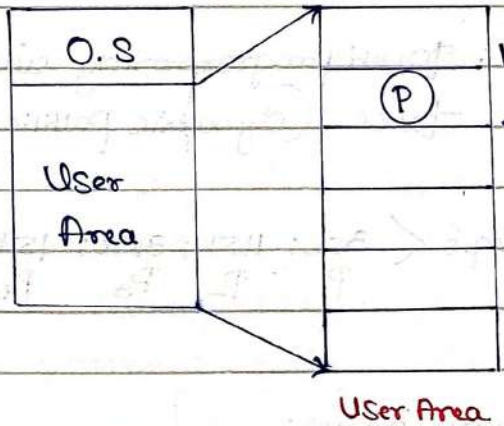
* What is minimum memory required to Successfully execute this program using Overlay?

Min. Memory = max { Path-length Required from root to leaf }.

= 29KB

(2.) Partitions:

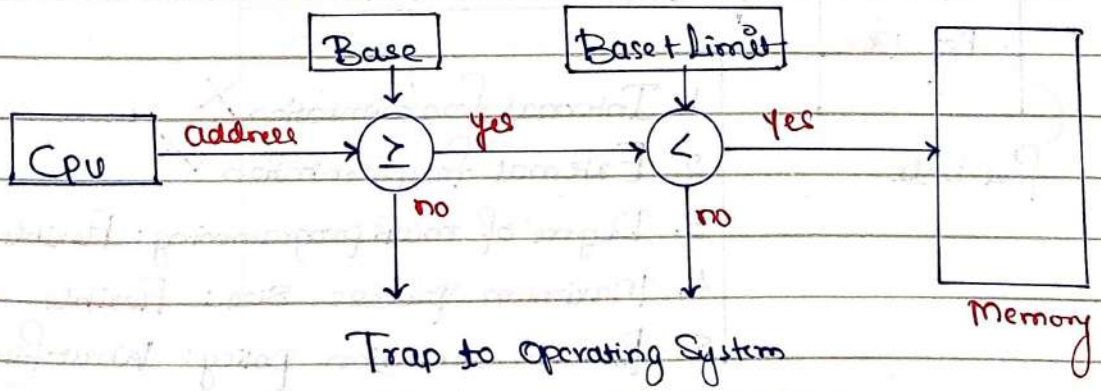
(i) Fixed partitions (MFT): Multiprogramming with fixed tasks.



→ User area will be divided into fixed no. of partitions

→ 1 partition = 1 partition

→ partitions maybe of different sizes.



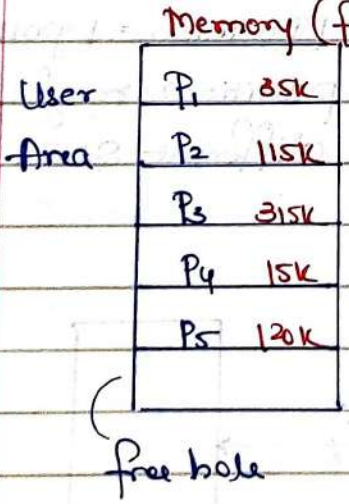
Partition allocation policies:

1. First fit: first free big enough (Internal fragmentation)
2. Best fit: smallest free big enough
3. Next fit: next fit works like first fit, but search for free partition beginning from last allocation.
4. Worst fit: largest free big enough.

Performance of fixed partition: (fixed partition)

1. Internal fragmentation ✓
2. External fragmentation ✗
3. Degree of multiprogramming: Limited
4. Maximum process size: Limited
5. Partition allocation policy: Best fit (less internal fragment.)

(ii) Variable Partitions (MVT): multiprogramming with variable task (dynamic partitioning).

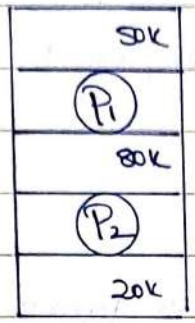


Process Req.'s < 35K; 115K; 315K; 15K; 120K; ....
P1 P2 P3 P4 P5

Performance issues:

1. Internal fragmentation ✗
2. External fragmentation ✓
3. Degree of multiprogramming: Flexible
4. Maximum process size: Flexible
5. Partition allocation policy: Worst fit is better.

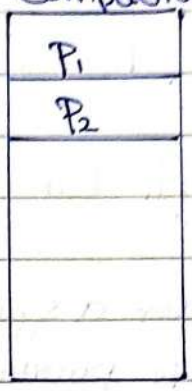
External fragmentation:



Memory map.

Free space: 150K
Process: P3 - 85K
(External frag of 150K)

(i) Compaction (time consuming operation)



relocate

* Compaction is not possible if there is no runtime address binding.

(ii) Non Contigous allocation:

* One of the reason of having non contigous allocation is to avoid the problem of external fragmentation
* N.C.G allocation divides the process into smaller parts and distributed to free holes.

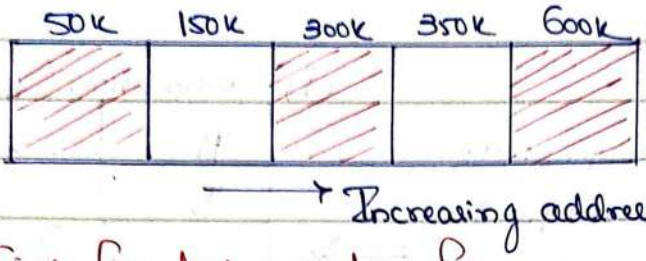
Note: Adjacent free holes are automatically merged to one free hole.

Q1 Consider a memory system having 6 partitions of sizes 200K, 400K, 600K, 500K, 300K, 250K. There are 4 processes of size: 351K, 210K, 468K, 491K. Using Best fit allocation policy, what are partitions that remain Unallocated?

P4	200
P1	400
	600
P3	500
	300
P2	250

600K, 300K remain unallocated

Q2 Consider the following memory map in which blank regions are not in use and hatched regions are in use. Using variable partitions with no compaction.



The sequence of requests for blocks of size 300K, 25K, 125K, 50K, can be satisfied if we use:

Ans First fit but not best fit.

Q3. Consider a system with memory of size 1000KBytes. It uses variable partitions with no Compaction. Presently there are 2 partitions of sizes 200K and 260K respectively.

(i) What is the allocation request of the process which could be always denied?

Ans: 541K

(ii) What is the smallest allocation request which could be possibly denied?

Ans: 181K

Q4. Consider a system having memory of size 2^{46} Bytes, uses fixed partitioning. It is divided into fixed size partitions each of size 2^{24} Bytes. The OS maintains a process table with one entry per process. Each entry has 2 fields: first, a pointer pointing to partitioning in which process is loaded and second, field is Process Id. The size of PID is 4 Bytes.

Calculate:

(a) The size of pointer to the nearest Byte :- 3B

No. of partitions (N) = $\frac{2^{46}}{2^{24}} = 2^{22}$, Partition address = 22 bits = 3Byte

(b) Size of process table in Byte is if system has 500 processes.

Ptr Size = 3B

Process id = 4B

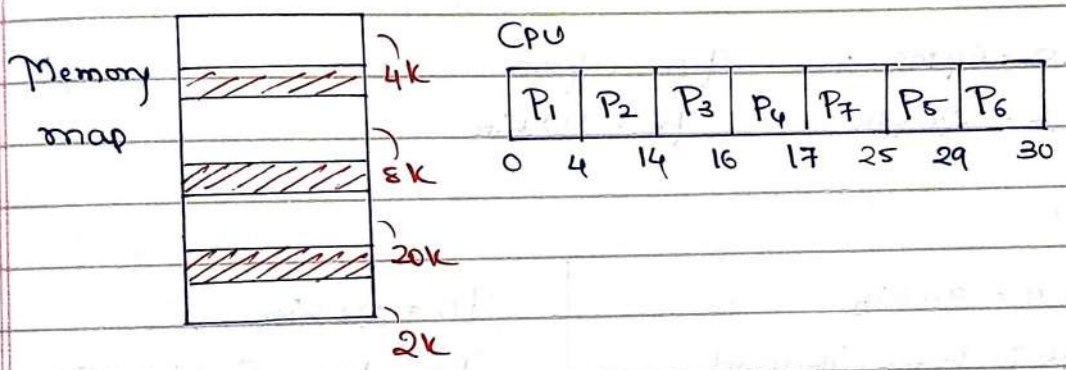
7B × 500 → 3500 B

Qs. Consider a system with Variable partition with no compaction

HW

Free holes: 4k, 8k, 20k, 2k.
Program size: 2k, 14k, 3k, 6k, 10k, 20k, 2k.
Time for execution: 4, 10, 2, 1, 4, 1, 8.

Using best fit allocation policy and FCFS Cpu scheduling, find the time of loading and time of completion of each program. The fault time in seconds are.



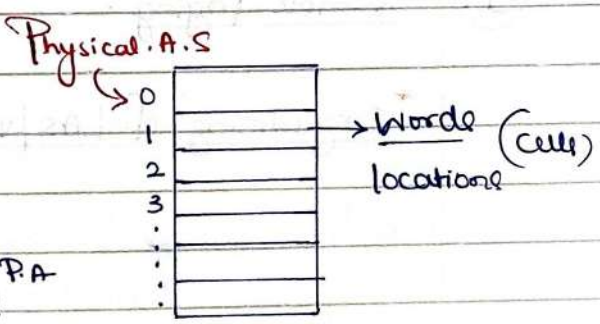
Q6. Consider the allocation of memory to a new process. Assume that none of the existing holes in the memory will exactly fit the Processes memory requirement. Hence, a new hole of Smaller Size will be created if allocation is made in any of the existing holes. Which of the following is true?

Ans- The hole created by best fit is never larger than hole created by first fit.

Non Contigous Allocation: (to prevent external fragmentation)

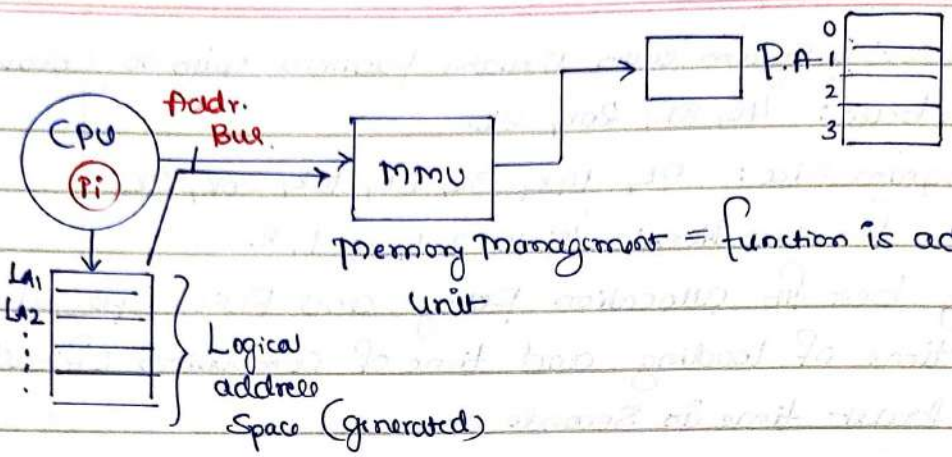
Address Space: A set of space/words associated with addresses

Virtual (A.S)



$P_{AS} = 2^{P.A}$

$P.A = \log_2 P_{AS}$.



Memory Management = function is address translation.
Unit

Eg: ① L.A.S = 64 MB ; PAS = 4 MB
LA = 26 bits PA = 22 bits

② LA = 33 bits
LAS is byte, if word Size is 64 bit = 8B.
$LAS(B) = 2^{33} \times 8B = 8G \times 8B$
$= 64\,GB$

PA = 23 bits
PAS, W.S = 64 bits = 8B
$8M \times 8B = 64MB$

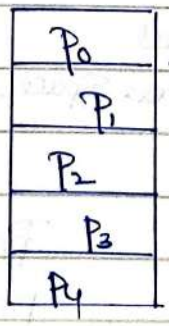
* Logical address: address generated by CPU to access instruction/data Unit of program in execution.

* Physical Address (Real/Absolute): address needed to access needed to access the program instructions/data unit in physical memory ie address in MAR.

1. Simple Paging:

* Organising of LAS/VAS:

LAS: 5KB. → LAS is divided into equal size units known as pages.
→ Page size is generally in the power of 2.



Formulas

→ No. of page (N) = $\frac{LAS}{PS}$

→ Page No (P) = $\log_2 N$

$N = 2^P$

→ Page offset / displacement (d) = $\log_2 PS$ (bits)

→ P.S = 2^d B

L.A / V.A format

LA = 13 bits

P	d
3	10

Q1.

(i)

L.A.S = 32 MB

P.S = 4KB = 2^{12}

N = 8K [2^{13}]

P = 13 bits

d = 12 bits

L.A :

P	d
13	12

LAS = 2^{25} = 32 MB

(ii)

L.A = 25 bits

d = 13 bits

*What are the no. of pages in the system?

LA : P d (25 bits)

12 13

N = 2^{12} = 4K

(iii)

System has 2K pages and LA of 32 bits. What is P.S?

N = 2K

P = 11 bits

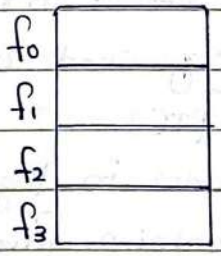
32 bits

P	d
11	21

d = 21, P.S = $2^d = 2^{21}$

= 2 MB

*Organization of P.A.S (4KB):



→ P.A.S is divided into equal size units known as frames [page frames].

→ Frame Size = page size

→ Each frame holds one page

→ Any page can be stored in any frame (NCG)

Formulas

* No. of frames (m) = $\frac{P.A.S}{FS}$

* frame no. (f) bits = $\log_2 m$.

$m = 2^f$

* frame offset = page offset = d

* P.A format:

f	d
2	10

12 bits

Q1. L.A = 31 bits, P.A = 20 bits, P.S = 4 KB

$N = LAS / PS = \frac{2^{31}}{2^{12}} = 2^{19} = 512K$

$m = \frac{2^{20}}{2^{12}} = 2^8 = 256$, $p = 19b$. $f = 8$ bits, $d = 12$ bits

LA: | p | d | 19+12 = 31

PA: | f (8) | d (12) | = 20 bits.

Q2 System has 4K page and 1K frames, Calculate the size of P.A.S, if LA = 32 bits.

p = 12

N = 4K ; m = 1K = f = 10

LA = 32 bit

LA: | p (12) | d (20) | ← 32 bit →

PA: | f (10) | d (20) | ← 30 bit →

$PAS = 2^{30} = 1GB$

* Organization of M.M.U [page table / address table]

→ Each process has its own P.T

→ P.T's are stored in memory

→ P.T is organised as set of entities, known as "Page table entries" (PTE).

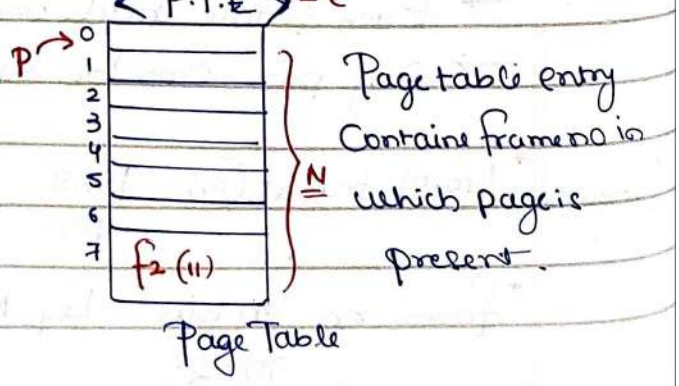
→ No. of entries in Page table = No. of pages.

→ P.T.E Contains frame number in which page is present.

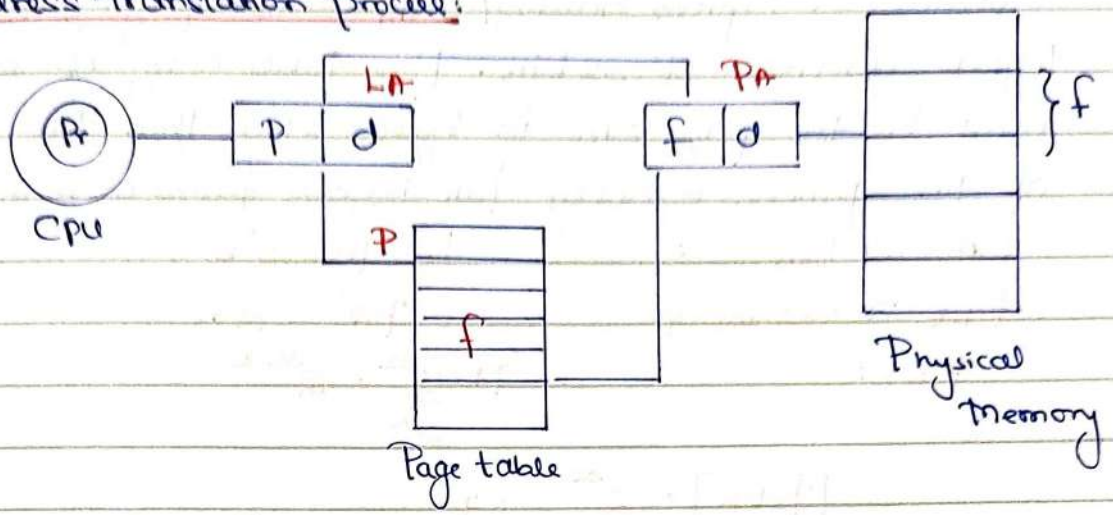
→ P.T.E is denoted as 'e' (Bytes)

→ P.T Size = N * e (Bytes)

e ≥ 1B



Address translation process:



Q1. A computer System using paging technique implements an 8KB page with page table of size 24MB. The page table entry is 24 bits. What is the length of virtual address in the system?

Ans

P.S = 8KB	P.T.S = 24MB
d = 13 bits	e = 24 bit = 3B.

P.T.S = N x e.

$N = \frac{P.T.S}{e} = \frac{24}{3}\,MB = 8M.$

$\therefore$ 23 bits

Q2 Consider a computer system with 40 bit virtual addressing and page size of 16KB. If the computer system has a one level page table per process and each page table entry requires 48 bits, then the size of per process page table is megabytes.

Soln

V.A = 'l' bits

N = '2'

m = 'H'

$P = \log_2 2$

$d = (l - \log_2 2)$ bits

PA: $(\log_2^H + \log_2^2)$

$(l + \log_2 H - \log_2 2)$

$P.T.S = 2$

$= 2^{1 + \log(H/2)}$

$= 2^1 \times 2^{\log(\frac{H}{2})}$

$= 2^1 \times (\frac{H}{2})$

Q3. Consider a System having/using Simple paging technique with logical address of 32 bits. Page table entry of 32 bits. What must be the page size in bytes, such that the page table of the process exactly fits in one frame of memory (PAS)?

Sol^n L.A = 32 bits
P.T.E = e = 32 bits = 4B
P.S = ?

Let P.S = 2^x B
P.T.S = N * e
$N = \frac{2^{32}}{2^x} = 2^{32-x}$

$[P.T.S] = 2^{32-x} \times 4B$
$= 2^{34-x}$

$2^{34-x} = 2^x$
$34 = 2x \quad \therefore x = 17$

Q4. Consider a System using Paging technique with VA = 32 bits and P.S = 4KB, what must be appropriate P.T.S in bytes, given P.A.S = 64MB.

Ans PAS = 64 MB
VA = 32 bits
PS = 4 KB

P.T.S = N * e
$N = \frac{2^{26}}{2^{12}} = 2^{14}$, $N = \frac{2^{32}}{2^{12}} = 2^{20} = 1M$

P.T.S = 1M x 2B
= 2MB

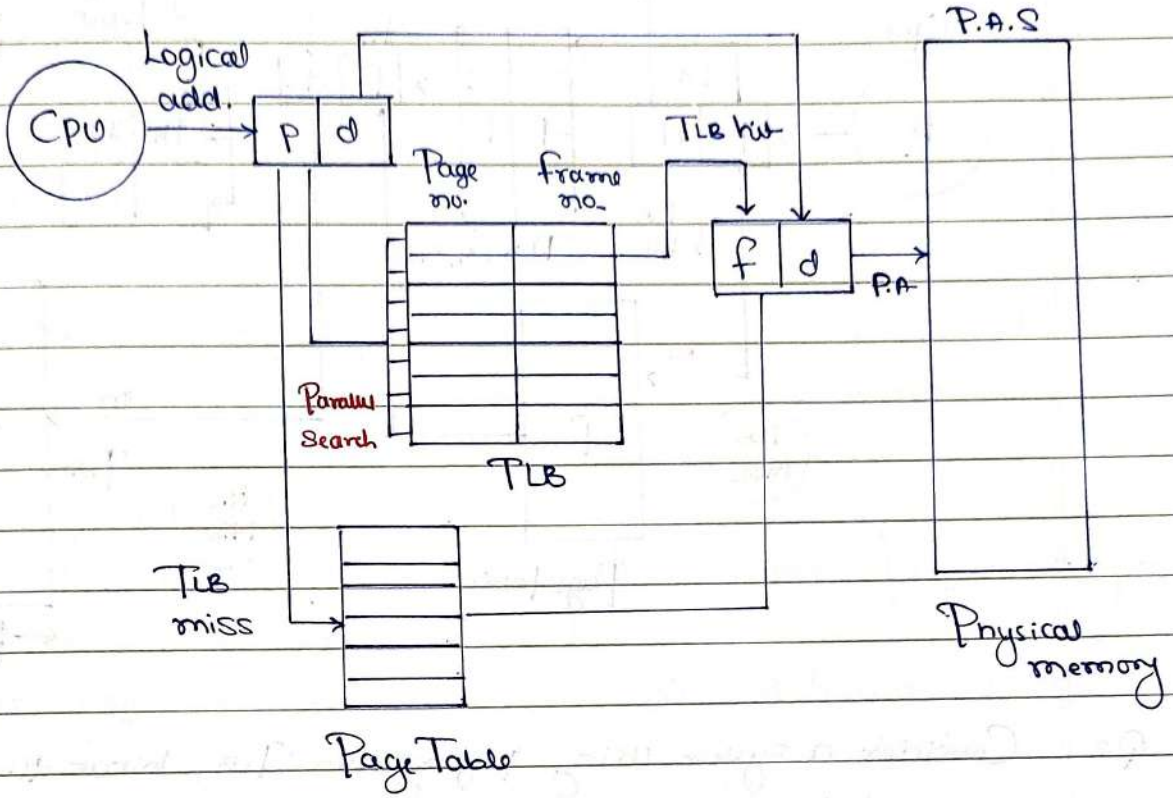
Performance of paging:

(1.) Timing issue:
* Main memory access time (MMAT) = 'm'
* Effective memory access time (EMAT) = 2m.
* Objective: Reduce EMAT from 2m to m.

Cache memory:
* faster memory access time
* costly
* Smaller in size
* Supports parallel search.

↓
TLB
(Translation Lookaside Buffer)

Paging Hardware with T.L.B.



Formulas

* EMAT = 'm'
* TLBAT = C
* EMAT = 2m.
* TLB hit = 'x' ratio
* TLB miss = '1-x' ratio
* EMAT = $x(C+m) + (1-x)(C+2m)$

Q1. m = 100 ns

C = 20 ns

(a) EMAT with paging: $2m = 2 \times 100 = 200$ ns

(b) If x = 90·1. what is ELB TLB: $x = 0.9$, $(1-x) = 0.1$

$= x(C+m) + (1-x)(C+2m)$

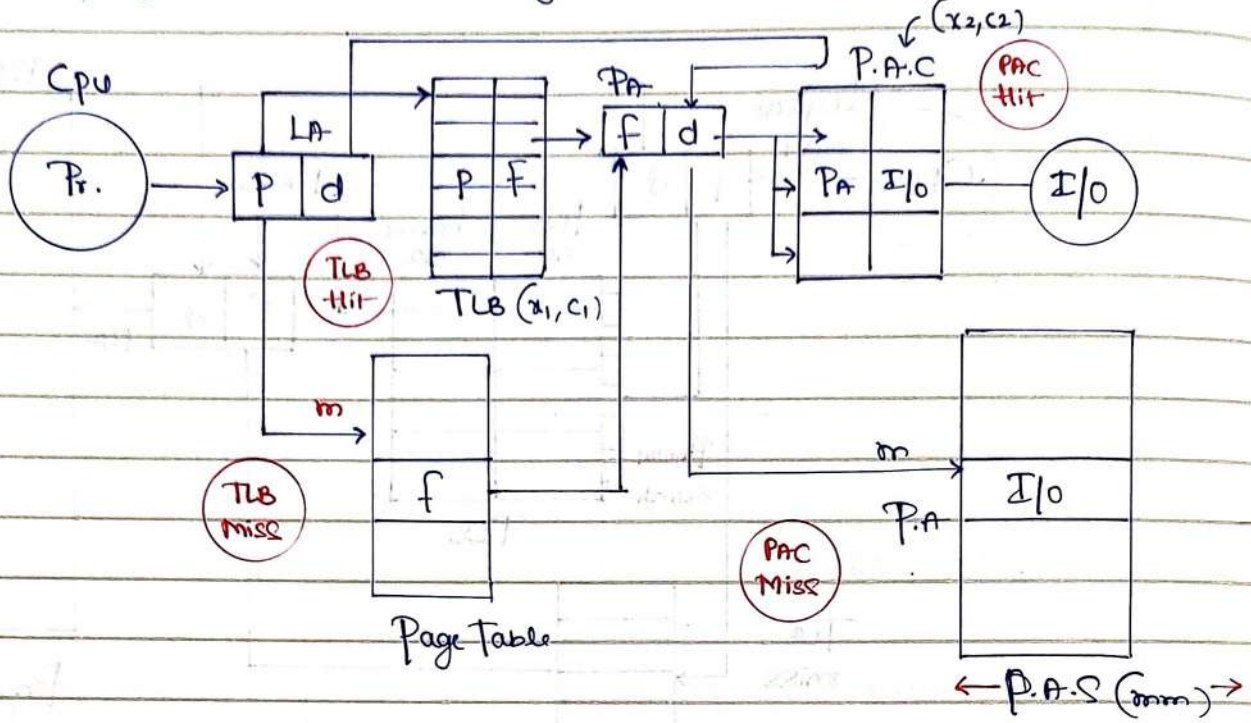
$= 0.9(120) + 0.1(220) = 108 + 22 = 130$ ns [200 - 120]

(c) If x = 10·1 what is EMAT TLB?

$= 0.1(120) + 0.9(220)$

$= 210$ ns [200 - 210 ns]

Paging with TLB and Physical Address Cache (PAC)



Q2 Consider a system using paging with TLB. What hit ratio is required to reduce the EMAT from 'D' to '2' ns using TLB. Assume that TLB access time is 'K' ns.

$2m = D$

$m = D/2$

$EMAT = x(C+m) + (1-x)(C+2m)$

$'2' = x\left[K + \frac{D}{2}\right] + (1-x)[K+m]$

(2) Space Consumption in Paging

$PTS_{(By)} = N * e\ (By)$

$PTS \propto N$

eg: LA = 32 bits, PS = 4KB

Appropriate PT size ?

$N = 2^{32}/2^{12} = 2^{20}$

Objective: Reduce Page table size of processes.

if e = 4B

PTS = 1m × 4B = 4MB

If system has 100 processes, amount of memory required = 100 × 4 = 400MB.

Reduce P.T Size | Associate Smaller PTs with processes.

* P.T.S = N * e

P.T.S $\propto$ N — (i)

* P.T.S $\propto N \propto \frac{1}{PS}$

* $N = \frac{LAS}{PS}$

$N \propto \frac{1}{PS}$ — (ii)

$$P.T.S \propto \frac{1}{PS}$$

(I.) Reduce P.T.S by increasing Page Size.

Optimal page Size?

Let V.A.S = 'S' Bytes

Let P.T.E = 'e' Bytes

Let Page Size = 'P' Byte.

(I). P.T.S = $\left[\left(\frac{S}{P}\right) * e\right]$ Bytes

(II). If = $\frac{P}{2}$ Bytes.

Total overhead = $\left[\left(\frac{S}{P}\right)e + \frac{P}{2}\right]$ — (1)

diff eqn (1) w.r.t. P — 0

$$\left[\frac{-1}{P^2} Se + \frac{1}{2}\right] = 0$$

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

$$P = \sqrt{2Se}$$

Q1. A machine has a 32 bit address space and an 8KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of CPU time is devoted to loading the page tables?

$$N \text{ pages} = \frac{2^{32}}{2^{13}} = 2^{19}$$

Time to load P.T = $2^{19} \times 100$ ns.

% Fraction of CPU time = $\frac{2^{19} \times 100 ns}{100 ms}$

$= \frac{2^{19} \times 100 \times 10^{-9}}{100 \times 10^{-3} s}$ $\Rightarrow$ 50%

Q2. Consider a system using paging technique with an address space of 65,536 Bytes. The page size in this system is 4096 Bytes. The program consist of Text, Data and Stack sections as per the space:

Text: 32,768 B
Data: 16,386 B
Stack: 15,870 B

A page of the program contains portion of only one section i.e either Text or Data or Stack

(a) Does program fit in given address space? No

(b) What is the maximum page size in Bytes such that the program fits in given address space.

VAS = 65,536 = 64KB

$N = \frac{64KB}{4KB} = \frac{2^{16}}{2^{12}} = 2^4 = 16$

Text = 32768/4096 = 8
Data = 16386/4096 = 5
Stack = 15,870/4096 = 4
Total = 17 So you cannot.

(II) Hashed paging / Paging with Hashing

* Element (x) → h(f(x)) → 'i' → index
* We will design a Page table using hashing

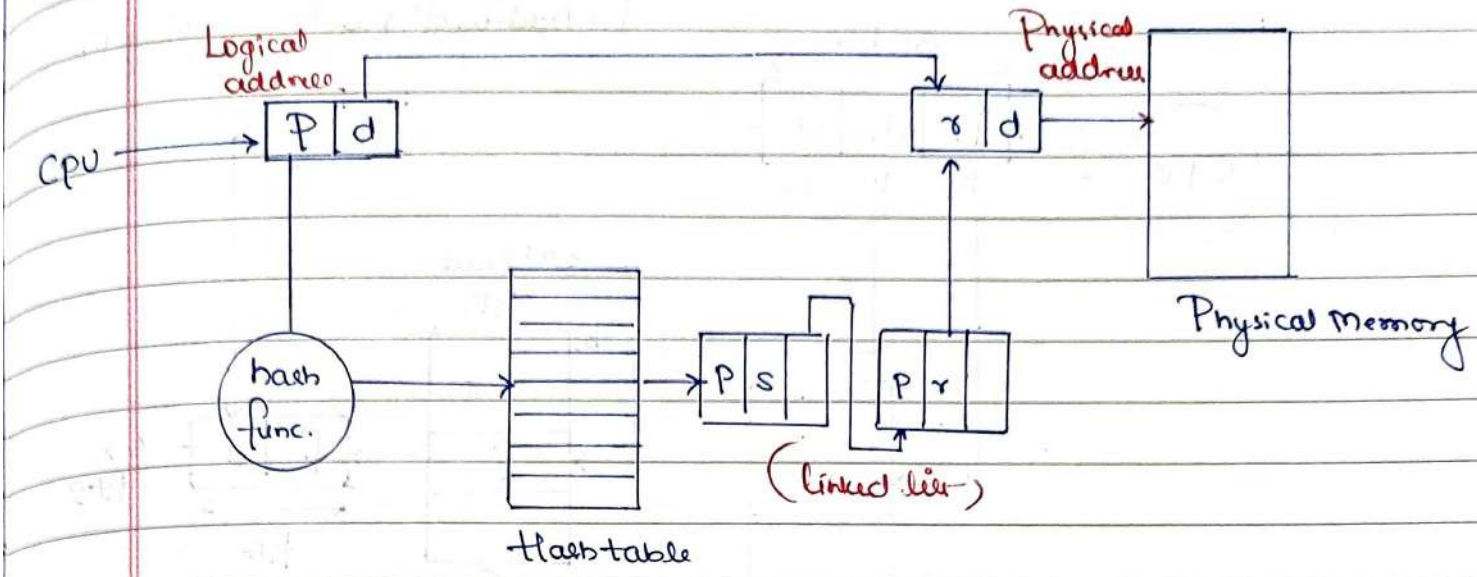
[x % 10] ← 0 : 9

Collision: Let I_1 and I_2 be two distinct elements

$f(I_1) = f(I_2) = i$

* To resolve collision → Chaining is used.
* Space optimized, but time inefficient

Hashed page table



(III) Multi Level Paging / Hierarchical Paging (Recursive paging)

Objective: To reduce page table size overhead by associating smaller page table's with process.

eg:- VA = 32 bit
P.S = 4 KB
e = 4B
PTS = 1MB × 4KB
= 4MB

When do we say P.T is small?
:- If the page table fits in one frame of memory

Paging as a Concept involves 3-Steps:

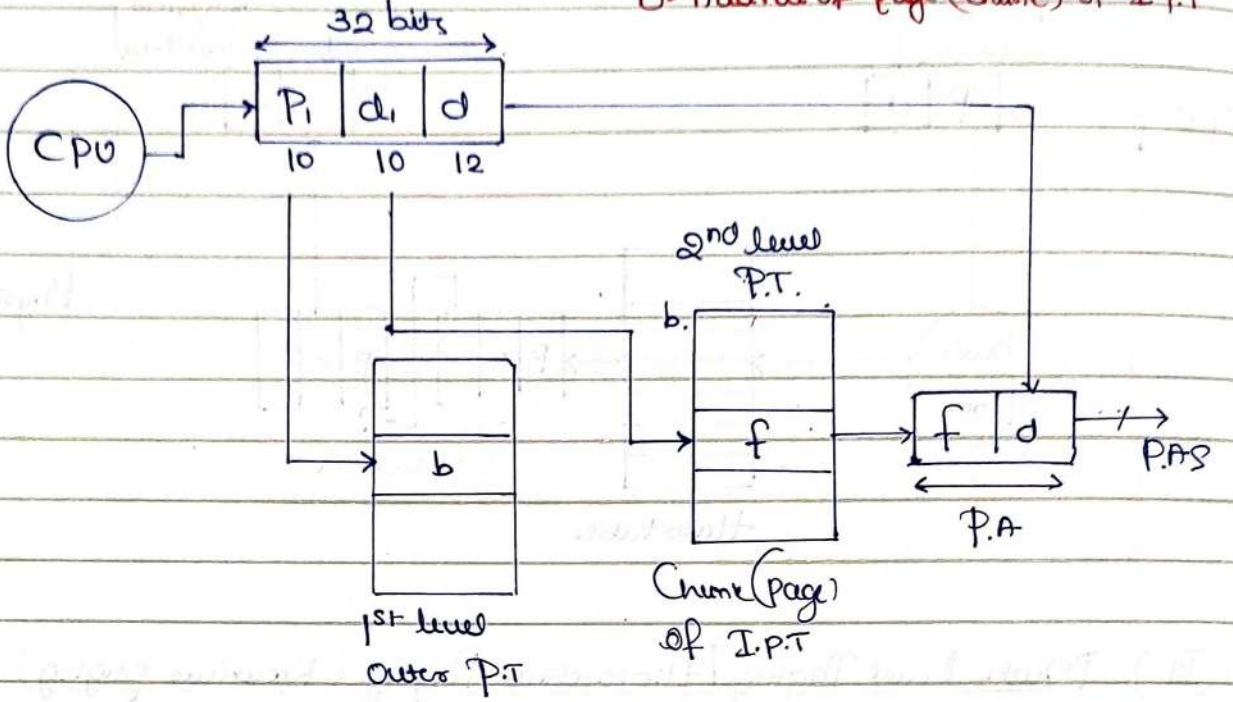
1. Divide the address space into pages (chunks).
2. Storing the page in physical address space
3. Access the pages (chunks) of address space in physical address space through page table

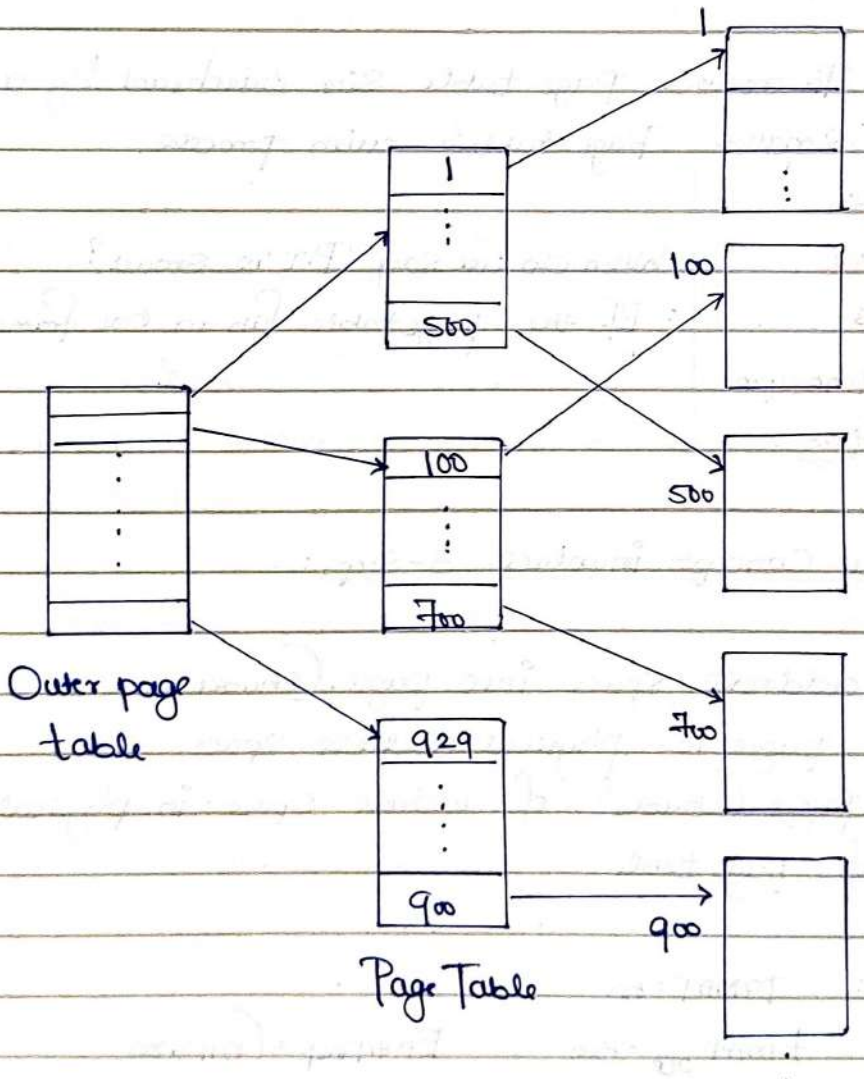
Performance: $EMAT = m$

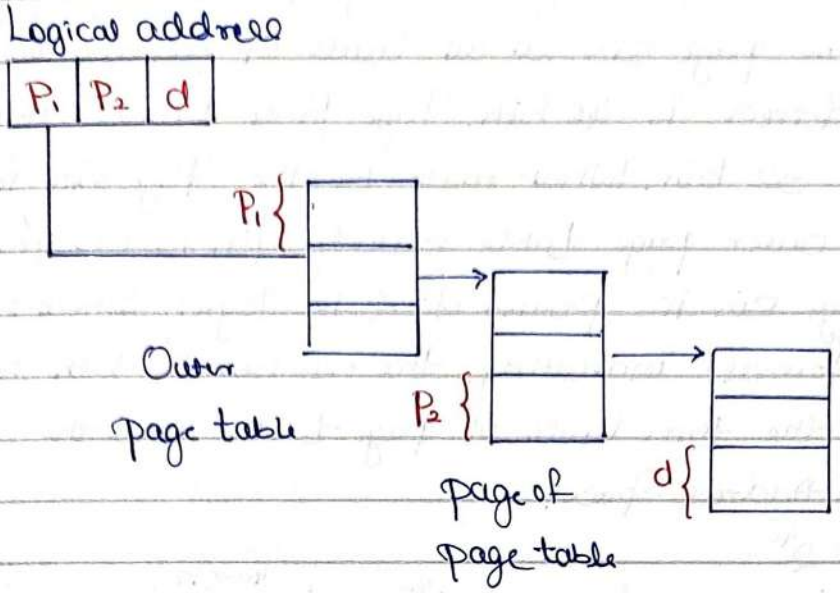
$EMAT_{2LP} = 3m$ $\quad EMAT_{nLP} = (n+1).m$

eg:: LA : 32 bits, P.S = 4 KB.

b = Address of page (chunk) of I.P.T







Q.1 In the context of operating systems, which of the following are correct w.r.t. paging?
* Paging helps solve the issue of external fragmentation
* Paging incurs memory overheads.

Q2. Consider a system using 2 level paging Architecture. The top level 9 bits of the Virtual Address are used to index into the outer Page Table. The next 7 bits of the addresses are used to index into next level page table. If the size of virtual address is 28 bits. Then

(i) How large are the pages and how many are there in virtual address space?

P → No of page in VAS , $N = 2^{16} = 2^6 \times 2^{10} = 64KB$

P.S of VAS $= 2^{12} = 4KB$

(ii) If P.T.E at both levels is 32 bits in size then what is the space overhead needed to translate Virtual Address to Physical Address of an instruction or Data Unit?

P1 P2 d 2KB + 512B

9 7 12. 2.5KB

Q3. Consider a Computer System using three level paging Architecture with uniform page size at all levels of paging. The size of virtual address is 46 bits. Page Table Entries at all levels of paging is 32 bits. What must be the Page size in Bytes such that the outer page table exactly fits in one frame of memory. Assume page size is power of 2 in Bytes. Show the virtual address format indicating the number of bits required to process all the three levels of page tables and the page offset of virtual Address Space.

$VAS = 2^{46}$

Let $P.S = 2^x B \rightarrow \left(\frac{2^{46}}{2^x}\right) = [2^{46-x+2}]_B \rightarrow \left[\frac{2^{46-x+2}}{2^x}\right]_B^2$

$\Rightarrow [2^{46-3x+2+2+2}]_B$

$x = 13$

$\therefore 2^{13} = \underline{8KB}$

Derivation:

Let $VAS = 2^S$ Byte $VA = S$ bits

$P.S = 2^x$ Byte, $P.T.E = 2^c$ Byte

Levels of paging = 'l'

$\rightarrow$ Size of outer level $P.T = [2^{S-l.x+l.c}]$ Byte.

$\rightarrow \frac{2^S}{2^x} \cdot 2^c = [2^{S-x+c}]$

Q4 Consider a computer System with 57 bit virtual Addressing, using multi level tree Structured page tables with L levels for virtual to Physical address translation. Page size is 4KB and Page Table Entry is of 8 Bytes at all level.

The value of L is 5

VA = 57 bit P.S = 4KB OPT = 1 Page / Frame

No. of level = 'L' P.T.E = 8B OPT = 4KB.

$$OPT = \left[2^{57-l.12+L_3} \right] \text{ Bytes.}$$

$$= 2^{12} B$$

$$57 - 9l = 12$$

$$l = 5$$

HW

Qs. Consider a three level page table to translate a 39-bit virtual address to a physical address as shown below.

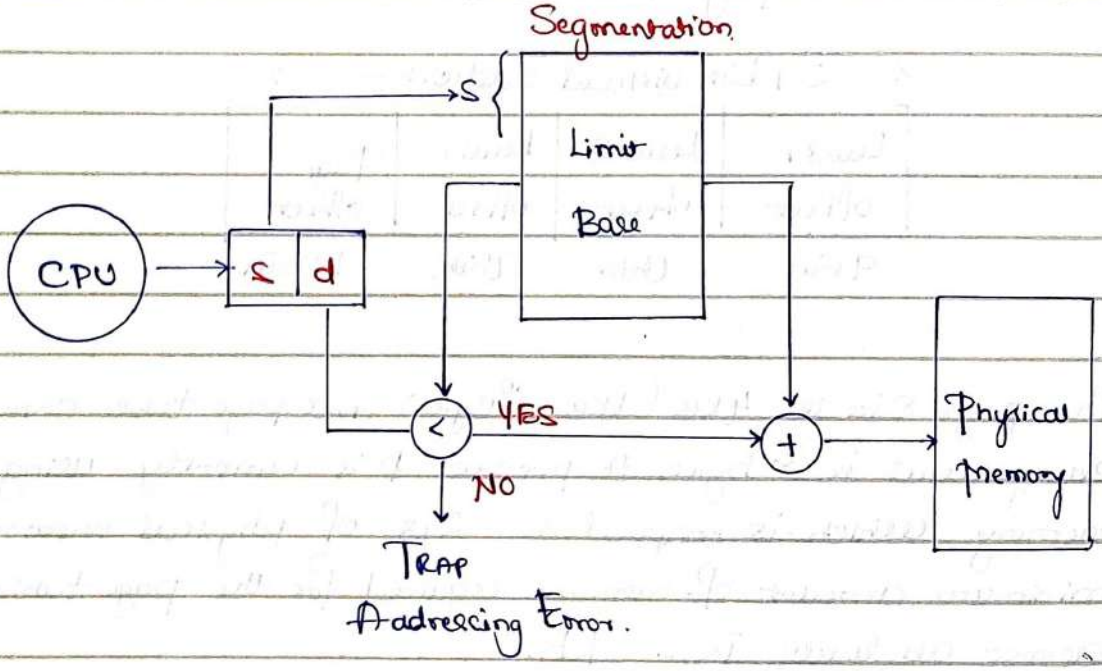
← 39 bit virtual address →

Level 1 offset	Level 2 offset	Level 1 offset	Page offset
9 bits	9 bits	9 bits.	12 bits

The page size is 4KB (1KB = 2^{10} Bytes) and page table entry size at every level is 8 bytes. A process P is currently using 2GB virtual memory which is mapped to 2GB of physical memory, the minimum amount of memory required for the page table of P across all levels is ___ KB.

2. Segmentation.

* Paging does not pressure user's view of memory allocation to programs
* As per user's view of memory allocation, Program is divided into logical units, known as segments (function, block, procedure, D.S, class).
* These segments are assumed to be stored in their activity entirely at non contigous locations.



HW

Q1. Consider the following segment table:

Segment	Base	Length
0	1219	600
1	3300	14
2	90	100
3	2327	580
4	1952	96

What are the physical address for the following logical addresses?

Performance of Segmentation

1. Time : MMAT = 'm' TLBAT = C

EMAT = '2m' TLB Hit ratio = α

TLB Miss ratio = $1-\alpha$

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

2. Space :

When Segment table become large, apply paging on Segment table

Compare Paging and Segmentation w.r.t fragmentation

	Int. Frag.	External frag.
Paging	✓ <last page>	X
Segment	X	✓ in P.A.S.

Segmentation

↓

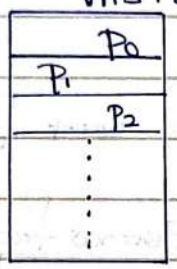
External fragmentation

1. Compaction
   De-fragmentation.
2. Paging
   ↓
   (Segmented-paging)

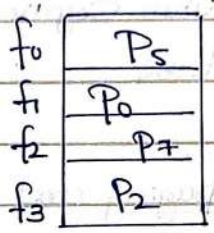
Virtual Memory (VM)

Virtual memory gives an illusion to the programmer that a huge / large amount of memory is available for executing programs, that are larger than the available / given physical memory

P.A.S = 8KB
P.S = 1KB.



P.A.S = 4KB.

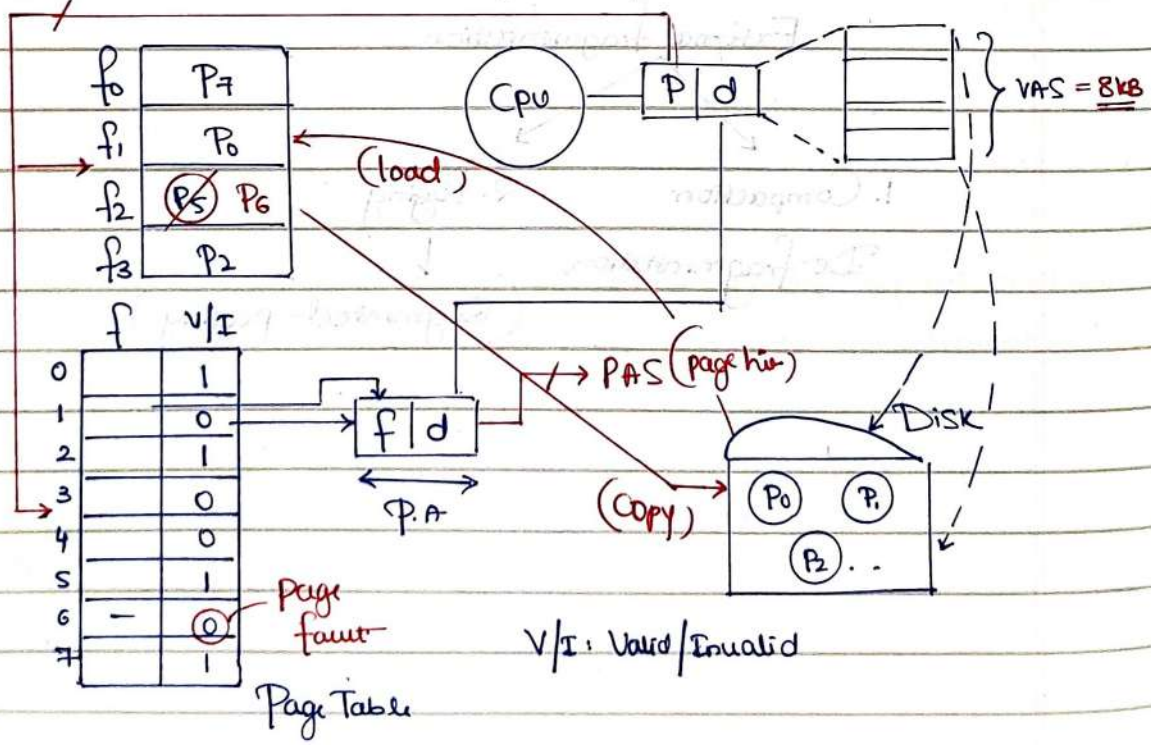


V.M
↓
Demand Paging

Demand paging: * Loading the page from disk to memory on demand at runtime.

* Virtual memory is implemented through "Demand Paging".

* Program (8KB / 8 Pages) : Stored initially on disk, required pages are loaded in P.A.S.

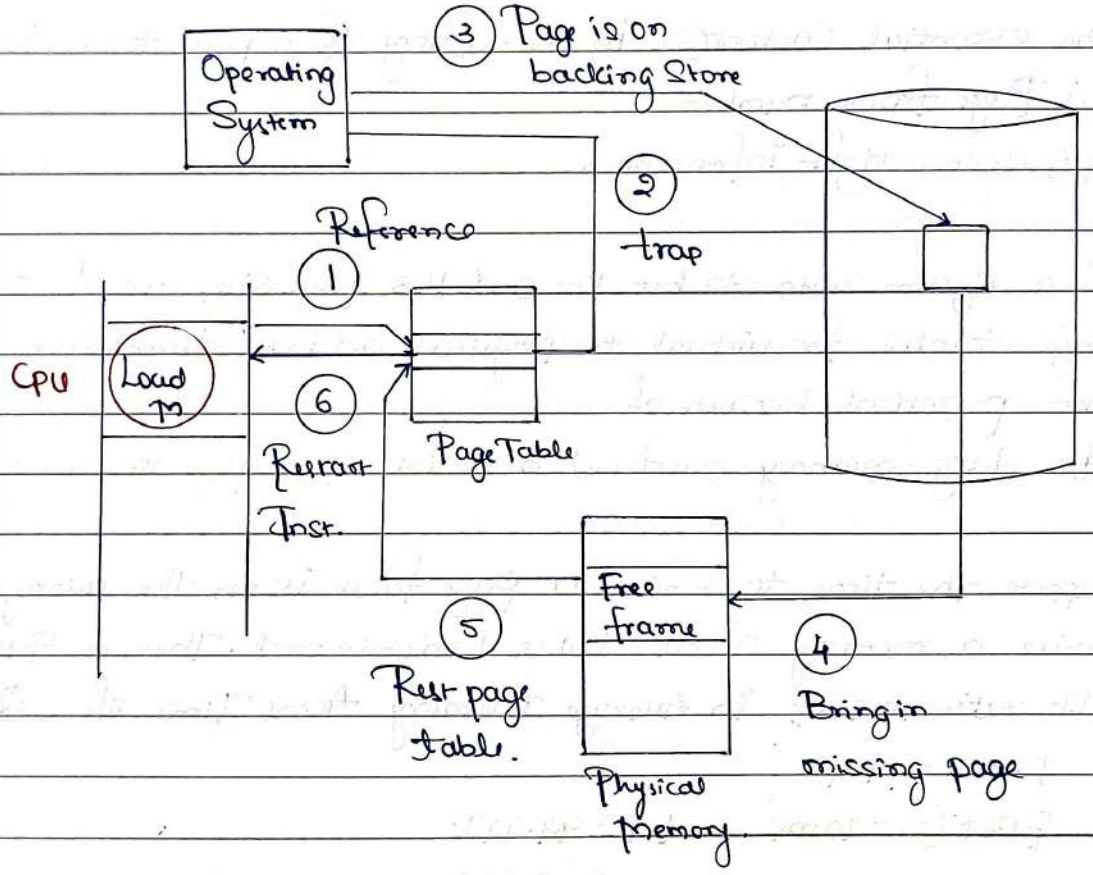


Demand paging

- Pure DP
  < Execution of the process will start with empty frame >
- Pre Fetched D.P

Note:
* Size of virtual memory is limited by size of disk.
* P.A.S ≤ VAS ≤ Disk AS.

Steps taken to service page fault:



* Amount of time taken by O.S (VMM) to service a page fault is known as "Page fault Service time." (PFST)

Q1. In a virtual memory system the address space specified by the address lines of the CPU must be Larger than the physical memory size and Smaller Secondary Storage size.

Performance of Virtual Memory:

1. Time issue (Temporal)

MMAT = 'm' [ns | µs]

PFST = 'S' [S >> m]

Page fault rate = 'P'

Page hit rate = '1-P'

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

D.P

Q2 The total size of address space in virtual memory system is limited by:

Ans. Size of Secondary Storage

Q3. The essential content(s) in each entry of a page table is/are

Ans (i) Page frame number

(ii) access right information

Q4 In a system with 32 bit VA and 1KB page size, use of one level page tables for virtual to physical address translation is not practical because of

Ans the large memory overhead in maintaining page tables

Q5. Suppose the time to service a page fault is on the average 10 ms, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in Average Memory Access Time of 2µs

Ans MMAT = 1µs

PFST (S) = 10ms 1-P = 99.99% = (0.9999)

P = 0.01%

= 0.0001. EMAT = (1-P) m + P*S

$= 0.9999 \times 1 \times 10^{-6}s + 0.0001 \times 10 \times 10^{-3}s$

$= 0.9999 \times 1 \times 10^{-6} + 1 \times 10^{-6}$

= 1.9999 ≈ 2µs ~ 2µs

Q6. If an instruction takes 'i' microseconds and a Page fault takes an additional 'j' microseconds, the Effective Instruction Time if on the average a page fault occurs every 'k' instructions is _____

Ans Instruction time without Page fault : 'i' μs
Instruction time with Page fault : 'i+j' μs
$P = 1/k$
$(1-P) = \left(1 - \frac{1}{k}\right)$ Effective Inst. Time : $\frac{1}{k}(i+j) + \left(1-\frac{1}{k}\right)i$

$= \frac{i}{k} + \frac{j}{k} + i - \frac{i}{k}$

$= \underline{i + j/k}$

HW

Q7. Assume that we have a Demand-paged memory. It takes 8 milli-seconds to service a page fault if an empty frame is available or if the rejected page is not modified, and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nano sec. Assume that the page to be replaced is modified 70 percent of the time. What is the acceptable page fault rate for an effective access time of no more than 2000 nanoseconds?

Ans S ——→ 8(ms) : Ef + Clean page [70%] (30%)
↘ 20 (ms) : Dirty page [30%] (70%)

m = 100 ns
P = ?
EMAT = 2000 ns

$2000\,ns = (1-P) * 100\,ns + P(0.7 * 20\,ms + 0.3 * 8\,ms)$

$P =$

Q8. Consider a process executing on an Operating System that uses demand paging. The average time for a memory access in the system is m units if the corresponding memory page is available in memory, and D units if the memory access causes a page fault. It has been experimentally measured that the average time taken for a memory access in the process is X units. Which one of the following is correct expression for the page fault rate experienced by the process?

Ans $m = M$ $\quad$ $S = D$

$X = (1-p)m + p * D$

$\Rightarrow p = \frac{(X-m)}{(D-m)}$

Q9. Consider a paging system that uses 1-level page table residing in main memory and a TLB for address translation. Each main memory access ~~time~~ takes 100 ns and TLB lookup takes 20ns. Each page transfer to/from the disk takes 5000 ns. Assume that the TLB hit ratio is 95%, page fault rate is 10%. Assume that for the 20% of the total page faults, a dirty page has to be written back to disk before the required page is in read in from disk. TLB update time is negligible. The average memory access time in ns (round off to 1 decimal place) is 154.5

Ans m — 100ns $\quad$ C — 20ns

Disk R/W — 5000ns

TLB hit rate = 95%

Page fault rate = 10%

80% — Clean page

20% — Dirty page

EMAT = 0.95 [20ns + 100ns (TLB hit)] + 0.05 [20ns + 0.9 (100ns + 100ns) + 0.1 (P.F, TLB miss) (100ns + 0.2 × (5000 + 5000ns) + 0.8 (5000 ns)]

(page hit; D.P. = Dirty page; Clean page)

EMAT = 154.54 ≈ 155

Q10. The minimum number of page frames that must be allocated to a running process in a virtual memory environment is determined by

Ans Instruction Structure Architecture.

Page Replacement : one important concept of Virtual Memory implement.

(a) Reference String (Page-Ref string) : Set of successively unique pages (Page Nos) referred in given list of V.A's.

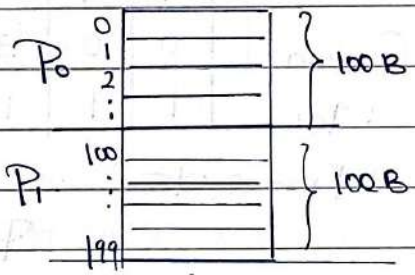
eg:: (Pi) < 702, 704, 123, 654, 483, 012, 122, 124, 180, 785, 934, ... >
< 7, 1, 6, 4, 0, 1, 7, 9, 6, 7, . >

Let P.S = 100 B

$P = \frac{VA}{PS}$ $d = VA \% PS$

Length = 10
Unique pages = 6

P7 (702)
(P) 7 (d) 2



<0...99> : P0
<100...199> : P1
<200...299> : P2

(b) Frame allocation Policies

* n = no. of processes (P1...Pn)
* Si = Demand (frames) of process Pi
* D = Total Demand $= \sum_{i=1}^{n} Si$
* m = Available frames
* ai = Allocated frames

n = 5 <P1...P5> ; m = 40

Pi	Demand Si	Eq. alloc. ai = m/n	Pro.p alloc ai = (Si/D) m	ai = Si/2
P1	10	8	5	
P2	5	8	3	
P3	35	8	17	
P4	18	8	9	
P5	12	8	6	

D = 80

* Min. no of frames to be allocated to a process:
→ < Process cannot execute without this minimum number of frames >
→ Process should be able to execute minimum one instruction.

Page Replacement Techniques:

1. Ref string : < 7; 0; 1; 2; 0; 3; 0; 4; 2; 3; 0; 3; 2; 1; 2; 0; 1; 7; 0; 1 >
length = 20, n = 6

(1) FIFO

	7	2	2	2	4	4	4	0	0	0	7	7	7
Criteria	0	0	3	3	3	2	2	2	1	1	1	0	0
: TOL	1	1	1	0	0	0	3	3	3	2	2	2	1

3 Frames : 15 — frames —

$$P = \frac{18^{3}}{20_{4}} = \underline{75\%}$$

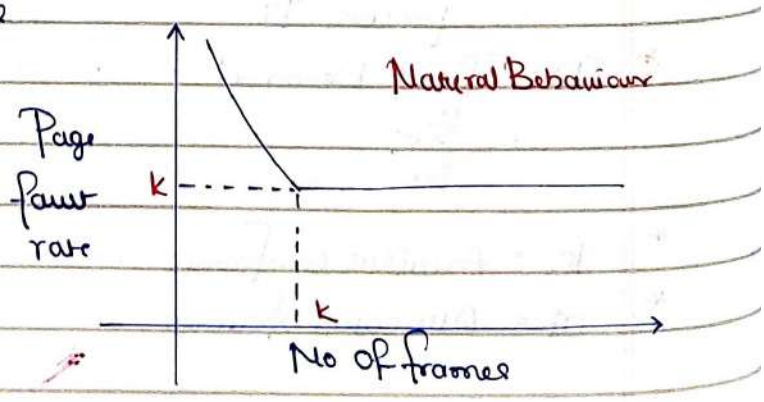
eg:: 2 Ref string: < 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 >
FIFO : pure Demand paging

hits 3 frames:

3 Frames: 9
4 Frame: 10
(HW

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

Frames



Belady's Anomaly :

As the number of frames allocated to the process increases, page fault also sometimes increases.

Only FIFO and FIFO based suffers from Belady Anomaly.

(2) Optimal Replacement : replace that page which will not be used for the longest duration of time in future references.

eg:: Ref string <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

3 Frames : 9
4 Frames : ?
(Hw)

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

Frames.

* We use this algorithm as a benchmark to check performance of other replacement strategies. Optimal replacement is not implemented practically.

(3) Least Recently used (LRU) : replace that page which has not been used for the longest duration of time in the past.

Criteria : Time of Refrence.

eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 7, 0, 1>

3 Frames : 12.
4 Frames : ?
(Hw)

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

(4) Most Recently used (MRU) : Smyttrical opposite of least recently used, replace that page which has been used for longest duration of time.

HW eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

3 Frames:

4 Frames:

(5) Counting Algorithms:

a) L.F.U [least frequently used]

b) M.F.U [most frequently used]

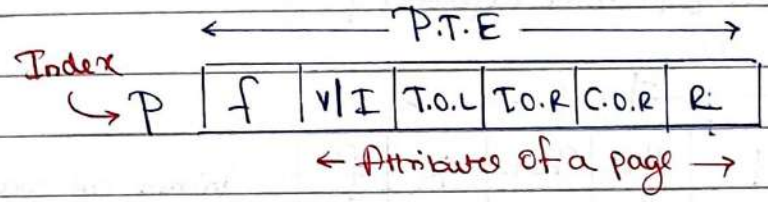
*Criteria: Count of reference.

eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

(HW: Solve with 3 frames (both)!)

Note: * Among all the algorithms, Optimal is having least page fault rate.
* LRU ~ Optimal
* Many OS implement either LRU / LRU approximately
* The best method to implement LRU is by "stack method."

LRU Approximations: <These algorithms are not really LRU, but they approximate to the behaviour of LRU>.
They are based on a concept called <Reference bit (R)> : Each page in the page table will be associated with a reference bit 'R'.



a.> Reference bit (R):

Criteria: 'R'
- 0 : Page is not referred so far during present Epoch.
- 1 : Page has been referred atleast once during present Epoch.

eg::

	P	f	V/I	T.O.L	R
	(0)	a	1	2	1
	1	e	1	3	0
	2	d	1	0	1
P.F	(3)	–	0	–	–
	4	c	1	4	10
	(5)	b	1	1	0

Page Table

Reference bit fails when all R values are '1'.

b>. Additional Reference bits: Each page is associated with more than one reference bits.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8 ↓ Current Epoch
P_i	1	0	0	1	0	1	1	1
P_j	0	0	1	1	1	0	1	1
P_k	0	0	1	1	0	1	1	1

(P_i) X : Cause Page fault

When current Epoch gets over

↓

[Shift left operation is]

c). Second Chance [clock algorithm]

Criteria : [T.O.L + Reference]

eg::

← P.T.E →

P	V/I	T.O.L	R	f
0	1	2	1	e
1	1	3	0	a
2	1	0	X 0	b
3	0	–	–	–
4	1	1	0 ✓	c
5	1	4	1	d

Page Table

When all page 'R' value is '1' then FIFO page gets selected

* Second ~ FIFO Chance

↓

Belady's Anamoly

d). Enhanced Second Chance / NRU (Not recently used).

Criteria: Reference + modified

RM

I. 00 → Not referenced and not modified

II. 01 → Not referenced but modified

III. 10 → Referenced but clean

IV. 11 → Referenced and modified

eg::

← P.T.E. Structure →

P	f	V/I	T.o.L	R	m
0	c	1	4	1	1
1	a	1	3	0	0
2	-	0	-	-	-
3	b	1	0	1	1
4	d	1	2	1	0
5	e	1	1	0	1

(i) FIFO: P_3
(ii) R: P_1
(iii) Second Chance: P_5
(iv) Enhanced Second Chance: P_1

Q1. Consider a system with V.A.S = P.A.S = 2^{16} Bytes. Page size is 512 Bytes. The size of the page table entry is 32 bits. If the page table entry contains besides information 1 V/I bit, 1 Reference, 1 modified bit, 3 bits for page protection. How many bits can be assigned for storing other attributes of the page. Also compute page table size in Bytes?

Ans:

← P.T.S = 32 bits →

P	f	V/I	R	m	Pr.	Other.
	7	1	1	1	3	2

No. of frames = $\frac{2^{16}}{2^9} = 2^7$

P.T.S = N x e
$= 2^7 \times 2^2 B = 2^9 B \rightarrow 512B$

Q2. Consider a virtual memory system with FIFO page replacement policy, for an arbitary page access pattern. Increasing the number of page frames in main memory will.

Ans Sometimes increases the no. of page faults (Belady's Anamoly)

Q3. A memory page containing a heavily used variable that was initialized very early and is in constant use is removed when

Ans. FIFO page replacement algorithm is used.

Q4. Recall that Belady's anamoly is that the page fault rate may increase as the number of allocated frames increases. Now consider the following Statements:

S1: Random page replacement algorithm (where a page chosen at random is replaced) suffers from Belady's anamoly

S2: LRU page replacement algorithm suffers from Belady's anamoly.

Ans: S1 is true and S2 is false.

Q5 Consider a process having reference string of length 'l' in which n - unique pages occur. 'z' frames are allocated to the process. Calculate the lower bound and upper bound of the no. of page faults.

a) maximum = 'l' [z=1]
(Upper bound)

b) minimum (l) = 'n' [z≥n] - if we use pure demand paging
(Lower bound)

= 0 - if we use pre-fetched demand paging

Thrashing: Excessive / high paging activity (high page fault rate)
↓
(The act of causing a page-fault leading to loading and saving the pages on disk.)

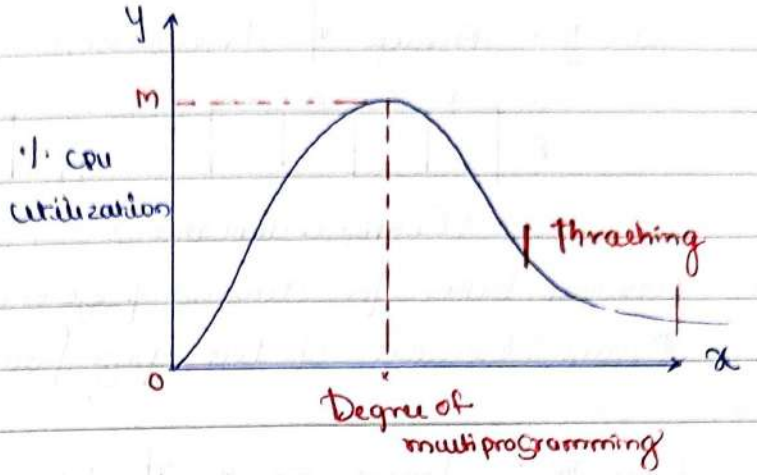
* Thrashing like deadlock is also an undesirable feature of O.S.

Reasons for Thrashing: (main)

1. High degree of multiprogramming
2. Frame allocation declines <lack of memory>

Other Reasons:

1. Page replacement policy
2. Page size:
   large: (less pages)
   ↓
   Less page faults
3. Programming techniques and data structures



Thrashing Control Strategies

1) Prevention

* Thrashing never occur.
* By controlling degree of multi-programming, thrashing can be controlled.
  ↓
  (Long term Scheduler)

2) Detection and Recovery.

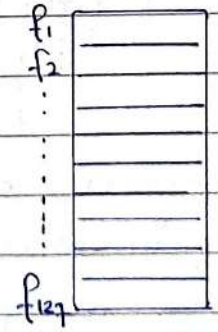
* Thrashing occurs
* Thrashing can be detected when there is low CPU utilization.
* Max no. of processes getting blocked
* High degree of multi-programming
* High disk utilization
* Recovery: Process suspension
  ↓
  (Medium Term Scheduler)

Case I: integer A[1...128][1...128]; P.S = 128 w; Row major order

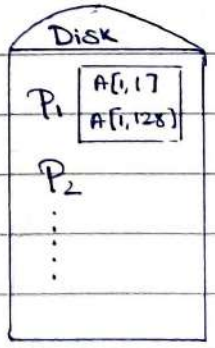
```
1.. for i ← 1 to 128
      for j ← 1 to 128
        A[j, i] = 1;

2.  for i ← 1 to 128
      for j ← 1 to 128
        A[i, j] = 1;
```

No of page faults:
a) Pure DP: 128^2
b) FIFO: 128

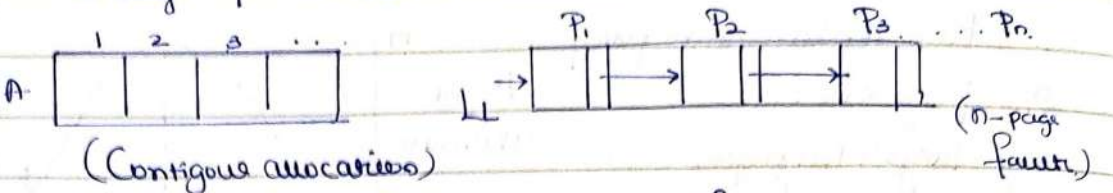
Locality of Reference



$128 \times 128 = 2^{14}$
$= 16k$



Case II : Array v/s Linked List



Q1. Which is better for demand page environment?

Ans Array, because of less page fault.

Case III : Linear Search v/s Binary Search

O(n) O(Logn)

Q1. Which is better for demand page environment?

Ans Linear Search may generate less page fault, (locality of reference).

Conclusion : A programming technique or a data structure is said to be good in demand page environment, if it satisfies locality model.

Working Set Strategy (model) : To minimize page fault rate and also utilize memory effectively.

< Principle of locality of reference >

10KB	2KB	28KB	15KB	
main() { . . . }	f() { . . . g(); }	g() { . . . h(); }	h() { . . . Scanf(); }	Prog. Size = 55KB Page Size = 1KB N. of pages = 55 50% Rule → 26 frames.

The basic idea of working set model is to estimate the size of locality in which the program is executing and only ask for those many frames.

W.S.S follows dynamic type memory allocation (Allocation of frames)

Working Set window (WSW) : Set of unique pages referred in the reference string during the past 'Δ' references.

eg:: Pr_i < 5, 4, 8, 2, 9, 12, 54, 58, 56, 53, 51, 52, 58, 56, 57, 54,

Δ=10 (Sliding window) t

WSW_i^t (Δ=10) = {51, 52, 53, 54, 56, 57, 58}

Δ = integer (value)

WSW_i^t (Δ=10) = 7

→ Let, No. of processes = 'n' <$P_1 \dots P_n$>
- → Total available frames = m
- → Demand of each = S_i = $WSWS_i^t$ (Δ=10)
- → Proces for frames
- → Total demand @ 't' = $\sum_{i=1}^{n} S_i = D$

I. D ≤ m [no-thrashing]

II. D < m [no thrashing → Inc. degree of multi-programming]

III. D > m [System is trashing]

Δ

Small
* Leads to more page faults

large
* Inefficient utilization of memory

Q1. Let the page reference and the Delta (Δ) be "c c d b c e c e a d" and 4 respectively. The initial working set a time t=0 contains the pages {a, d, e, where 'a' was referenced at time = t=0, 'd' was referenced at time t=-1, and 'e' was referenced at time t=-2. Determine the total number of page faults and the average number of page frames used by computing the working set at each reference.

t <	-2	-1	0	1	2	3	4	5	6	7	8	9	10 >
	e	d	a	c	c	d	b	c	e	c	e	a	d

$\Delta = 4$

$t_0 = \langle e\ d\ a \rangle$
$t_1 = \langle e\ d\ a\ c \rangle : 4$
$t_2 = \langle d\ a\ c \rangle : 3$
$t_3 = \langle a\ c\ d \rangle : 3$
$t_4 = \langle c\ d\ b \rangle : 3$
$t_5 = \langle d\ b\ c \rangle : 3$
$t_6 = \langle d\ b\ c\ e \rangle : 4$
$t_7 = \langle b\ e\ c \rangle : 3$
$t_8 = \langle c, e \rangle : 2$
$t_9 = \langle c\ e\ a \rangle : 3$
$t_{10} = \langle c\ e\ a\ d \rangle : 4$

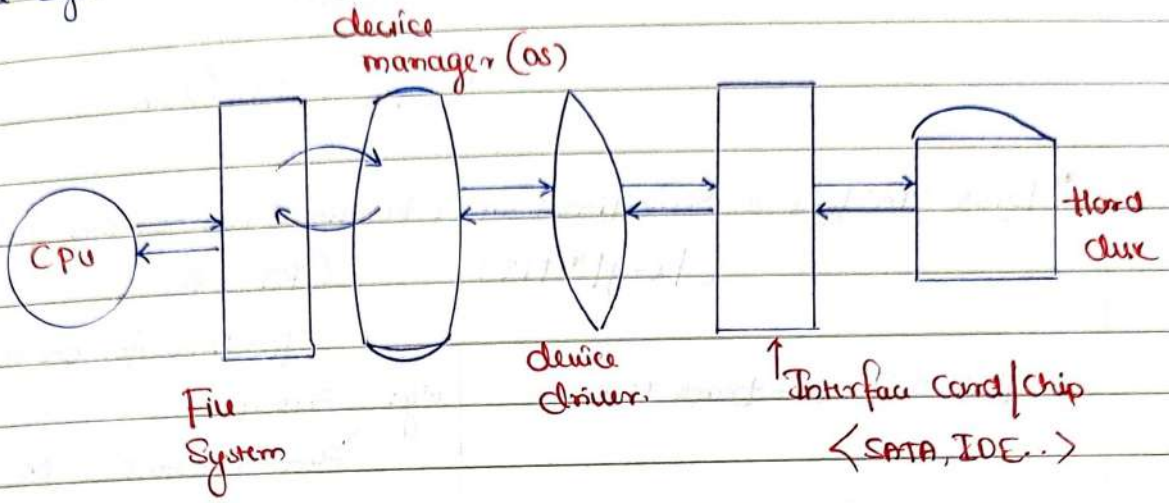
No. of page faults = 5

Avg. WSS$_{10}$ = $\frac{x}{10}$

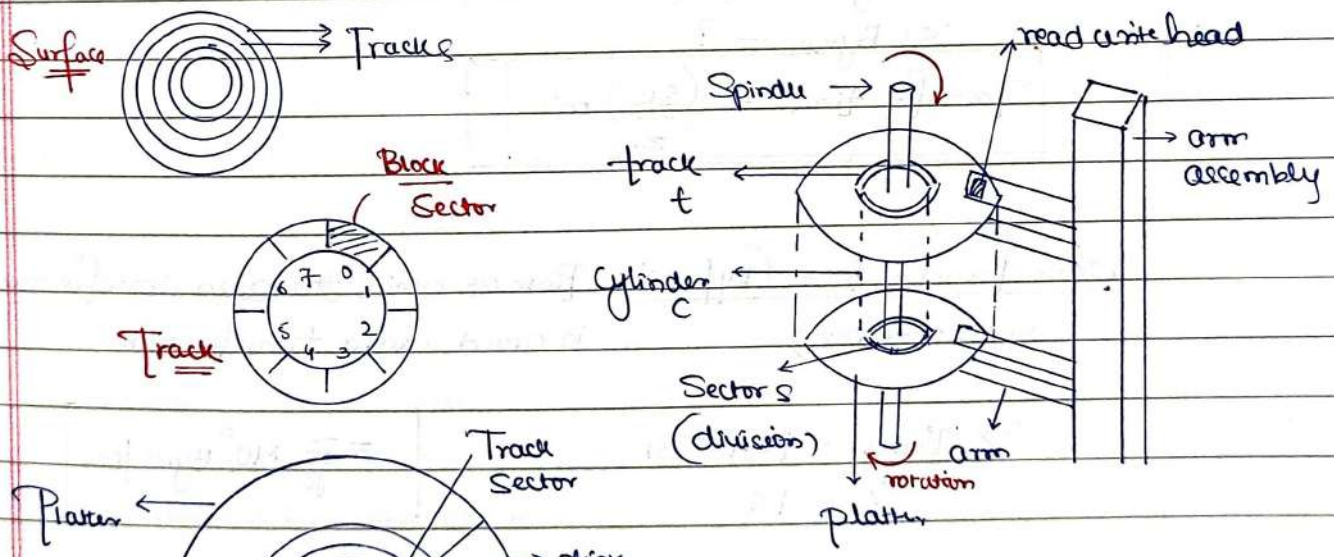
Date 7/3/23
Page

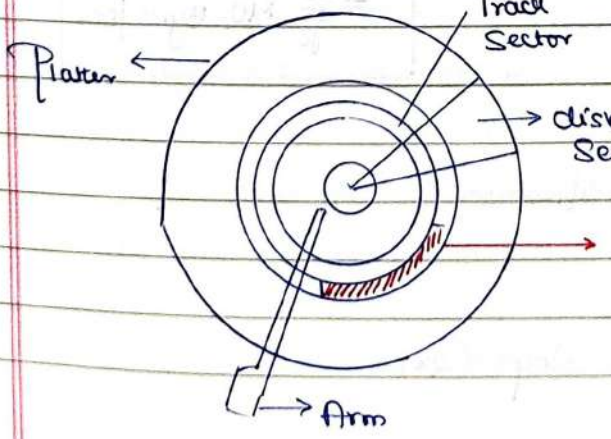
File System and Device Management

* File System is the only visible part of the operating System.



* Before storing the data on disk, it must be formatted in some file System (NTFS, ExFAT32, Ufs)
* Disk is a collection of circular/elliptical platters.
* Every platter has 2 ~~tracks~~ Surface, and consists of circular tracks.
* Each track consist of Sectors, Sector is measured in bytes





Cluster: a group of one or more adjacent Sectors

Disk → Platter → 2 surfaces → Tracks → Sectors

Sectors: Sec-No., Size (Bytes)

* **Disk IO Time** = Seek time (S.T.) + Rotational latency time + Transfer time

($|x-y|^{*}$ TTT) ($RT = (\frac{R}{2})$)

R – time for one rotation.

TTT = Track-track time

eg. 3600 rpm

3600 – rotations – 60s

1 rotation – ?

$R = \frac{60}{3600} = 16.67$ ms

→ Track Size = 'Z' bytes

→ Sector Size = 'S' bytes

→ Rotation time = 'R'

→ In one rotation, we can read one complete ~~time~~ Track.

'Z' Bytes – R (ms)

'S' Bytes – ?

$$\text{Transfer time} = \frac{(SR)}{Z} \text{ ms}$$

Data Transfer Rate (By/sec) (speed of disk): Rate at which the data transfer takes place is called data transfer rate.

'Z' Bytes – $R \times 10^{-3}$ (s)

? – 1s.

$$= \frac{Z}{R} \times 10^{3} \text{ Bytes/sec}$$

Q1 Consider the following disk specifications

Number of platters = 16

No. of tracks/surface = 512

No. of Sectors/track = 2048 (2k)

Sector offset = 12 bit $= 2^{12} = 4KB$

Average seek time = 30 ms

Disk rpm = 3600

Calculate:

(i) Unformatted Capacity of the disk:

$= 32 \times 512 \times 2K \times 4KB = 2^5 \times 2^{11} \times 2^{12} = 2^{37}$ ($+2^5$ missed to write.)

$= 128\,GB$

(ii) Io time / Sector:

$ST + LT + TT = 30ms + 8.3ms + 0 = 39\,ms$

Track Size = $2K \times 4KB$

$= 8MB$

$\left(\frac{4KB}{8MB} \times 16.6\,ms\right) \leftarrow TT.$

(iii) Data transfer rate:

$\frac{Ts}{8MB} - \frac{R}{16.6 \times 10^{-3} s}$

$? - 1s$

$\Rightarrow \frac{8MB}{16.6} \times 10^3 s. = 1/2\ GB/s$

(iv) Sector address = 25 bit required to select one sector.

4	1	9	11	12
P/t	Surf	Track	Sec	d

25 bit

37 bit

Disk capacity $= 2^{37} = 128\,GB$ (another approach)

Hw

Q2. Consider a disk with following specs:

No of surfaces = 64

Outer diameter = 16 cm, Inner dia = 4 cm

Inner track space = 0.1 mm

Max density = 8000 bit/cm

Calculate the unformatted Capacity of disk.

No. of tracks $= \frac{6cm}{0.1mm} = 600$

Inner Track Size = $2\pi r$ = $\pi \times d$
= $3.14 \times 4 = 12.56$ cm

8000 bits — 1 cm
? — 12.56 cm.

Track Size = $\frac{8000 \times 12.56}{8}$
= 1000×12.56 B.

Disk Capacity = $64 \times 600 \times 12.56$ KB
= $64 \times 6 \times 1256$ KB
= $64 \times 6 \times 1.256$ MB
= 482.3 MB = 0.48 GB

∴ Disk Cap = 0.48 ≈ 0.5 GB

Q3. How long does it take to load a 64KB program from a disk whose avg. Seek time is 30 ms, Rotation time is 20ms, Track Size is 32 Kbytes, Page Size is 4Kbytes, Assume that pages of program are distributed randomly around the disk. What will be the % Saving in time if 50% of the pages of program are Contigous?

S.T = 30 ms
R = 20ms
T.S = 32KB
P.S = 4KB
N = $\frac{64KB}{4KB}$ = N = 16

Time to load program = T. to load a page * N
Time to load page = ST + L.T + T.T
= 30 ms + 10 ms + 2.5 ms
= 42.5 ms

Prog-loading (T) = 42.5 × 16
= 680 ms = 0.68 s

32KB — 20ms
4KB — ?
$\frac{4KB}{32KB} \times 20 = 2.5$ ms

When pages are Contigous then only one seek and one latency:
= (30 ms + 10 ms + 2.5 × 8) + 8 × 42.5 ms
(CG) (NCG)
= 60 ms + 340 ms = 400 ms = 0.4 s

(i) 0.68 s
(ii) 0.4 s = 0.28 s (diff)

% Savings = $\left(\frac{0.28}{0.68}\right)$ = 41% percentage Savings = 41

Date 9/3/23

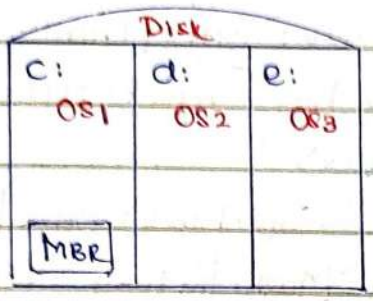
File System and Device Management (Continued)

Logical Structure of disk: [Formatting Process]

↳ Placing infrastructure on disk

↓

[for effective Storage and faster retrival.]



"Multi boot Computer".

→ Disk divided into partitions (Volumes / Mini disks)

(OS) Primary (Bootable) : Data + S/W

Extended / Logical drives (Non-bootable) : Data + S/W

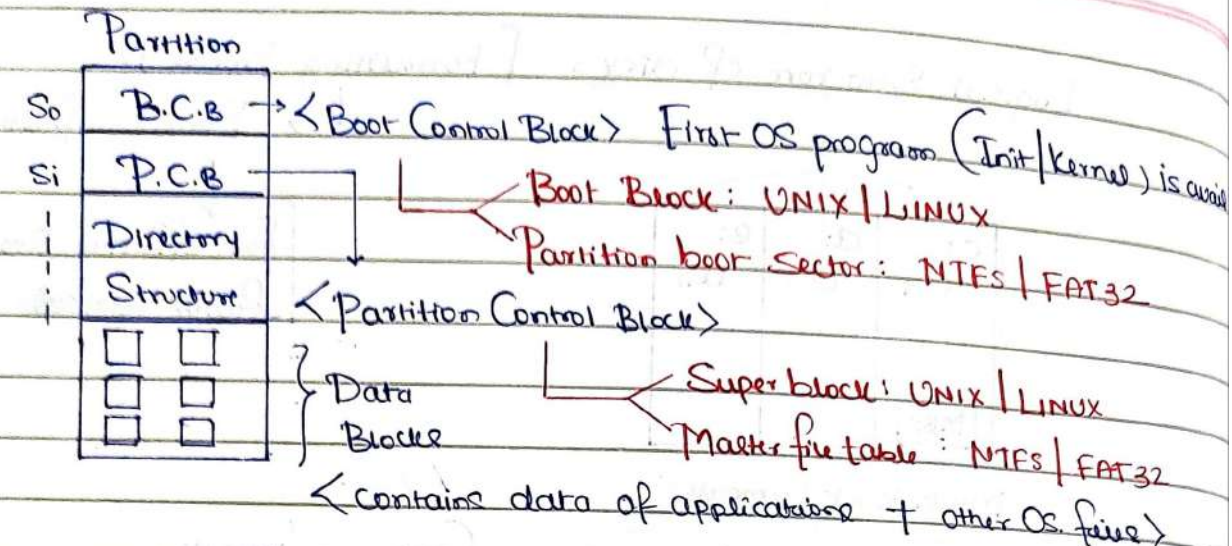
MBR <Master Boot record>

- Partition Table
- Boot Loader

Steps in Booting Process:

1. Power On or Switch on.
2. POST <Power on Self test>
3. BIOS <Basic IO System>
4. BootStrap loads MBR into RAM and hands over the control to boot loader
5. Boot loader reads partition table and display options if available
6. Boot loader will load the kernel program from disk into memory.

Partition Structure [What data structure (infrastructure) are embossed on the disk/partition after formatting]



Disk is divided into 2 parts: 1. Control data
2. User data.

Some of the memory will be taken as memory overheads (B.C.B + P.C.B + D. Structure), that is the reason in Pendrives the size mentioned will be 128 GB but memory actual avail is 114GB.

File v/s Directory:

File: File is a collection of logically related records of entity. It is an abstract data type < Defination, Representation, Operations, Attributes>

Representation: Flat (Series of Bytes), Record (Series of records) — Hierarchical Tree (B-trees, B+..)

Operations: Create, Open, read, write, truncate, seek, close, copy, move, delete,..

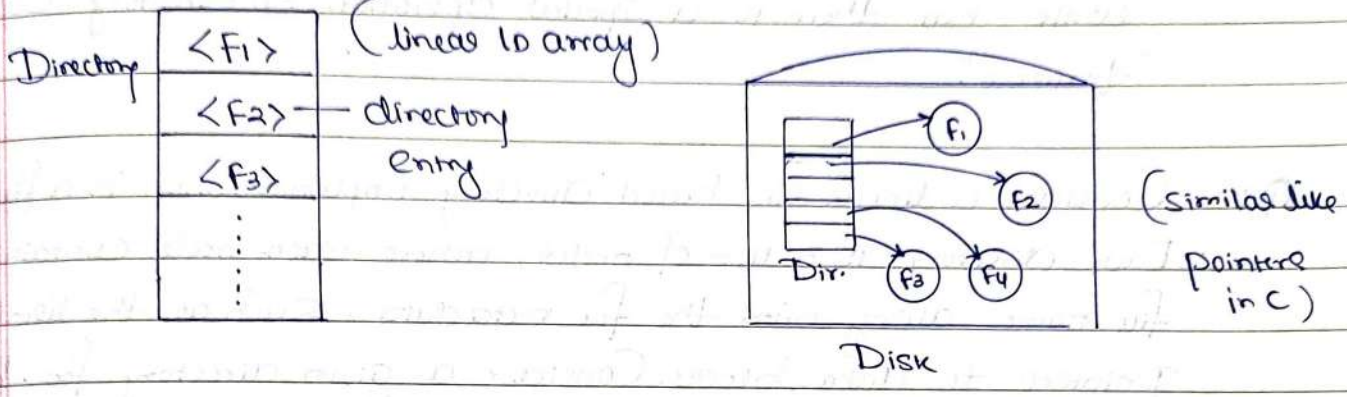
Attributes: Name, identifier, type, protection, time, date, location... (different OS have diff attributes)

* File's attributes are stored in File Control Block (FCB)

UNIX (I-node)

Windows (Directory entry)

Directory: It is a collection of files. It is also a special file (data about other files → "Meta data")
Directory contains meta data of files.



Directory Structure.

* "Single level directory"

directory	cat	bo	a	test	data	mail	cont	hex	records

(Single directory for all users)

* no sub directory
* Simple to implement
* Search time is more
* Name conflict.

* Two level directory
  * Seperate directory for each user
  * Path name
  * Can have the same file name for diff. user
  * efficient Searching
  * no grouping Capability

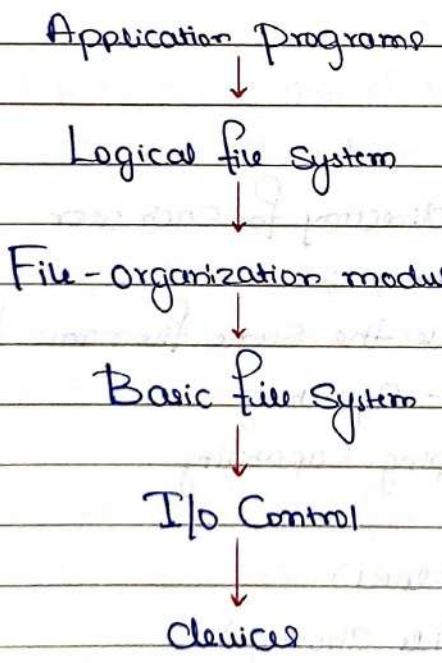
* Tree Structured directories (Multi level)
* Acyclic graph directories (File Sharing)
  - also called directed acyclic graph directory for file sharing purposes
  - Link count : Counts the no. of people sharing a file.
* General graph directory (Cycles)

* To search a file, we may have to traverse the directory, for that we may use Graph traversale like DFS, BFS.
* Directory as a file also supports operations like open, read, write but there is a special operation on directory called "traverse".

Q1. Consider a linear list based directory implementation in a file system. Each directory is a list of nodes, where each node contains the file name along with the file metadata, such as the list of pointers to data blocks. Consider a given directory foo. Which of the following operations will necessarily require a full scan of foo for successful completion?

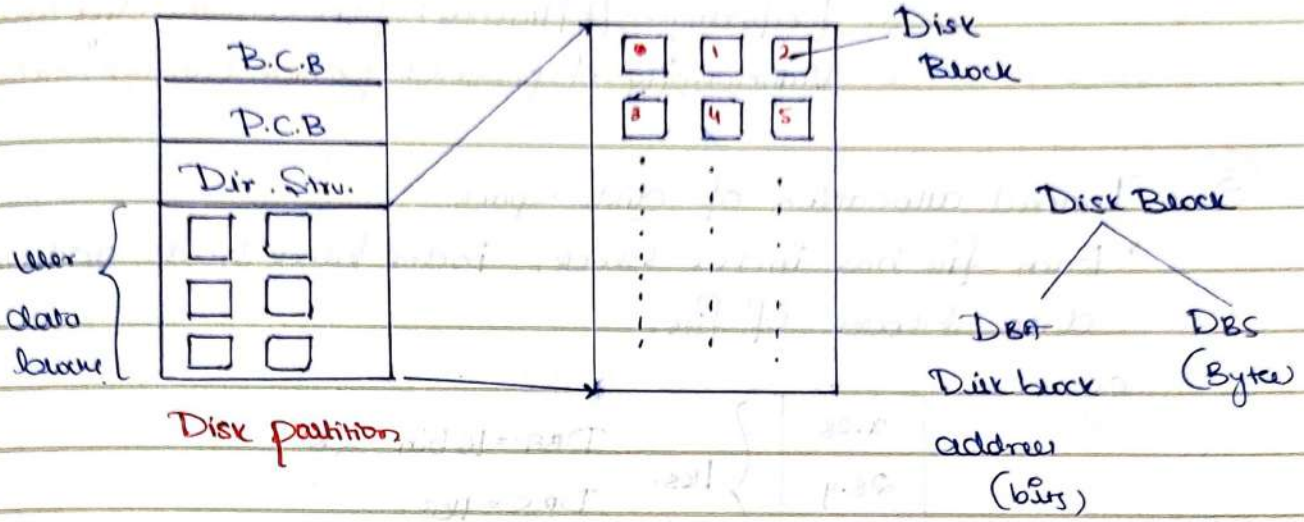
Ans Creation of a new file in foo.
Renaming of an existing file in foo.

Layered File System.



11/3/23

Allocation Methods



* DBA = 16 bits
DBS = 1 KB * Max possible file size on the disk = Max possible (given) disk size.

eg:: no. of blocks = 2^{16} = 64K
∴ Disk Size = 64K × 1KB = 64MB

★ Maximum possible disk Size = $(2^{DBA}) \times DBS$

(1) Contigous Allocation of disk space.

Performance: 1. internal fragmentation : Yes (last block)
2. external fragmentation : Yes
3. Increasing file size : Inflexible
4. type of access : Sequential / Random/direct access (faster)

(2) Linked allocation of disk Space.

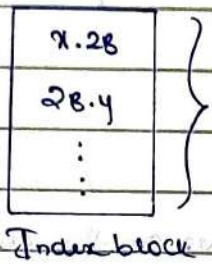
Performance: 1. Internal fragmentation : Yes (last block)
2. External fragmentation : No
3. Increasing file size : Flexible.

4. Type of access : only sequential (slower)
5. Performance / Efficiency : ptrs consume disk space
6. Vulnerability of pointers : pointers can get broken.

(3) Indexed allocation of disk space

* Each file has index block, index block holds address of data blocks of file.

eg::



DBA = 16 bits = 2B
DBS = 1kB.
No. of addresses = $\frac{1kB}{2B} = \underline{512}$

eg:: DBA = 32 bits
DBS = 4kB.

Max possible file size with index block : ?

No. of addresses = $\frac{4kB}{4B} = \underline{1K}$

File size = 1K x 4KB = $\underline{4MB}$

Q1. A file system supports a DBS = 4kB, each block can hold 512 addresses (ptrs). What is the size of DBA in bits?

$$DBA = \frac{DBS}{N.\ addresses.}$$

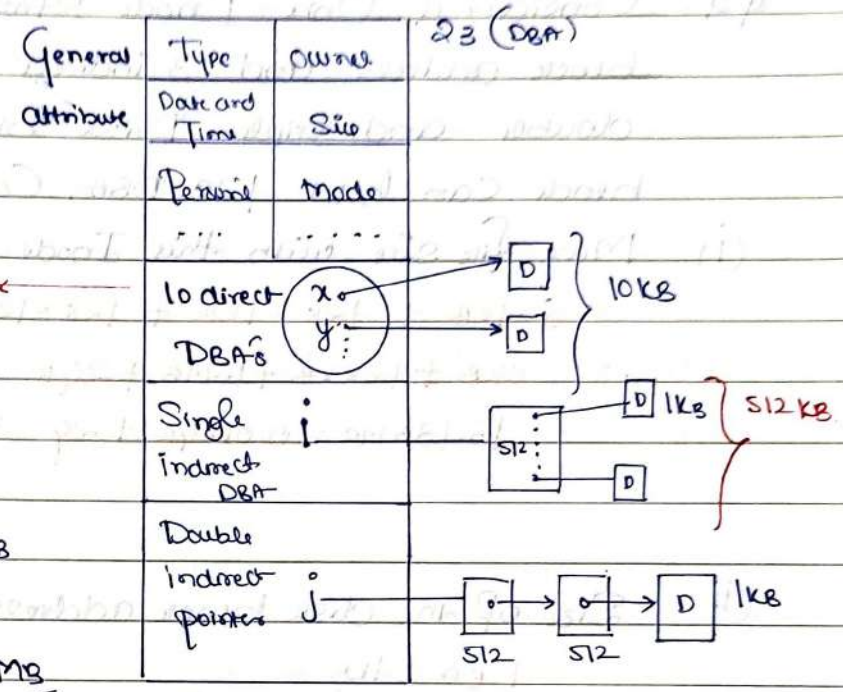
$$DBA = \frac{4kB}{512} = \frac{2^{12}}{2^9} = 2^3 = 8B = \underline{64\ bits}$$

Case Studies of O.S

(1.) UNIX / LINUX : each file is associated with a I-node

File N	I-node
KKC	28

I-node Structure :



→ With double indirect

$512 \times 512 \times 1KB = 2^{28} = 512MB$

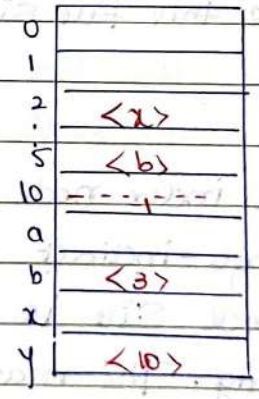
→ Total file size = $256 * 522 MB$

(2) DOS / WINDOWS

File Name	(Other attributes)	First DBA	Last DBA
KKC	- - - - - -	5	10

Directory

Block info.



Linked allocation

* No. of entries in FAT (MFT) = no. of blocks
* Fat entry contains addresses (DBA) of next data block

FAT-(32) FAT entry size DBA

Q1. In a file allocation system, which of the following allocation scheme(s) can be used if no external fragmentation is allowed?

Ans (i) Linked allocation
(ii) Indexed allocation

Q2. Consider a Unix-Inode structure that has 8 direct disk block address and 3 indirect block address, namely single, double and triple. Disk block size is 1KB and each block can hold 128 DBA. Calculate:

(i) Max file size with this Inode structure: 16.136 MB

$= 8 \times 1KB + 128 \times 1KB + 128 \times 128 KB + 128 \times 128 \times 128 KB.$

$= 8KB + 128KB + 16MB + 2GB$

$16.136MB = 0.016GB + 2G = 2.016GB$

$= \underline{2GB}.$

(ii) Size of the disk block address:

$DBA = \frac{1KB}{128} = \frac{2^{10}}{2^7} = 2^3 = \underline{64 Bit}$

(iii) Is this file size possible over given disk?

Q3. The index node of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4KB, and the disk block address is 32 bit long. The max possible file size is 4GB GB.

Ans No. addressed $= \frac{4KB}{4B} = 1K$

File Size $= 12 \times 4KB + 1K \times 4KB + 1K \times 1K \times 4KB$

$= 48KB + 4MB + 4GB$

$= 4.004GB \sim 4GB$

Q4 A file system with 300GB disk uses a file descriptor with 8 DBA, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 B and size of each DBA is 8 Bytes. Max possible file size in the file system is

Ans DBS = 128 B

DBA = 8B No. addressed = $\frac{128}{8} = 16$

File Size = 8×128B + 16×128B + 16×16×128 B
= 1KB + 2KB + 32KB
= 35KB

Q5. The data blocks of a very large file in the Unix file System are alloced using:

Ans An extension of indexed allocation

Q6 Using a larger block size in a fixed block size file system leads to

Ans Better disk throughput and poor disk space utilization

Q7 A FAT based file System is being used and total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size is 10^3 Bytes, the maximum size of a file that can be stored on this disk in units of 10^6 Bytes is 99.6

$N = \frac{100 \times 10^6}{10^3} = 100 \times 10^3$

Max. file size = Given disk size − FAT size
$= 100 \times 10^6 - 0.4 \times 10^6$
$= 99.6 \times 10^6$

FAT Size = $100 \times 10^3 \times 4B$
$= 400 \times 10^3 B$
$= 4 \times 10^5$
$= 0.4 \times 10^6 B$.

Q8 A file system with one level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk block 0 : Boot control block
Disk block 1 : File allocation system/table, consisting of 10-bit entry per data block, representing the data block address of next data block in file.
Disk block 2, 3 : Directory with 32-bit entry per file.
Disk block 4 : Data block 1
Disk block 5 : Data block 2, 3, etc.

(i) What is the max possible number of files?
(ii) What is the max possible file size in bytes?

Ans Number of files (max) = no. of entries

$= \frac{8KB}{4B} = 2K = 2048$

DBA = 10 bit
DBS = 4 KB
No. of blocks = 2^{10}
$= 1024 - 4 = 1020$

Max file size = Disk size − (control overheads)
$= 1024 - 4 = 1020 \times 4\,KB$
$= 4080\,KB$
$= 4.98\,MB$

Q9. Consider a file system that stores 128 DBA in the index table of the directory. Disk block size is 4 KB. If the file size is less than 128 Blocks, then these addresses act as direct data block addresses.
However, if the file size is more than 128 Blocks, then the 128 addresses in the index table point to next level index blocks, each of which contains 256 data block's address. What is the max file size in the system?

Ans. Min file size = $128 \times 4\,KB$
$= 512\,KB$

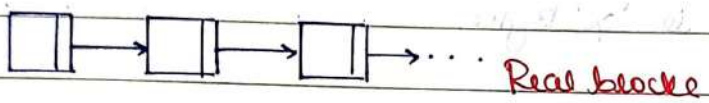
$128 \times 256 \times 4\,KB$
$= 2^7 \times 2^8 \times 2^{12} = 2^{27} = 128\,MB$

Disk free space managment algorithms :

Asume: disk - 20 MB;
DBS : 1KB
DBA : 16 bits
N. blocks = $\frac{20MB}{1KB} = 20K$

Given disk size	≤	Max. possible D.S
		$2^{DBA} \times DBS$

1. **Free linked list** : linked list of free blocks.



2. **Free list** : Keeps track of free blocks only, faster compared to bit map.

Q. How many blocks are (free list) are needed to store 20K free DBS?
Ans: 1 Block - 512 free DBA
? - 20K free " $= \frac{20K}{512} = 40$

3. **Bit-Map / Vector**

: Associate a Binary bit with each block
: Bit map keeps track of all blocks.
: Slower Compared to bit-map.

Q. How many blocks of disk are needed to store our bit map?
Ans: DBS = 1KB
DBA = 16 bits
1 Block → 8K bits
? → 20K bits
$= \frac{20K}{8K} = 2.5 \simeq 3$ Blocks

4. Counter method: when we have many contigous free blocks.

	Start free DBA.	No. of CG free blocks
1.	5	45
2.	85	200
3.	325	125
4.	600	25

eg: New file F=20 blocks
Consumption can be done by first fit, best fit, worst fit.

Q1. Consider a disk with 'B' blocks, 'F' are free, DBA is 'D' bits. Disk block size is 'X' Bytes.

(A) Calculate:

(i) Given disk size: "B * X" Bytes.

(ii) Max. possible disk size = $2^D \times X$ Bytes

(iii) Relation b/w 'B' and 'D' : $B \leq 2^D$.

(B) What is the condition in which Free list uses less space than Bit-Map?

: $F * D < B$

Q2 The beginning of a free space bit map looks like after the disk partition is first formatted: 1000 0000 0000 0000. The system always searches for free blocks starting at the lowest numbered block, so after writing file A, which uses 6 blocks, the bit map looks like this: 1111 1110 0000 0000. Show the bit map after each of the following additional actions at the code.

Ans
* File B is written, using 5 Blocks
* File deleted

Q3. A file system uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from cache is 1 ms and to read a block from disk is 10ms. Assume that the cost of checking

whether a block exists in the cache is negligible. Available cache sizes are in multiples of 10 MB.

The smallest cache size required to ensure an average read latency of less than 6ms is 30 MB.

Ans

$6ms = x(1ms) + (1-x)(10ms)$

$= x + 10 - 10x$

$-4 = -9x$

$x = 4/9 = 0.45.$

Miss ratio = 0.55

55% ⇒ 25 MB

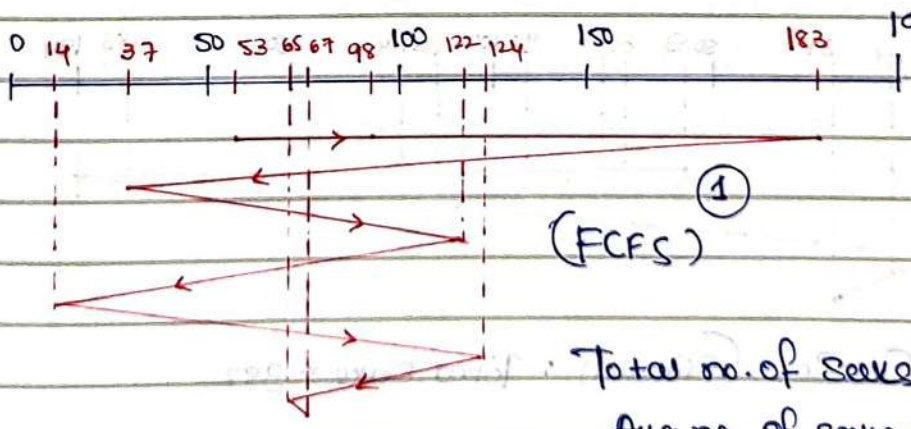
= 30 MB (for this miss ratio reduces, EMAT ↓)

Q4. The amount of disk space that must be available for page storage is related to maximum number of processes 'N', the number of bytes in Virtual Address Space 'B' and the number of bytes in RAM 'R'. Give an expression for worst case disk space required.

Ans. N * S Bytes

Device Scheduling:

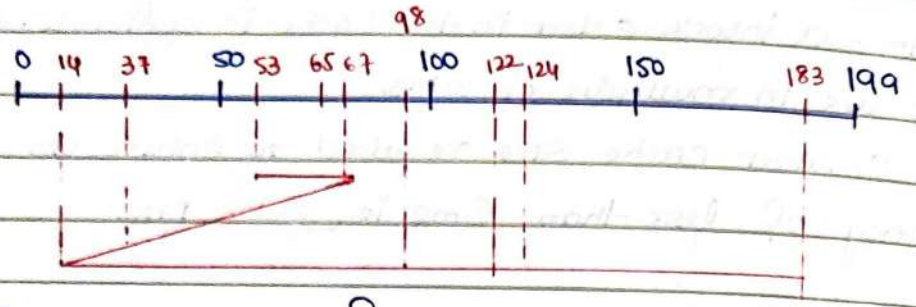
Process requests : 98, 183, 37, 122, 14, 124, 65, 67. (Track no./cylinder no.)



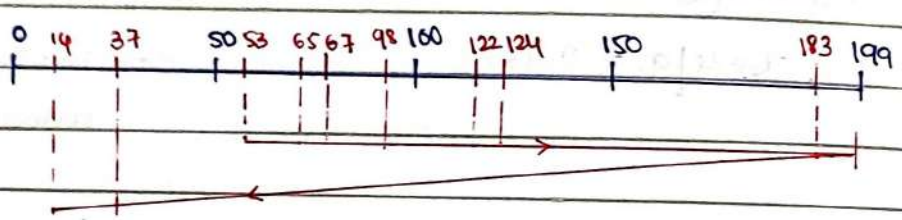
Total no. of seeks = 640

Avg no. of seeks = 640/80 = 80

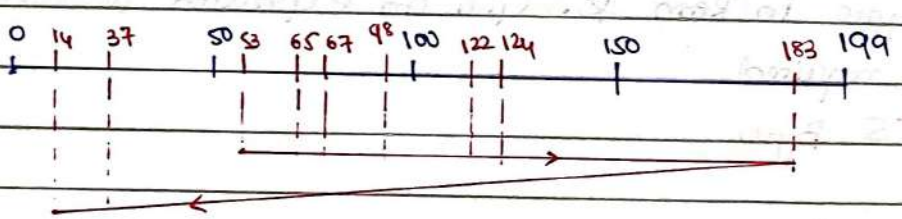
* Objective of disk scheduler is to reduce average seek time.



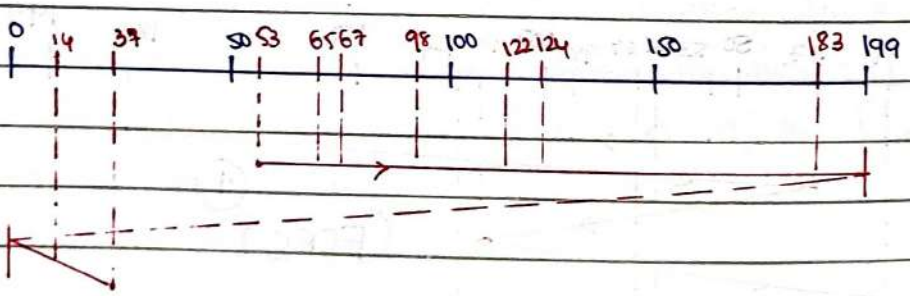
(2.) Shortest Seek time first : Total Seeks = 236



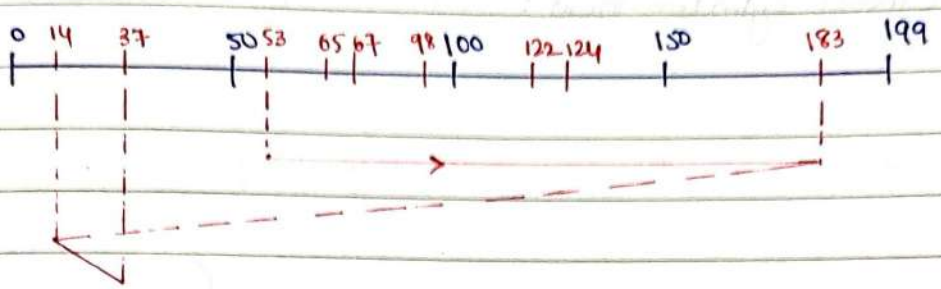
(3) Scan / Elevator : Total Seeks : 331.



(4) Look: Total Seeks = 299.

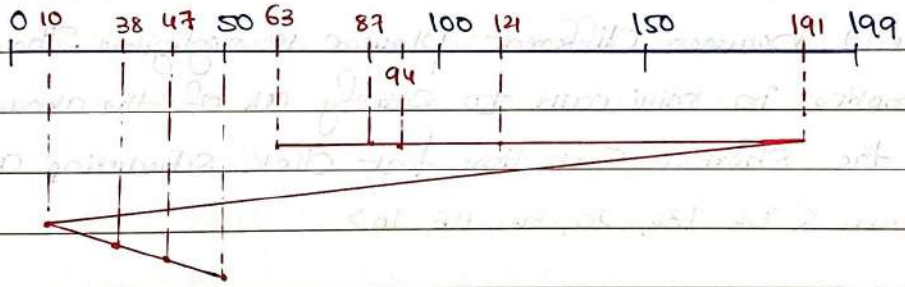


(5) C-Scan (Circular) : Total Seeks = 382



⑥ C-Look : Total Seeks : 322

Q1. Consider a disk queue with requests for I/O to blocks on cylinder 47, 38, 121, 191, 87, 11, 92, 10. The C-look scheduling algorithm is used. The head is initially at cylinder no. 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement incurred while servicing these requests is: 346



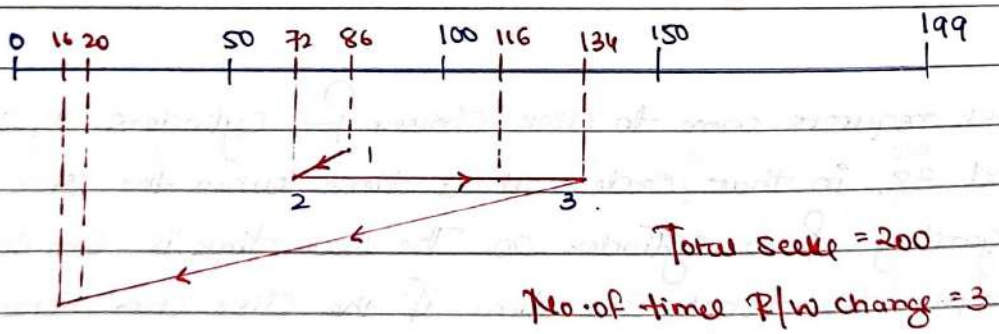
Q2. Disk requests come to disk driver for cylinders 10, 22, 20, 2, 40, 06 and 38, in that order at a time when the disk drive is regarding from cylinder 20. The seek time is 6ms sec per cylinder Compute the total seek time if the disk arm scheduling algorithm is :

(a) First come first serve.

(b) Closest cylinder next:

Q4. Consider a storage disk with 4 platters (0-43) 200 cylinders (0-199) and 256 sectors per track (numbered as 0, 1, 255). The following 6 disk requests of the form [Sect.no, Cyl.no, platter no] are recieved by disk controller at same time [120, 72, 2], [184, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 6, 1].
Currently head is at no. 100 of cylinder 80 and moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatt and for reversing the direction of the head movement once is 15 milliwatt. Power dissipation associated with rotational latency and switching of head between different platters is negligible. The total power consumption in milliwatts to satisfy all of the above disk request using the shortest seek time first disk scheduling algorithm is 85

Ans. Request <72, 134, 20, 86, 116, 16>



Total seeks = 200 − 20×2 = 40

No. of times R/W change its direction = 3 − 3×15 = 45

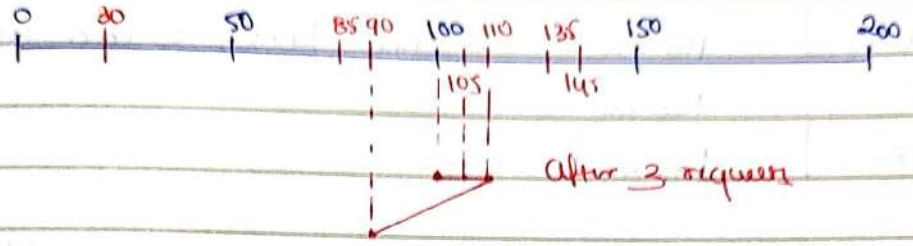
85

∴ 85 milli watts

Q5. Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135, 145. If SSTF is being used for scheduling the disk access

the request for cylinder 90 is serviced after servicing 3 no of request

Ans.



Q6 Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used in FCFS. If FCFS is replaced by SSTF, claimed by vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs?

Ans 0% (no improvement)

Q7. Consider the following five disk access requests of the form (req. id, cylinder number) that are present in the disk scheduler queue at a given time (P, 155), (Q, 85), (R, 110), (S, 30), (T, 115)

Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek time first to serve the requests. Which of statements is false?

Q8. Consider a disk with following details:

Track-track time = 1ms.

Current head position = 65

Direction: moving towards higher tracks with higher track = 199

Current clock time = 160ms.

Consider the following data set:

HW

Track no.	T. of arrival
12	65 ms
85	80 ms
40	110 ms
100	120 ms
75	175 ms

Calculate time of decision, Pending requests, Head position, Selected request, Seek time using FCFS, C-Scan, SSTF, SCAN, Look, C-Look algorithms.