OPERATING SYSTEMS (OS)

21/06/2021

by Dr. Khaleel Khan
Email: {Khaleelrkhan@gmail.com}

{Short Notes + Practice Questions}

Pre-Requisite:

- COA
- Data Structures
- Programming Language (C/C++)

MSQ

Text-Books:

1) O.S Concepts – Galvin (9th Ed.) *
2) O.S – Stallings
3) Mod. OS – Tanenbaum

Recd Notes in ACE Material

{Preparation of Notes}

- Long (detailed Notes) (Lectures + Extracts from Text Book)
- Short Notes (end-of-course) Revision

Teaching Schedule :

1. Introduction & Background
2. Process Management : Major (40)
   - Process Concepts
   - CPU Scheduling (*)
   - IPC & Synchronization (Coordination) (**)
   - Concurrency
   - Deadlocks (*)
   - Threads

No. of Hrs : 60

3. Memory Subsystem (Mem)
   - Abstract view of Memory (Addressing vs Capacity)
   - Loading, linking & Address Binding
   - Basic Techniques (*)
   - Virtual Memory (**)
4. File Systems & {Device Management (IO)}
   - Interface
   - Implementation Aspects (*)
   - Disk/IO Scheduling (*)

Types of Questions — Reasoning

Theory — Numerical

Concepts (Riddles) ← → Aptitude

{GATE + Interviews + other Exams}

Jobs — Higher Studies

Problem Solving : Theoretic Foundations + Typical Questions + C·R·P (Volume I)

Hit ratio: 99.99%

Essential ↓ ✓ (Exercises)

{Handout}

Prepare in Worst Case + Best Case in exam

www.khaleelrkhan.info

+ {Bit & Bytes}

Test Series: after Completing the Subject:

{Time Management} {To Know your preparedness of topics}

(depth + Breadth)

(Peer-Comparison) +

(Data wise Hours Big-Data) ×

(I am going to start from Scratch

What is OS

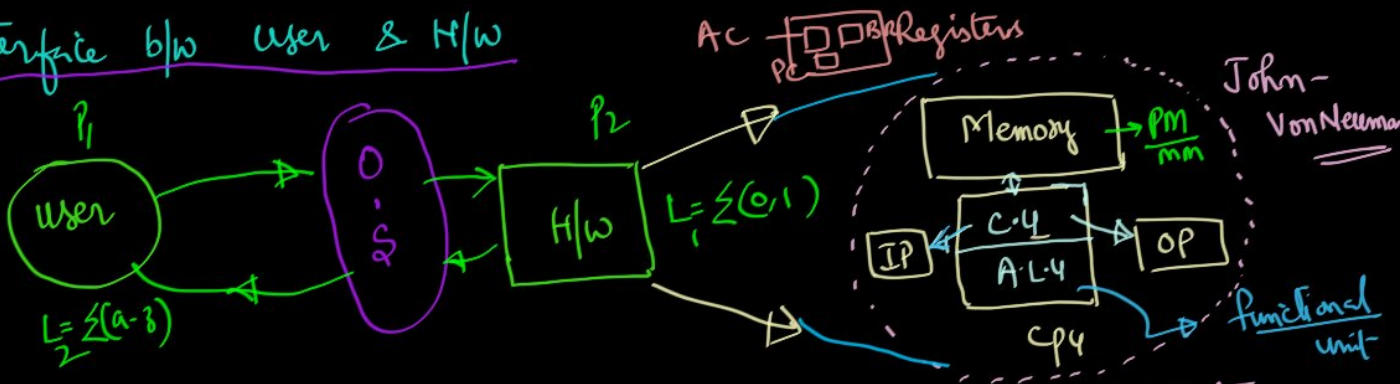
What is an O.S;

- → is an interface between User and Hardware
- → Control Program(s); Control S/W (System S/W)
- → Resource Manager ← Allocating, Protecting, Deallocating
- → Set of utilities (pm; mm; fm) to simplify application development.
- → Acts like a government;

Resource:
- H/W: CPU; Memory; I/O; Registers, clock, ...
- S/W: Files (System) like device Driver; Semaphores, Monitors, pipes, Signals, Messages, ...

→ (OS is a Services Provider and users/applications/Programmers are Service users)

Interface b/w user & H/w



Architecture

{C.U} ← generation of timing/control signals:
Sequencing of Micro-operations: (M-ops)

Arith
Logical
Data Transfer ← R-R, R-M

(operations that are performed on the data stored in registers, during a single cpu/clock cycle)

Eg: a = b+c;

```
Load  R1, b
Load  R2, c
Add   R1, R2
Store a,  R1
```

As per Von-Neumann architecture all Sec. Storage devices are treated as IO-devices

Memory < Primary (Main) : RAM / Rom / Cache / Registers : Faster
Secondary (Auxiliary) : Disk (PDs / Disks / Tapes / Cartridges : Slower

(i) Speed : order of access_time of M.M : ns (10^{-9} s)
" " " " " S.S : ms (10^{-3} s)

(ii) Cost : M.M is costlier

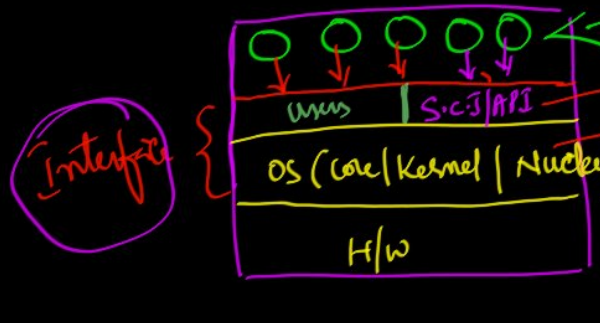
(iii) Size : Sec. Storage is larger size : (TB)
Mem Memory " order : (GB)

(iv) Persistence / Volatility : P.M is Volatile (except Rom) → Boot Loader (Booting of O.S)
S.M is Non Volatile

The basic objective of O.S as an interface is to provide an environment / Platform (using the H/w) to & its users / application.

ns → 10^{-9}
ms → 10^{-3}
μs → 10^{-6}

TB → 10^{12}
GB → 10^{9}
MB → 10^{6} (2^{20})
KB → 10^{3} (2^{10})



User applications

Level 1: user-os interface : S.C.I/API (Command Interpreter) L1

Level 2: S-H/W interface (Core) L2

L_2

Interface L_1

- Command Interpreter:
  - GUI based : Desktop (app) — WIN
  - Text based : Shell ($; C:>) — (DOS/UNIX)
- for Programmers : OS-Programmers interface : System call interface → fork()

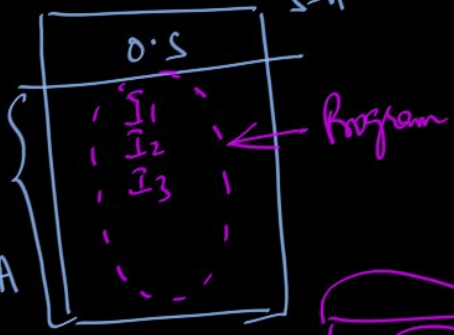
→ Each Computer Architecture has its own computing Model :

∴ The Computing Model of Von-Neumann Architecture is : Stored Program Concept

(Any Program must be stored in Main Memory b/f execution)

Architecture ✓
+
Comp. Model ✓
+
Flow of Control ✓
(Sequential Flow)

(WIN + LINUX + UNIX + MAC) ; → IO



Other Architectures :

Harvard ;
R·T· Arch ;
Multi processor ;
distributed ;
Cell Phone ;
└ Android
IOS, WIN ; Symbian

Note : (Architecture changes OS (design) changes)

→ None of the given definitions are complete:
↳ Since diff. People working in diff. environments view O.S differently.

Functions & Goals:

Carry out

OS
↓
Von-Neumann
↓
Convenient

UNIX | LINUX | WIN

A → Convenience (user friendly)
B → Efficiency (effective utilizij of resources)
C → Portability (run across diff. Platforms)
D → Scalability (Ability to evolve)
→ Robustness (Strong enough to bear error)
→ Reliability

Q) Which of the following should be primary goal of O.S?

Eg: RTOS: {VxWorks} → (NASA) dev. O.S

Kid:- Games Cartoons
Off Sec: Data Mang
Tech: Sp. devices
1st yr: C/C++ Prog.
4th: Sys. Call Prog - Shell Prog.

Convenience vs Efficiency : ← Secondary

Primary

WIN ✓ 4 — UNIX | DOS ✓ → More eff ; Reliable

Easy to use

From majority Appl-domain Perspective

Appl. / Computing domain :

(i) Person Computing : (Desktop & Laptops) : Home / office : Convenience

(ii) Real Time Systems : [efficiency + Reliability + Robustness]

Adv. Domains { (iii) Distributed " : efficiency + Scalability

(iv) Peer-peer "

(v) Cellular " (Mobile Computing) : Convenience + efficiency (Power) ✓

Types of O.S: (von-Neumann Architecture)

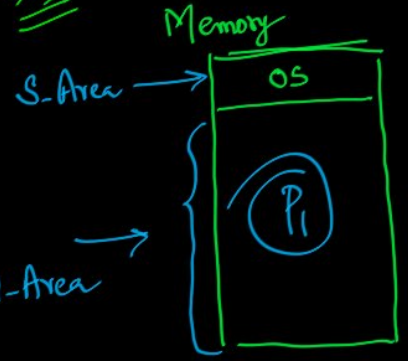
Generations (Evolutionary Perspective)

1. 1st Gen (1930-40's): No O.S (Card-Readers & Punch Cards)
2. 2nd Gen (1940-50s): No O.S (Mag. Tapes) → Batch Processing
3. 3rd Gen (1950-60's): {Disk Technology (HD + FD)} (chips) P.C (OS-Concept)
   1) Uniprogrammed 2) Multiprogrammed
4. 4th Gen (Present): OS for Handheld devices + Adv. Architecture

22/6/2021

Uniprogramming vs Multiprogramming :

Ex: DOS



: OS capable of Managing a single Program in Memory.

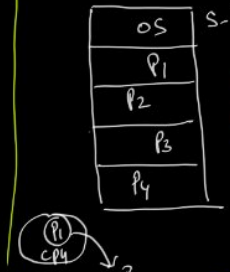
one Program in Memory is not sufficient to keep the cpu busy all the time

∴ ineffective utiliz. of cpu (Idleness of cpu)

I/O

(UNIX, LINUX, WIN)

: "ability of O.S to manage multiple ready to run Programs in Memory."



"Multiplexing of cpu among ready Progs in Mem.

"Maximize cpu utiliz"

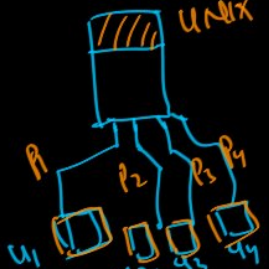
↳ Throughput:

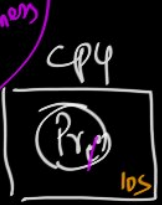
(Throughput: No. of Programs Completed Per unit Time)

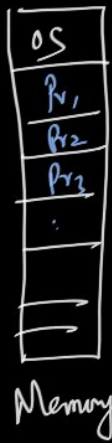
Types of Multiprogramming

- Users Persp
  - Single user
  - Multi-user
    - UNIX - Terminal
- Implementation (*)
  - (U; L; WIN; MAC) (*)
  - Pre Emptive (forceful deallocation)
    - → Complete
    - → IO|SC
    - → Pre Emptive Condition
      - (i) Priority
      - (ii) Time
  - Non Pre Emptive
    - No - forceful dealloc.
    - (Voluntary release)
      - (i) Completes
      - (ii) (IO / Sys. Call / res_req)
- W 3.0
- Lack of (responsive / Interactiveness)

If time is used for enforcing Pre Emption then such systems are known (Time shared O.S)







To Study the design of a Pre Emptive based M.Pr O.S, that can run on V.N Architec.

Multi-Tasking: WIN: Pre Emptive based M.Pr.O.S; in WIN

UNIX: (program)

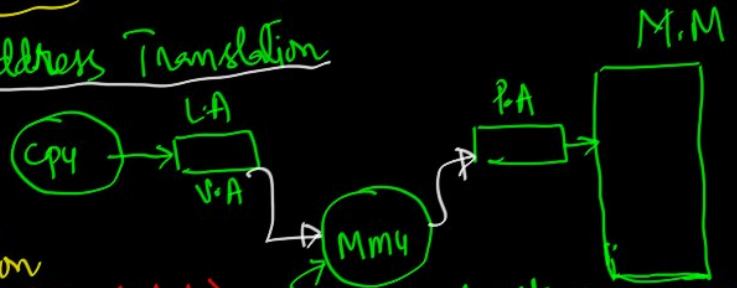
(Program = Task = App)

Architectural Requirement for Impl. a M.Pr.O.S: (V.N Architec.)

H/w

✓ (i) IO (Sec. Storage devices); D.M.A (Direct Memory Access) CO

✓ (ii) Memory: Address Translation

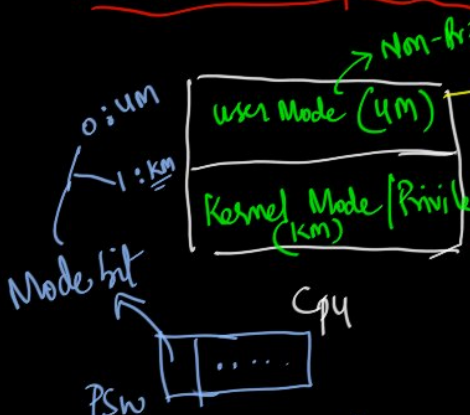


Mem. Mgmnt unit (P.I + R.R + B.R)

(iii) CPU: Dual Mode operation

(User Mode + Kernel Mode)

Dual Mode operation :



→ user appls run in UM | PreEmptive

→ All OS routines run in K-M | Non-PreEmptive < Atomic >

Mode-Shifting :

→ O·S is a Service Provider (KM)

→ users are Service users (appls → UM)

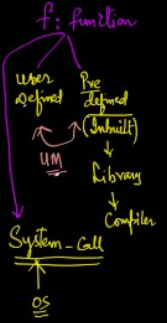
"Many a times it become necessary to shift the mode from U → K inorder to avail O·S services & then shift the Mode back from K → U "

Mode Shifting: is facilitated thru System Call → interrupt (s/w)

SVC: Supervisory Call

BSA: (Branch & Same address)

ISR: int Service routine

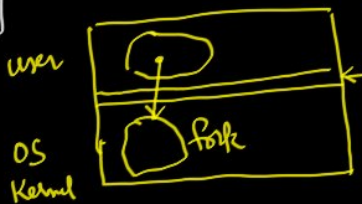


Memory: Static, Dynamic

```
main()
{
  int a,b,c;
  1. a=1;      Load R1, a
  2. b=2;
  3. c=a+b;
  4. f(c);  → BSA –;
  6. fork();  → SVC
}

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

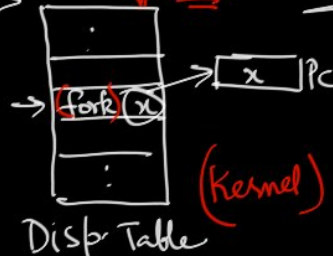
C.T 1 → SVC → 2 R.T → s/w Interrupt → ISR → (i) Mode Shifting ✓, (ii) Address

SVC → Trap-Instn / s/w Int. instn

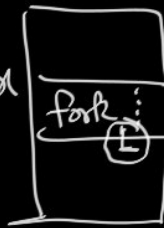


I.F | Opcode | Addr

Svc



RM



→ Defn O.S

→ Functions & Goals of O.S

→ Types
- U.Pr
- M.Pr
  - Pre
  - N.Pr.

→ Arch. Theory. For M.Pr O.S
- DMA
- Addr. Transl.
- Dual Mode opn
  - ↳ Mode Shifting thru Sys. Call.

Process Management | Subsystem:

1. Process Concepts: (*)

Program vs Process:

(.exe) → Instrns + Data

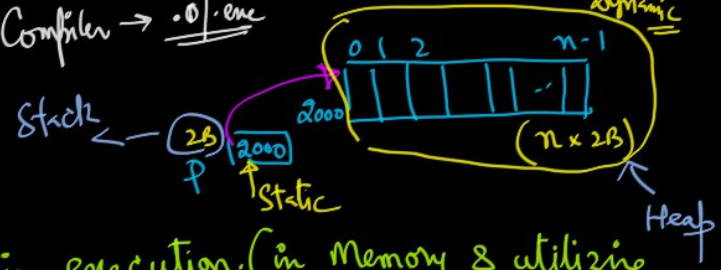
Ex: int a,b,c;

Instrn:
- a=1; Load
- b=2; Load
- c=a+b; Add, Store

Data:
- Static
  - → Fixed Size
  - → Allocation
    - : C.T x
    - : L.T: (Before R.T)
- Dynamic : Size is Variable
  - : alloc. @ R.T (Heap)

↳ Program in execution (in Memory & utilizing resources)

Compiler → .o/.exe




```
main( )  static  static
{
  int a, b, *p, n;
  int A[n];   Dynamic Array
  scanf("%d", &n);
  p=(int *) malloc(sizeof(int)*n);
}
```

Process

UNIX / LINUX

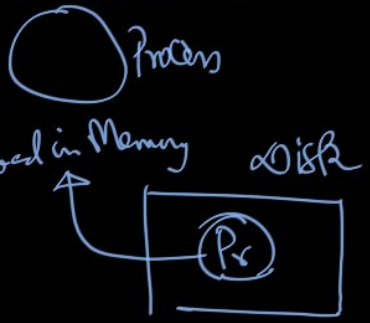
→ Program in execution
→ Instance of a Program
→ unit of cpu utilization
→ Active entity
→ Locus of control (of OS)
→ Animated Spirit
→ is in Memory

WIN / Task / TM

Program : class

→ is passive
→ is on Disk

UNIX / LINUX
ls - list
PS
Top : list of Processes



Prog vs Process
class vs object

How does O.S developer view Process:

→ as an A.D.T (Abstract data type)

<Defn ; Repr ; operations ; Attributes>

Impl.

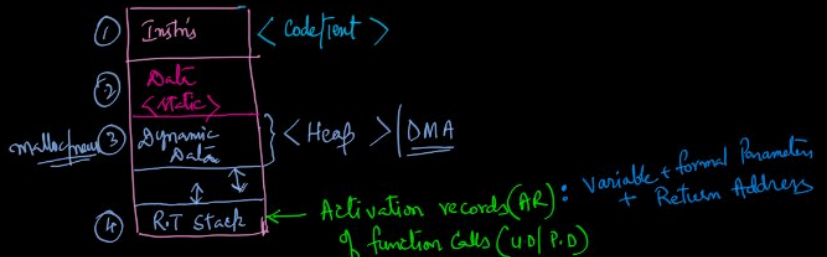
Impl. (i)

- Create: Res_alloc
- Schedule: onto cpu
- Block: for I/O
- Suspend: onto disk
- Terminate: Res_dealloc

⇓

Opn's are Implemented by O.S on behalf of users

Process structure:



Process Attributes (characteristics / Properties):

(i) Identification related: Pid; PPid; gid

(ii) cpu related: State; Type; Priority; PC; General Register Set

(iii) Mem related: Size (bytes); Limits

(iv) File related: list of open files + Standard files (Stdin; Stdout; Stderr)
Stdin → Keyboard; Stdout → Monitor; Stderr → Monitor

(v) Device related: list of open devices

(vi) Accounting: BT (Burst time); AT (Arrival_time)

Attribute values are kept in a Table known as P.C.B (Process Control Block / Process descriptor)

→ The ptr in PCB is used to create a list {R.Q, B.Q}

→ Each process during its lifetime is associated with its own PCB;

→ PCB is just like Aadhar card of the Process.

Content of PCB is Process Content (Environment)

Process control Block

pointer to Next PCB

Process

Content switching

Volume of info in PCB is known as: {CONTEXT}

Process States and State Transition Diagram :

→ <New ; Ready ; Running ; Block / Wait ; Terminate>

- New → Created → Res gets allocated
- Ready → to run
- Running → on cpu
- Block / Wait → for I/O / Sys-call
- Terminate → Res-Deallocation

Suspend-State & resume opn

→ As long as Process is in Ready, Runn, & Block States it is in Memory.

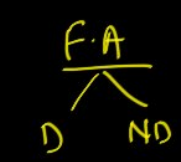
→ Many a times it become necessary for the O.S to The Process from memory onto Disk (Temporily)

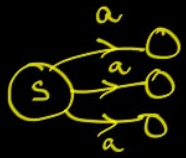
Swap-out / roll-out → Suspend

→ Process can get suspended from Ready, Running & Block States

Swap-in / roll-in → resume

* State-Transition Diagram :



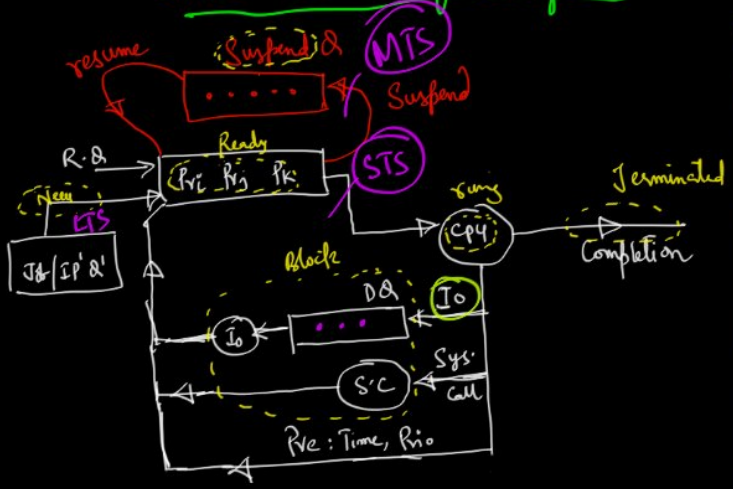


Note: Resource-Pre Emption of a Process in Ready State Causes the Process to get Block.

Scheduling Queues :

- In-Memory
  - Ready Q
    - List of PCB's of all Ready Process
    - RQ → Pri | Prj | Prk | ... → PCBi, PCBj, PCBk
  - Block Q (O.S maintains a separate 'Q' per device)
    - DQ1 → ... → IOD1
    - DQ2 → ... → IOD2
    - DQ3 → ... → IOD3
- on-Disk
  - Suspend 'Q' — List of all Suspended Process — Disk
  - Job-Q | IP-Q (New) — List of Programs ready to be Loaded in Mem — Disk

State - Queueing diagram :



LTS is supposed to control degree of M.Pr

(No. of Process in the System)

< OS - Machine Diagram >

Schedulers & dispatcher :

(i) Long Term Scheduler (LTS) : N → R

(ii) Short Term " (STS) : Rd → Run

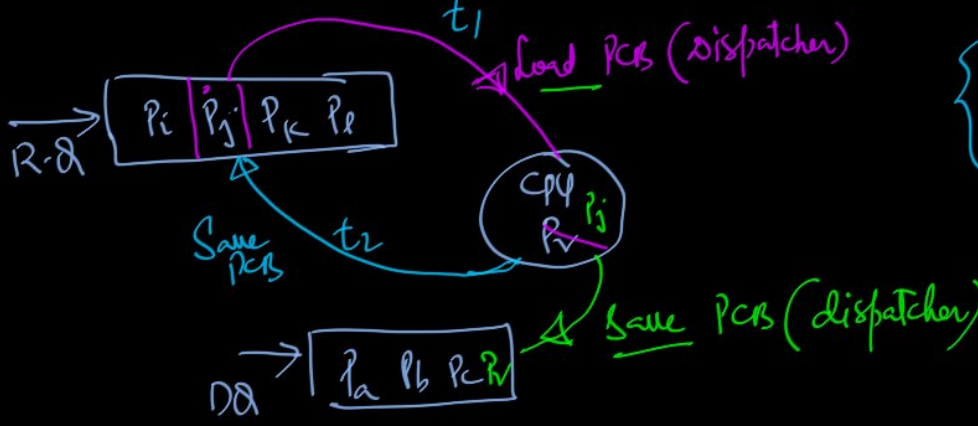
CPU scheduler

(iii) Medium Term " (MTS) : Susp & Resm

Swapper

Dispatcher : Carries out the activity of Context-Switching

Content-switching (CS): Act of saving & loading the PCBs of processes from & to RQ/BQ etc is content-switching that happens during a Process switch on CPU.



{ Content-switching time (dispatch Latency) }

↓

overhead

$(t_1 + t_2)$

The state of CPU during content-switching is Busy (with OS)

01. Consider a System having 'n' CPUs ($n \geq 1$) and 'k' Processes ($k > n$). Calculate lower bound and upper bound of the number of Processes that can be in the Ready, Running and Block states.

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

$n = 1$

$K = 3$

02. Consider the following statements about process state transitions for a system using preemptive scheduling

a) I. A running process can move to ready state ✓ : Pre

b) II. A ready process can move to running state ✓ : Sch

c) III. A blocked process can move to running state : X

d) IV. A blocked process can move to ready state. : ✓

Which of the above statements are TRUE?

(a) I, II and IV only (b) I, II, III and IV (a, b, d)

(c) I, II and III only (d) II and III only

MSQs

{Key-oriented Approach}

Binary

RQ → P1 P2 P3

CPU

User Process

1 CPU - 1 Core

```
While (1)
{
  switch (cond)
  {
    Case I : ...
    Case II : ...
  } Case III : ...
```

P1 P2 P3 BQ

OS is event driven

2) ⊛ CPU Scheduling / Process Scheduling [Design of S.T.S / CPU Scheduler]

(Function) It runs on R.Q to decide (using Criteria) which Ready process should run next onto CPU. Using the criteria it implements the CPU Scheduling Algo.

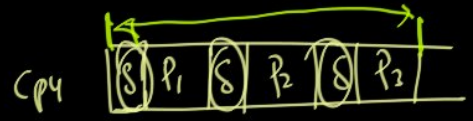
Goals:

→ Max. CPU utilization (Max of Thruput: No. of Processes completed Per unit time)

→ Min. waiting time (WT), Turn-Around-Time (TAT), Response-time (RT)

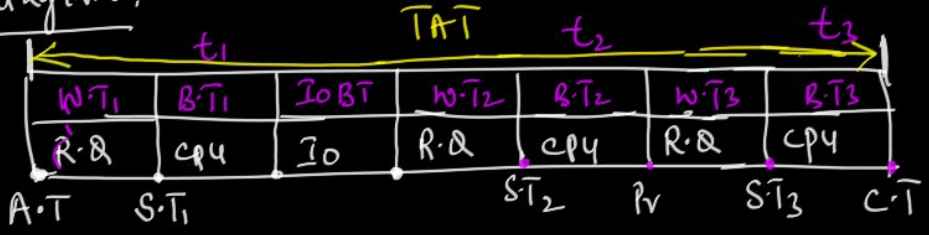
→ Fair (unbiased)

Process_times:



Process_timing_diagram:

Pi:



Total BT = $t_1 + t_2 + t_3$

→ Arrival_time (AT):

→ Waiting_times (W.T): ($WT_1 + WT_2 + WT_3$)

→ Scheduling_time (S.T)

→ Burst_time (B.T)

→ Io_Burst_time (IoBT)

→ Completion_time (C.T)

→ Turn_Around_Time (TAT): CT − AT — (1) ✓

→ Waiting_time = TAT − (B.T + IoBT) — (2) ✓

if IoBT = 0, W.T = TAT − BT

Frame-work:

n: no. of Processes
A_i: AT of Pr_i
x_i: BT of Pr_i
y_i: IoRT of Pr_i
c_i: CT of Pr_i

(i) $TAT(Pr_i) = (c_i - A_i)$; $Av.TAT = \frac{1}{n}\sum_{i=1}^{n}(c_i - A_i)$

(ii) $WT(Pr_i) = (c_i - A_i) - (x_i + y_i)$; $Av.WT = \frac{1}{n}\sum_{i=1}^{n}(c_i - A_i) - (x_i + y_i)$

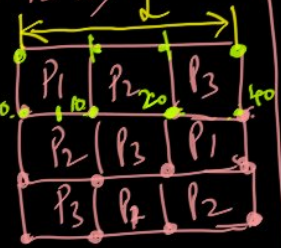
(iii) $\boxed{\text{Schedule\_length } (\mathcal{L}) = \text{Max}(c_i) - \text{Min}(A_i)}$

(iv) $\boxed{\text{Throughput } (\eta) = \frac{n}{\mathcal{L}}}$

$n - \mathcal{L}$
$? - 1$

Let s: denote context-switch time b/w Processes (Scheduling overhead)

$n = 3; \langle P_1\ P_2\ P_3 \rangle$



No. of Schedules $= \lfloor n$ (N.Pr)

No. of Schedules in Pr. Schedule $= \infty$

(v) **Response Time (R.T):**

(The time of admitting the request by the Process to the time @ which it starts producing results (response))

CPU Scheduling Techniques:

(i) First Come First Served (FCFS):

TAT = CT − AT
WT = TAT − BT

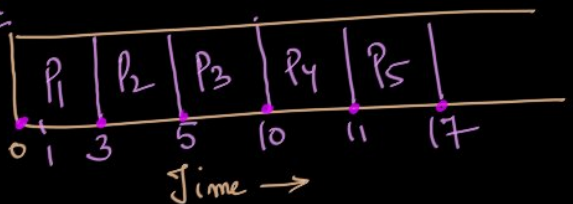
L = 17 ; Av. TAT = 40/5 = 8
Av. WT = 23/5 = 4.6

Sel. criteria: A.T
Mode of opn: Non-Pr
Conflict resolution: {Lower Pid}

Assumptions:
(i) Time is in clock-Ticks
(ii) Context-Switch ovhd is negligible ($\delta = 0$)
(iii) IOBTs of Processes is zero
(All processes are CPU-Bound)

P.No	A.T	B.T	C.T	TAT	W.T
1	0	3	3	3	0
2	0	2	5	5	3
3	0	5	10	10	5
4	1	1	11	10	9
5	5	6	17	12	6

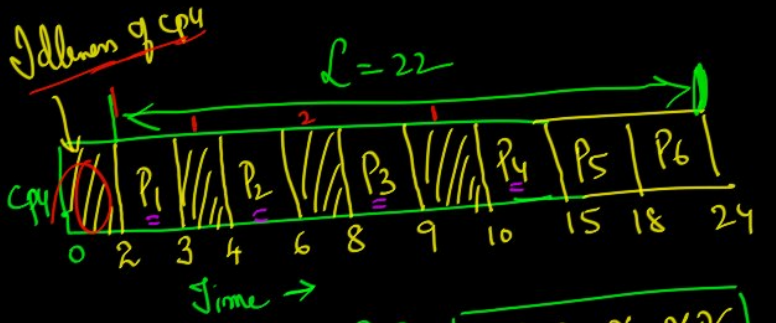
Gantt chart



R.Q: P1 P2 P3 P4 P5

2)

P.No	A.T	B.T	C.T	TAT	W.T
1	2	1			
2	4	2			
3	8	1			
4	10	5			
5	12	3			
6	15	6			



R.Q $P_1, P_2, P_3, P_4, P_5, P_6$

Av.TAT = 4

Av.WT = 1

$L = 24 - 2 = 22$

W.T $(P_1, P_2, P_3, P_4) = 0$

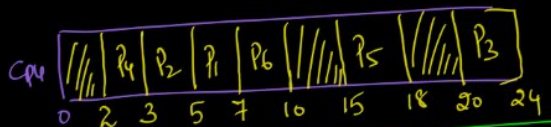
$L = 6/22$

% (CPU-Idleness / L) $= \frac{4}{22} = 18\%$

3)

P.No	A.T	B.T
1	5	2
2	3	2
3	20	4
4	2	1
5	15	3
6	5	3

↗ Jumbled A.T



$L_t = 24 - 2 = 22$ ✓

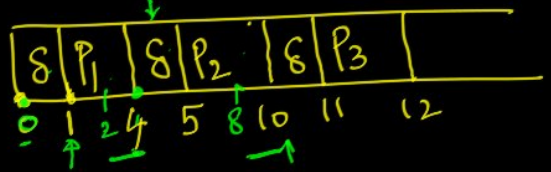
$TAT = CT - AT$

$WT = TAT - BT$ ✓

$WT = TAT - (BT + \delta)$ ✓

4) Assume CPU Scheduling ovhd is non negligible ; $\delta = 1$

P.No	AT	B.T	TAT	W.T &
1	0	3	4	1 0
2	2	5	8	3 ②
3	8	1	4	2 ✓



→ (*) FCFS with non-negligible CPU-Scheduling ovhd and non-zero IORTs

: Assume Concurrent IO : ✓

$\delta = 0$

Repeat with $\delta = 1$

P.No	A.T	(B.T ; IORT ; B.T)
1	0	(3 ; 10 ; 4)
2	1	(1 ; 2 ; 3)
3	4	(1 ; 4 ; 2)

RQ: ~~P1 P2 P3 P2 P3 P1~~

W.T = TAT − (BT + IORT)

$C = 17$

$\% \left(\frac{\text{CPU-Idleness}}{C}\right) = \frac{3}{17}$

DQ: ~~P2 P1~~

W.T(P1) = 0

W.T(P2) = 2 + 0 = 2 ✓

W.T(P3) = 0 + 0 ✓

Non-Conc. IO
(only one IO device)

TAT(P1) = 17 − 0 = 17 ✓

TAT(P2) = 9 − 1 = 8

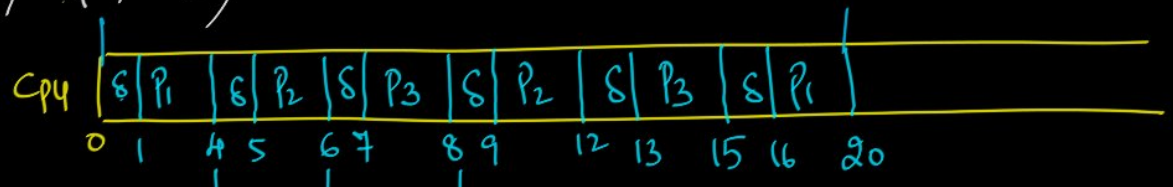
TAT(P3) = 11 − 4 = 7 ✓

P No	A.T	(BT ; IOBT ; BT)
1	0	(3 ; 10 ; 4)
2	1	(1 ; 2 ; 3)
3	4	(1 ; 4 ; 2)

S = 1

R.Q: P1 ; P2 ; P3 ; P2 ; P3 ; P1 ;

25/6/2021



TAT = (BT + IOBT + S)

11 = (4 + 2 + 2·S)

3 ✓

WT = TAT − (BT + IOBT + C·S) ✓

C = No. of CPU-Bursts

TAT (P2) = 12 − 1 = 11 ✓

W.T (P2) = 3 + 0 = 3 ✓ (From G.C)

% (CPU − ovhd / T) = 6 / 20 = 30%

% CPU-efficiency = 70%

03. 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 & 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 I/O before completion.

Assuming Concurrent I/O and negligible Scheduling Overhead. Calculate for FCFS Scheduling

(i) Average TAT of Processes

(ii) % CPU idleness

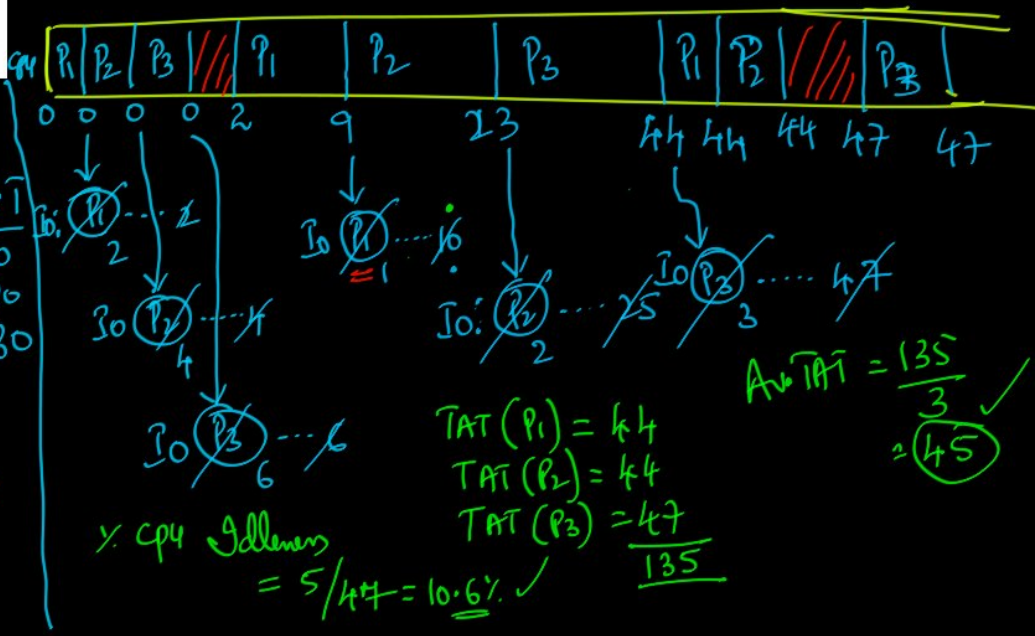
G

R.Q. $P_1; P_2; P_3; P_1 P_2 P_3 P_1 P_2 P_3$

$\delta = 0$

P.No	A.T	(IOBT; B.T; IOBT)	S.T
1	0	(2; 7; 1)	10
2	0	(4; 14; 2)	20
3	0	(6; 21; 3)	30

$\langle \overset{0}{BT}; \overline{IOBT}; \overline{BT}; \overline{IOBT}; \overset{0}{BT} \rangle$



TAT(P1) = 44

TAT(P2) = 44

TAT(P3) = 47

135

Avg TAT = 135/3 = 45

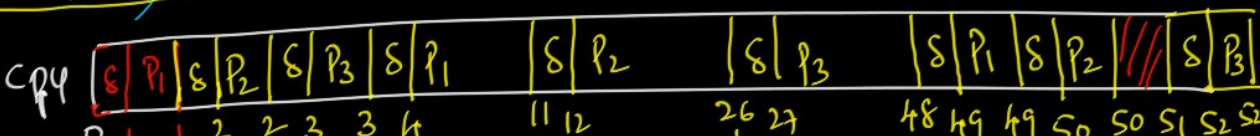
% CPU Idleness = 5/47 = 10.6%

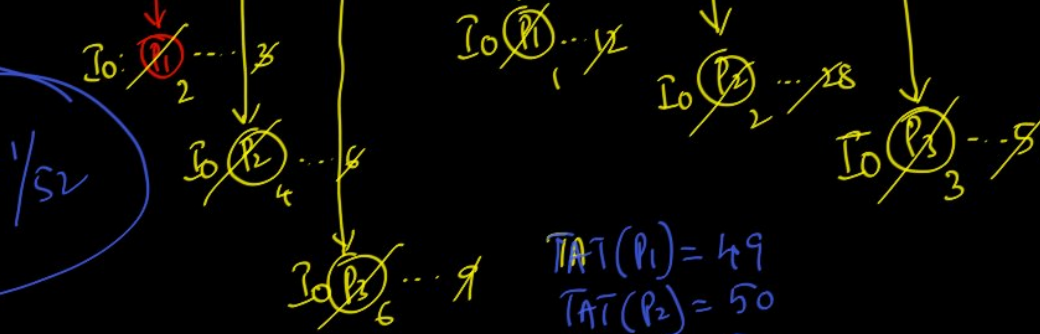
P.No	A.T	(IOBT; B.T; IOBT)	S.T
1	0	(2; 7; 1)	10
2	0	(4; 14; 2)	20
3	0	(6; 21; 3)	30

R.Q: ~~P1 P2 P3 P1 P2 P3 P1 P2~~ P3

Q: The data structure used for Impl. of R.Q in FCFS Scheduler is FIFO-Q;

$\delta = 1$





$\%\ CPU\ Idleness = \frac{1}{52}$

TAT(P1) = 49
TAT(P2) = 50
TAT(P3) = 52

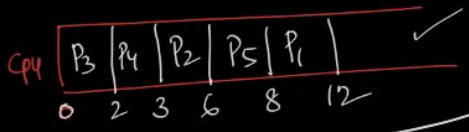
2) Shortest Job First (SJF) / Shortest Process Next (SPN)

Sel. criteria: Burst Time (B.T)
Mode of operation: Non-Pre
Conflict resolution: Lower Pid

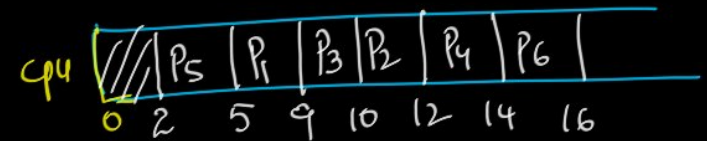
"Select the process with least B.T and Schedule the process onto cpu & complete"

R.Q : ~~P1 P2 P3 P4 P5~~

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



P.No	A.T	B.T
×1	5	4
×2	6	2
×3	7	1
×4	12	2
×5	2	3
•6	10	2



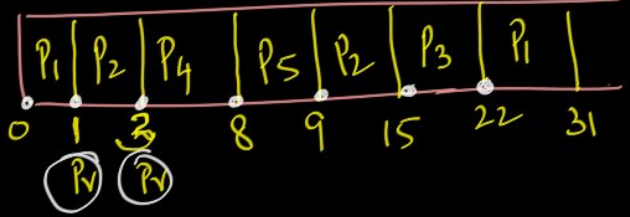
3) Shortest Remaining Time First (SRTF) | Pre Emptive SJF

Sel. criteria: BT
Mode of opn: Pre Emptive
Conflict Resl: Lower Pid

Note: Pre Emption of running Process is based on the arrival/availability of a Strictly Shorter Process

P.No	A.T	B.T
1	0	10
2	1	8
3	2	7
4	3	5
5	7	1

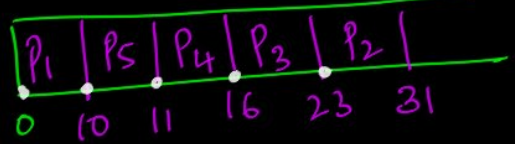
RQ | ~~P1~~; ~~P2~~ P1, P2 P3 P5



SRTF

Av.TAT = ~~12~~ 14.4 ✓
(δ = 0)

SJF:



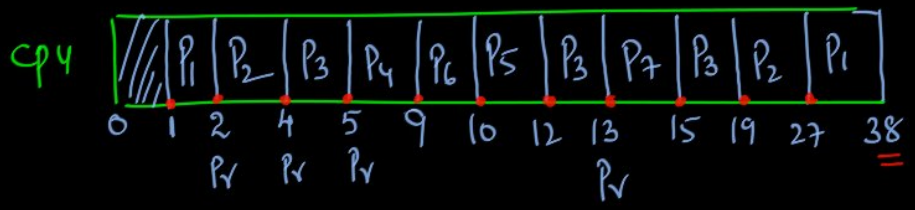
Av.TAT = ~~15.8~~ 15.6 ✓

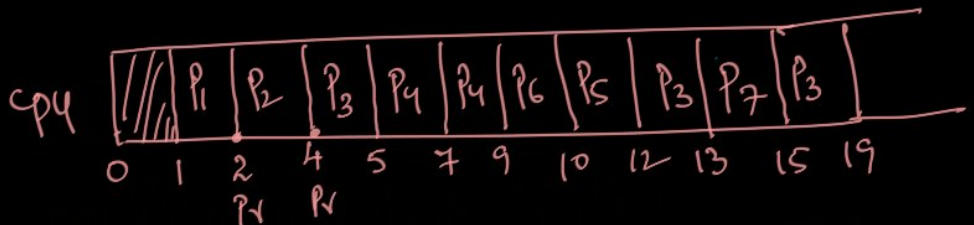
2)

P.No	A.T	B.T
1	1	~~12~~ 11
2	2	~~10~~ 8
3	4	~~6~~ ~~5~~ 4
X 4	5	~~4~~ 2
X 5	7	2
X 6	8	1
X 7	13	2

S.R.T.F :

Avg TAT = 12.85





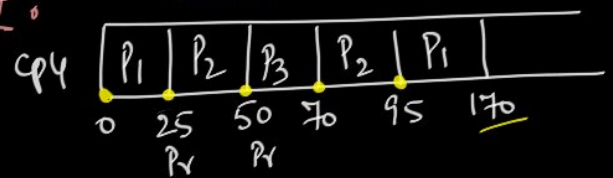
k = 37

δ = 0

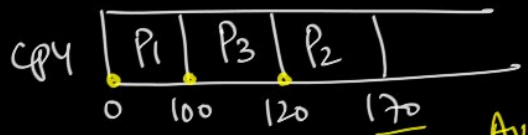
3)

P.No	A.T	B.T
1	0	100
2	25	50
3	50	20

S.R.T.F:



S.J.F:



δ = 0

Av.TAT = 105 ✓

Av.TAT ∝ C

Av WT ∝ C

Challenge Q:

For what values of δ, that avg. TAT of SRTF continues to be less than Avg. TAT of SJF;

$0 < \delta \leq x$

(or)

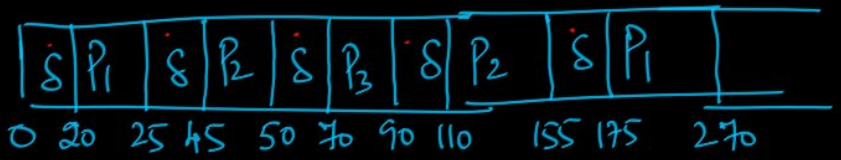
(For what range of values of δ, $0 < \delta \leq x$, that SRTF continues to perform better than SJF w.r.to Av.TAT)

P.No	A.T	B.T
1	0	~~100~~ 95
2	25	~~50~~ 45
3	50	20

$\delta = 20$;

SRTF

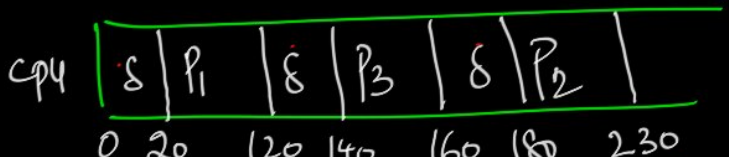
Av TAT = 146.6



$\delta = 10$

S.J.F

Av. TAT = 145



4) Longest Remaining Time First: (LRTF):

Criteria: B.T
Mode: PreEmptive

Tie breaking rule: "In Case of a tie b/w Processes w.r.to B.T then Favor the process having Lower Pid"

P.No	A.T	B.T
1	0	~~8 4 3 2 1~~
2	0	~~4 3 2 1~~
3	0	~~2 1~~

RQ: ~~8 4~~ 4 2 | P1; P2; P3

CPU:

P1	P2	P1	P2	P1	P2	P3	P1	P2	P3

0 5 6 7 8 9 10 11 12 13 14

Pr

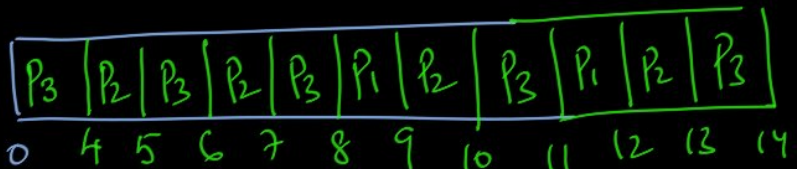
Av.TAT = $\frac{12+13+14}{3} = 13$

Av.TAT (LRTF): 13/11.33/12.66

2)

P.No	A.T	B.T
1	0	~~2~~ 1
2	0	~~4~~ ~~3~~ ~~2~~ 1
3	0	~~8~~ ~~4~~ ~~3~~ ~~2~~ 1

L.R.T.F :



Q) The Ready Queue is implemented using Priority & O.S, in SJF Scheduler;

(B.T)

Av.TAT = $\frac{12+13+14}{3}$

= 13 ✓

Performance of S.J.F:

: Always Favors Shorter Processes

Advantage

- → Since More Processes are completed in a duration of time, Hence it leads enhanced Throughput
- → Reduces Av. TAT & Av. WT

Optimal (Ideal) Algo.

Drawback

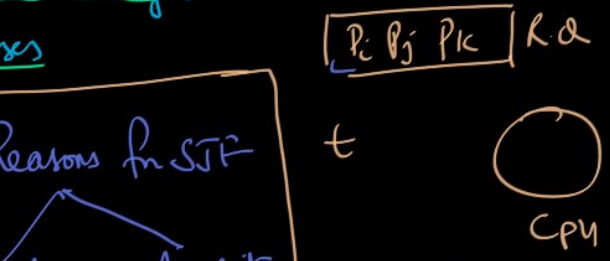
→ Starvation to Longer Processes

Reasons for SJF

- Benchmark to Compare it with other Algo.'s
- Impl it with Predicted BTs

S.J.F has a practical Limitation

It is Non-Impl.



Since BT's are not known apriori, Hence SJF is non-Impl.

Exponential Averaging Technique / Aging Algorithm

Let P_i : Process
t_i : Completed B.T
τ_i : Predicted B.T
τ_{n+1} : Next Pred. cpu B.T

(we completed 'n' B.Ts)

τ_1 = Initial condition

$0 \le \alpha \le 1$ $\quad$ τ_1 = given

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

$$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1} \quad - (2)$$

→ Recurrence

$$\tau_{n-1} = \alpha t_{n-2} + (1-\alpha)\tau_{n-2}$$

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

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

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

$\vdots$

τ_1

Factorial $n!$

$$f(n) = n \times f(n-1)$$
$$f(0) = 1$$

Q) Consider a System using expo. avg. Technique to Predicting next CPU BT of Pri; The previous Burst of Pri generated B.T's in the order $\underset{t_1}{4}, \underset{t_2}{8}, \underset{t_3}{16}, \underset{t_4}{10}$. Assuming $\tau_1 = 10$ & $\alpha = 0.5$; Predict Next CPU B.T of Pri;

$\tau_5 = 0.5(t_4) + 0.5(\tau_4)$

$= \frac{1}{2}(t_4 + \tau_4) = \frac{1}{2}(10 + \tau_4) = \frac{21.25}{2} = 10.625$ ✓

10.875 ✓

$\tau_4 = \frac{1}{2}(t_3 + \tau_3) = \frac{1}{2}(16 + \tau_3) = \frac{23.5}{2} = 11.75$

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

$\tau_2 = \frac{1}{2}(t_1 + \tau_1) = \frac{1}{2}(4 + 10) = 7$

OS Service

M.m

5) Highest Response Ratio Next (HRRN):

Sel. criteria: Response Ratio (RR) $= \left(\frac{w+s}{s}\right)$

Mode of opn: Non-Pre

$\Rightarrow$ Not only favors shorter Process but also limit the waiting time of longer Process.

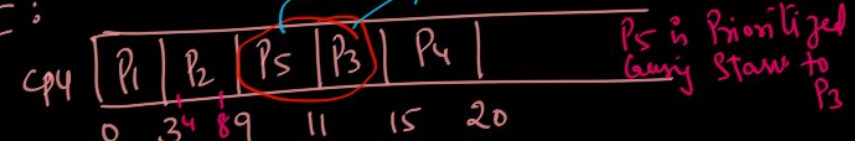
w = waiting time so far

s = Service time (B.T)

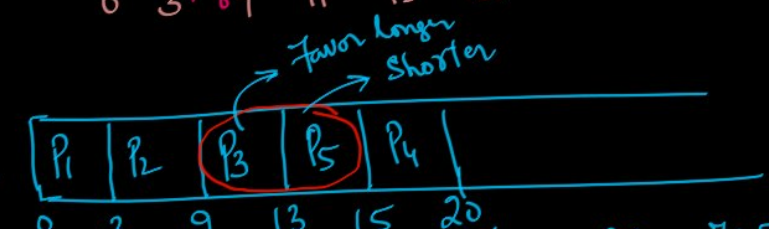
P.No	A.T	B.T
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

27/6/2021

S.J.F:



HRRN:



t9:

$RR_3 = \frac{5+4}{4} = 9/4$

$RR_4 = \frac{3+5}{5} = 8/5$

$RR_5 = 1+3/2 = 3/2$

t13:

$RR_4 = \frac{7+5}{5} = 12/5$

$RR_5 = \frac{5+2}{2} = 7/2$

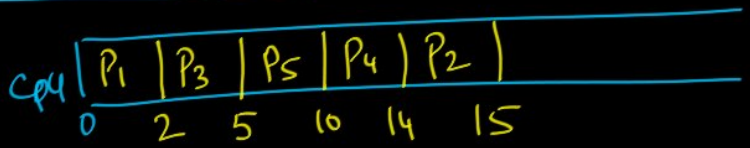
6) Priority based Scheduling:

Selection criteria: Priority
Mode of opn: N.Pr | Pr
Conflict Resoln: Lower P.id

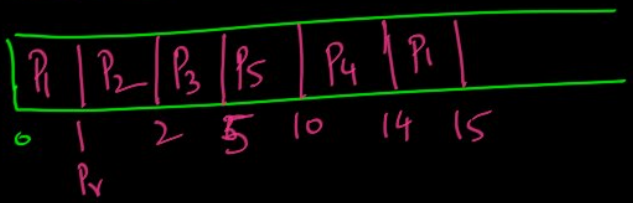
It works exactly like SJF. Except that it looks @ Priority instead of B.T.
<Process with higher Priority runs always>

Non-Preemptive Prio

Prio	P.No	A.T	B.T
4	1	0	2
6	2	1	1
8	3	2	3
7	4	3	4
12	5	5	5

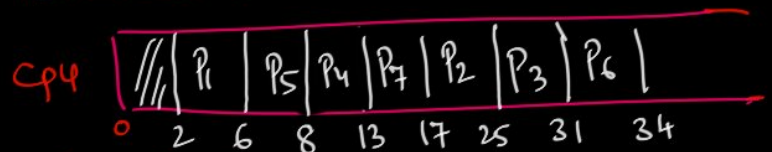


Preemptive Priority:

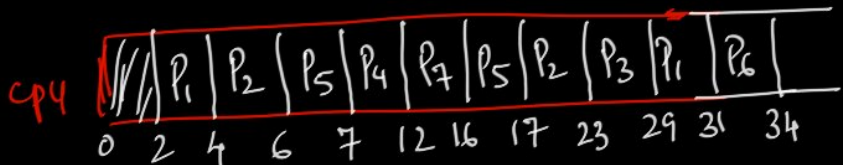


Prio	P.No	A.T	B.T
5	1	2	4
7	2	4	8
7	3	8	6
9	4	7	5
8	5	6	2
5	6	10	3
10	7	12	4

Non-Pr-Prio:



Pre-Priority:



Priority = f(Type, Size, Resources, ...) = integer

Static

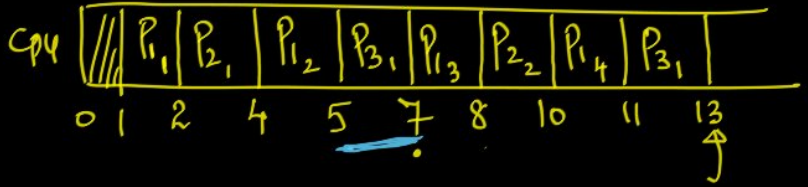
→ Lower Priority Process may get starved.

Dynamic ⇒ Aging Algorithms

→ Priority increases @ regular interval of times

04. Consider a System Preemptive Priority based Scheduling with 3 Processes P1, P2, P3 having infinite instances of them. The instances of these Processes arrive at regular intervals of 3, 7 & 20 ms respectively. The priority of the Process instances is the inverse of their periods. Each of the Process instance P1, P2, P3 consumes 1, 2 & 4 ms of CPU time respectively. The 1st instance of each Process is available at 1 ms. What is the Completion time of the 1st instance of Process P3?

R.Q: $P_1, P_2, P_3, P_1, P_1, P_2, P_1$



Prio	P.No	AT	BT	Period	Instances
1/3	1	1	1	3	→ <4, 7, 10, 13, ...
1/7	2	1	2	7	→ <8, 15, ...
1/20	3	1	4	20	→ <21, ...

6) Round Robin: (R.R) : < used for M.Pr time shared O.S >

Sel. criteria: AT + Time Quantum (TQ) / Slice

Mode of opn : Pre Emptive

"Pre Emption of running Process is based on completion of time quantum"

<Preemptive FCFS>

TQ = 2

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

R.Q: ~~P1~~ ; ~~P2~~ ; ~~P1~~ ; ~~P3~~ ; ~~P2~~ ; ~~P4~~ ; ~~P3~~ ; ~~P2~~ : FIFO

CPU

P1	P2	P1	P3	P2	P4	P3	P2

0 2 4 6 8 10 12 13 14

Av. TAT =

Av. W.T =

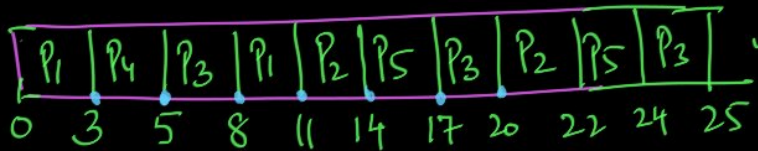
"New Processes & PreEmpted Process will be added @ the back of the Queue"

TQ = 3

P.No	A.T	B.T
1	0	6
2	4	5
3	2	7
4	1	2
5	8	5

R.Q: ~~P1~~; ~~P4~~; ~~P3~~; ~~P1~~; ~~P2~~; ~~P5~~; ~~P3~~; ~~P2~~; ~~P5~~; ~~P3~~

CPU:



TQ = 2

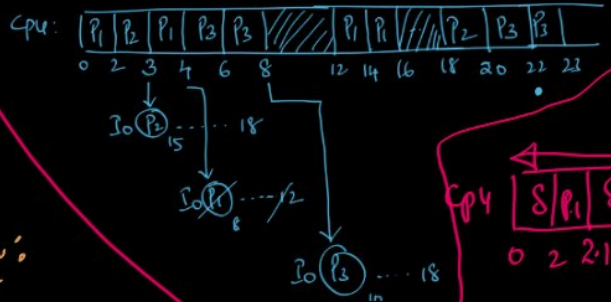
R.R
Av.TAT
Av.WT
Conc.To

P.No	A.T	<BT ; IOBT ; BT>
1	0	<3 ; 8 ; 4>
2	1	<1 ; 15 ; 2>
3	3	<4 ; 10 ; 3>

P.No	A.T	<BT ; IOBT ; BT>
1	0	<3 ; 8 ; 4>
2	1	<1 ; 15 ; 2>
3	3	<4 ; 10 ; 3>

TQ = 2

RQ: P1 P2 P1 P3 P3 P1 P1 P2 P3 P3

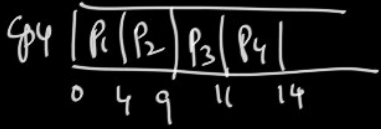


CPU U = 0.4 / 0.4 = 4/8.4 = 1/10 = 5%

δ = 2
TQ = 0.1

TQ = 10

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



Performance of Round Robin:

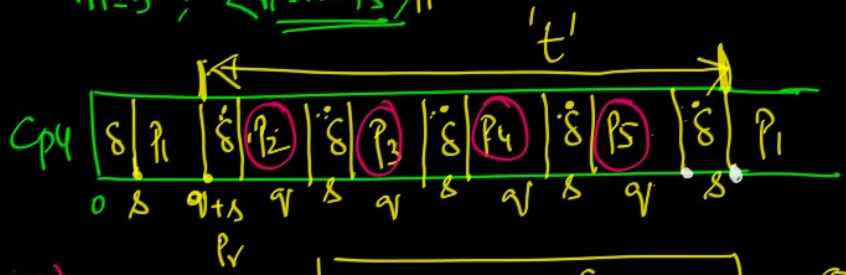
Value of TQ

- **Very Small** → CPU eff ~ 0%
- **Small** → More Context Switches (overhead) → Improve responsiveness (Interactiveness)
- **large** → less context switches → poor responsive
- **Very large:** → R.R degenerates to work like FCFS

06. Consider a System with 'n' Processes arriving at time 0^+ with substantially large Burst Times. The CPU scheduling overhead is 's' seconds, Time Quantum is 'q' seconds. Using Round Robin scheduling, what must be the value of Time Quantum 'q' such that each Process is guaranteed to get its turn at the CPU exactly after 't' seconds in its subsequent run on CPU.

$s = s$; $TQ = q = ?$

$n = 5$; $\langle P_1 \dots P_5 \rangle P_1$



6(a) $q \le \left(\frac{t - ns}{n-1}\right)$: atleast once within 't' (inclusive)

6(b) $q \ge \left(\frac{t - ns}{n-1}\right)$: atleast every 't' units, process gets a chance in subsequent run

$$t - ns = (n-1) \cdot q \quad -(1)$$

Total ovhd time

$$\therefore q \le \left(\frac{t - ns}{n-1}\right) \quad -(2)$$

exactly after 't'

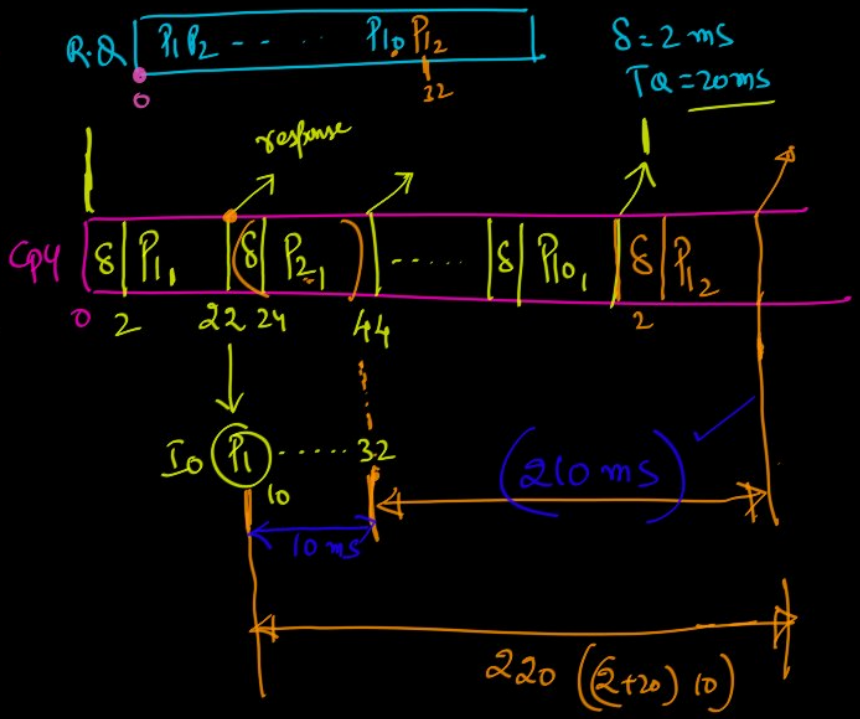
07. Consider a System using Round Robin Scheduling with 10 Processes all arriving at the time 0. Each Process is associated with 20 identical Request. Each Process request consumes 20 ms of CPU time after which it spends 10 ms of time on I/O, thereafter initiates subsequent Request. Assuming Scheduling Overhead of 2 ms and Time Quantum of 20 ms, Calculate

i. Response time of the 1st request of the 1st Process
ii. Response time of the 1st request of the last Process
iii. Response time of the subsequent request of any Process.

a) 22 ms ✓

b) $10 \cdot (2+20) = 220$ ms ✓

c) 210 ms ✓



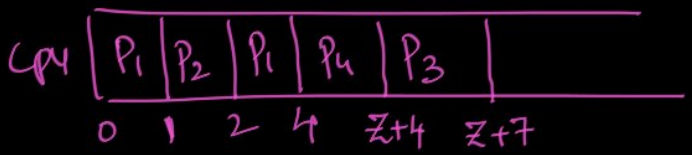
13. Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below:

Process	P1	P2	P3	P4
Arrival time	0	1	3	4
CPU burst time	3	1	3	Z

These processes are run on a single processor using preemptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is 2

SRTF: Av. w.t = 1ms

Case I: $Z < 3$



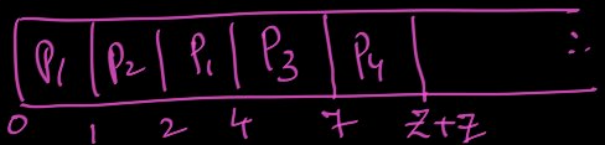
TAT = $Z+7-3 = (Z+4)$
$= Z+4-3$
$= Z+1$

Av.WT = $\frac{1+0+(Z+1)+0}{4} = 1$

$Z+2 = 4$

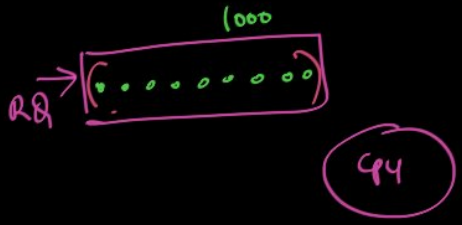
$\therefore Z = 2$

Case II: $Z > 3$



$\therefore \frac{1+0+1+3}{4} = \frac{5}{4} \times (1)$

7) Multi-Level Queue Scheduling :

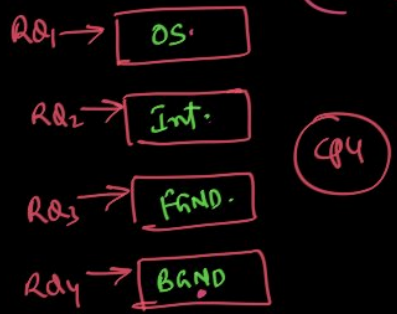


Single Ready Queue

→ Search Time / Filteration Time is needed.

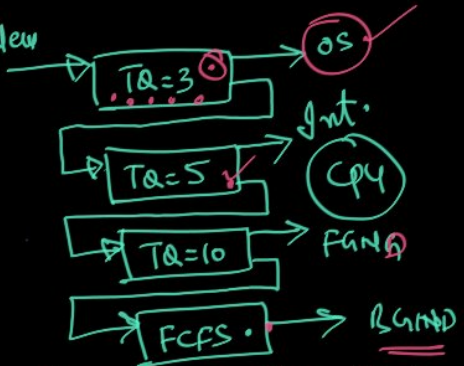
→ one Scheduling Technique

Multi-Level Q (MLQ)



* Diff. Scheduling Techniques
* Processes laying in Low-level Queues will be subjected to Suffer from Starvation

Multi-Level Feedback Q



More Inter. than MLQ

* Values of TQ are set in such a way that corresponding level Processes complete.

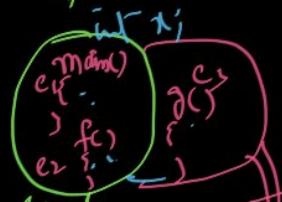
→

II. IPC & Synchronization : Process Coordination

IPC : Inter-Process-communication

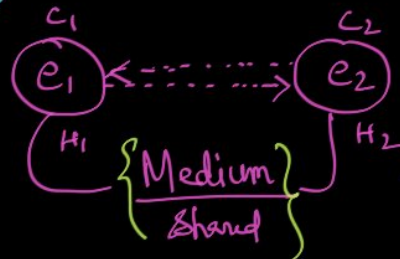
Inter

Intra



global variables
parameter-passing

IPC Mechanisms for Intra Process



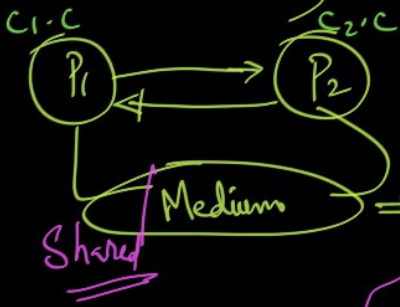
H/w

Cable + NIC + Electromagnetic

+

S/w : Protocols

TCP/IP



IPC Mechanisms

File, Pipe, Shared Memory, Message Q

Process coordination: Communicating Process must be Synchronized; otherwise it (IPC-Environment) will lead to problems.

Problems due to lack of Synchronization in IPC-Environment: (Need for Synch.)

a) Inconsistency (Incorrectness / wrong-results)

b) Data loss

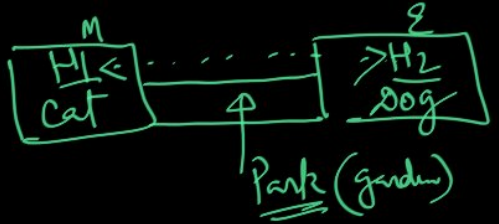
c) Deadlocks (lockup):
   - (Communication stops)
   - (Adamancy)

1) E-commerce
   - Amazon
   - 1-copy

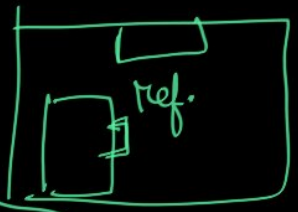
2) Reservation System:
   - SA
   - 1 Seat

3) Netbanking:
   - IOL
   - Joint-Acct
   - Acc-Balance

4) Neighbour :



5) More Milk Problem : PGs



Flat 3/4 PG

Synch. Mechanism / Protocol b/w Communicating Processes is needed

28/6/2021

Types of Synchronization:

Competition

Two or more processes are said to be in Competition, iff they contend/compete for the accessibility of a shared resource.

Ex:

: Inconsistency, Data loss

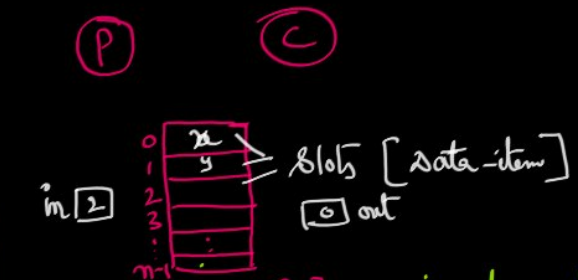
: CSMA/CD.

Cooperation

Two/more processes are said to be in Cooperative Synchronization, iff they get affected by each other.

Ex: Producer - Consumer Problem

Producer-Consumer:



: Deadlocks:

An appl. in an IPC environment may involve Competition or Cooperation or both.

Impl. of Prod. & Consumer :

```
#define N 100
int Buffer[N], Count=0;

Void Producer(void)
{
  int itemp, in=0;
  While(1)
  {
    a) itemp = Produce_item();
    b) While(Count==N);
    c) Buffer[in] = itemp;
    d) in = (in+1) % N;
    e) Count = Count+1;
  }
}

Void Consumer(void)
{
  int itemc, out=0;
  While(1)
  {
    a) While(Count==0);
    b) itemc = Buffer[out];
    c) out = (out+1) % N;
    d) Count = Count-1;
    e) Process_it(itemc);
  }
}
```

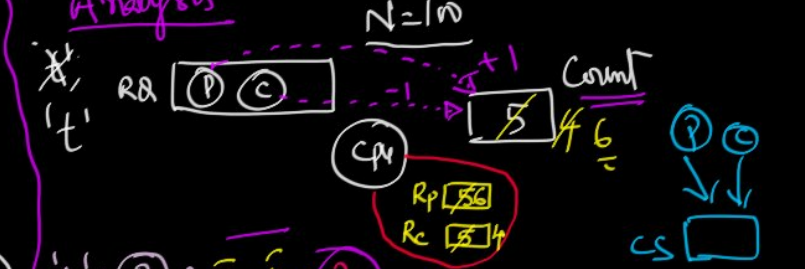
NCS → a)
CS → b)
CS → c)
NCS → d)
CS → e)

S/C
S/M

I. Load Rp, Count
II. Inc Rp
III. Store Count, Rp

I. Load Rc, Count
II. Dcr Rc
III. Store Count, Rc

Analysis



∴ Inconsistency

Consistency will happen iff processes doesn't get Pre Emption;

Necessary Conditions for Synchronization Problems in IPC Environment:

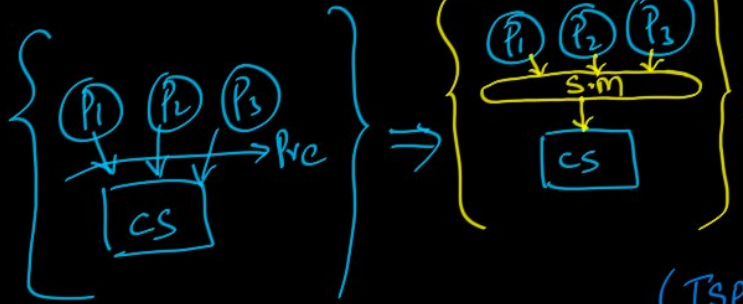
(i) **Critical Section (CS)**: is that portion of the program where shared resources are accessed.

Non-critical Section (NCS): is that part of the program that does not access shared resource.

other-region

(ii) **Race Conditions**: Processes must contend to access CS together @ the time (same) / concurrent access

(iii) **Pre Emption**:



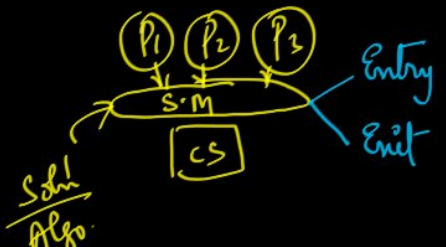
Problem Name:

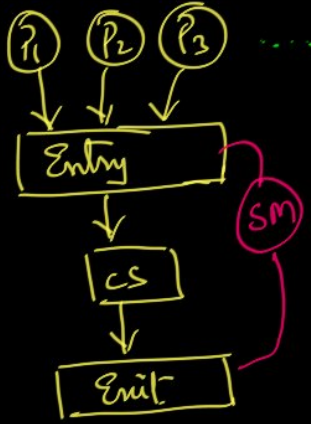
{CS-Problem}

Struct.

(TSP):

Solution / Algorithm of CS Problem is known as Synchronization Mechanism (SM) / Protocol





Model / General Architecture of S.M as a Soln for C.S Problem

```
void Process (int i)
{
  while (1)
  {
    Non_CS();
    entry
    CS
    exit
    Non_CS();
  }
}
```

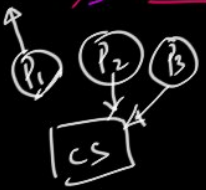
Bre

⊛ Requirements / Specifications / Conditions / Constraints of C.S Problem (Soln: S.M)

* P

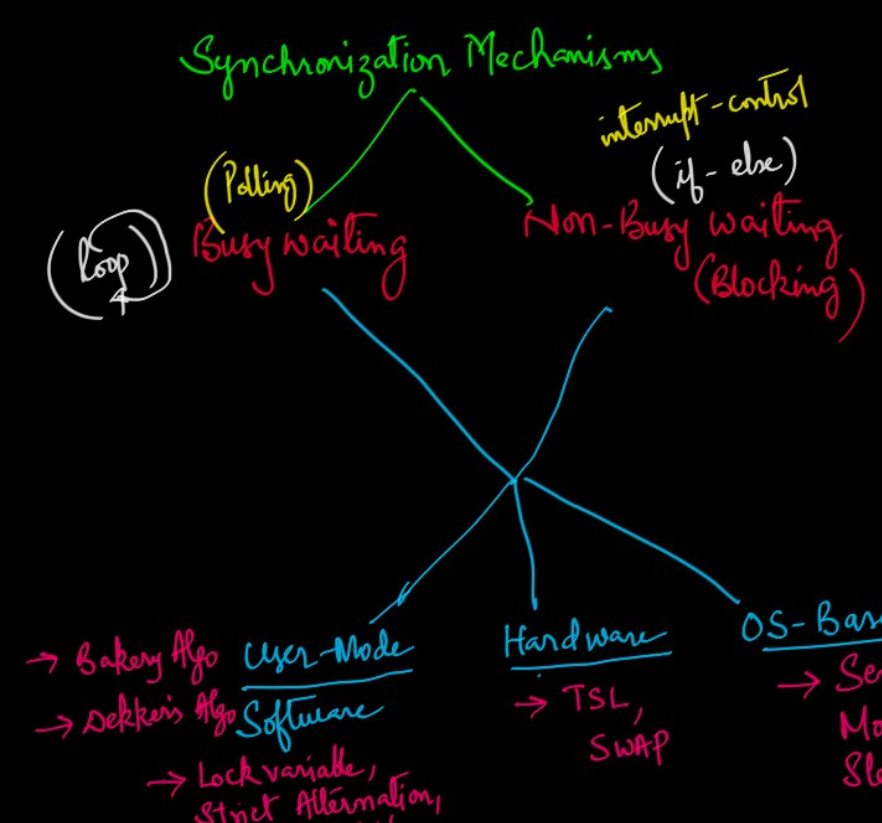
a) **Mutual Exclusion** : No two processes may be present in CS @ the same time

b) **Progress** : No Process running outside the CS (NMCS) should block / Prevent Participate (in decision making) the other interested Process(s) from entering CS. (unfair)



c) **Bounded Waiting** : No process has to wait for ever to access its CS. There must be bound / limit on the no. of times a process is allowed to enter CS before other Process requested is granted

d) **Architectural Neutral** : Independent of Arch.



Assumptions:

→ process spends finite time in c.s & comes out

→ c.s is flawless. (error-free)

→ CS is also piece of code. (instns)

→ processes can get preempted from cpu, while executing CS. (CS is still in use)

→ When a process is in entry section, implies it is intrsted in c.s

→ A process is said to leave the cs only when it completes exit section

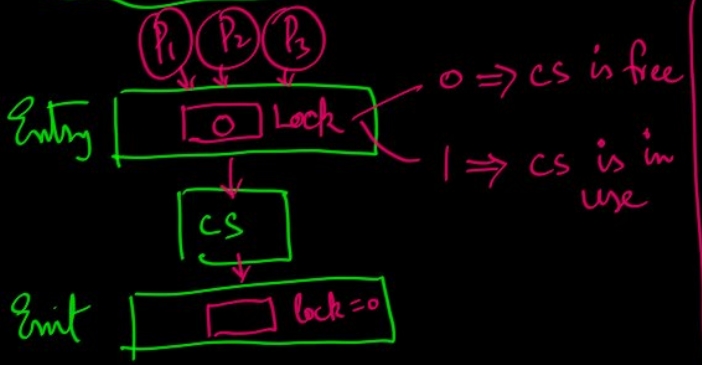
→ Process can as well get Pre-Empted while in Entry & Exit sec

1) Lock_Variable :

→ S/w Soln | Runnable @ u/m

→ Multi-process Soln

→ Busy-waiting



HLI

```
int lock=0;
void Process(int i)
{
  while(1)
  {
    a) Non_CS();
    b) while(lock!=0);
    c) lock=1;
    d) <CS>
    e) lock=0;
  }
}
```

a) M/E b) Progress ✓ ✗

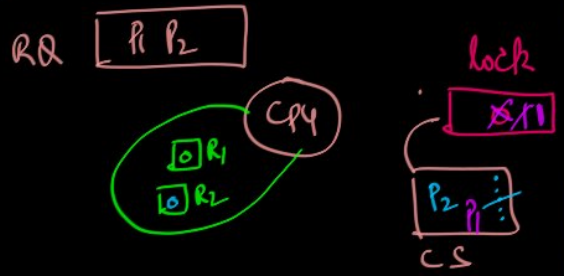
d) B/w:

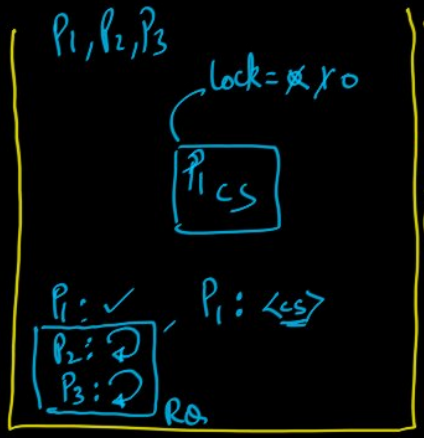
LLI

Process:

```
a) Non_CS:
b) Load Ri, Lock
c) Cmp Ri, #0
d) JNZ Step b
e) Store lock, #1
f) <CS>
g) Store lock, #0;
```

Analysis :





Process :

```
a) Non_CS :
while(1){
b) Load Ri, lock
c) Cmp Ri, #0
d) JNZ Step b
e) Store lock, #1
f) <CS>
g) Store lock, #0 ;
}
```

t_0: (P1) : a; b; c; d; Pr

t_5: (P2) : a; b; c; d; e; f; Pr

t_{10}: (P1) : e; f;

∴ M/E is violated.

∴ Bd/W is violated :

2) Strict Alternation:

- → Busy waiting
- → S/w Soln Impl. @ u/m
- → 2-process Solution $\langle P_0 | P_1 \rangle \mid \langle P_i | P_j \rangle$

Strictly on alternate Basis processes takes turn to enter CS

```
int turn = rand(0,1);

Void Process (0)
{
  While(1)
  {
    Non-CS();
    While(turn!=0);
    <CS>
    turn=1;
  }
}

Void Process (1)
{
  While(1)
  {
    Non-CS();
    While(turn!=1);
    <CS>
    turn=0;
  }
}
```

(i) Mut. Exclusion: ✓

(ii) Progress: ✗

turn 0

: P_0 : Non-CS();

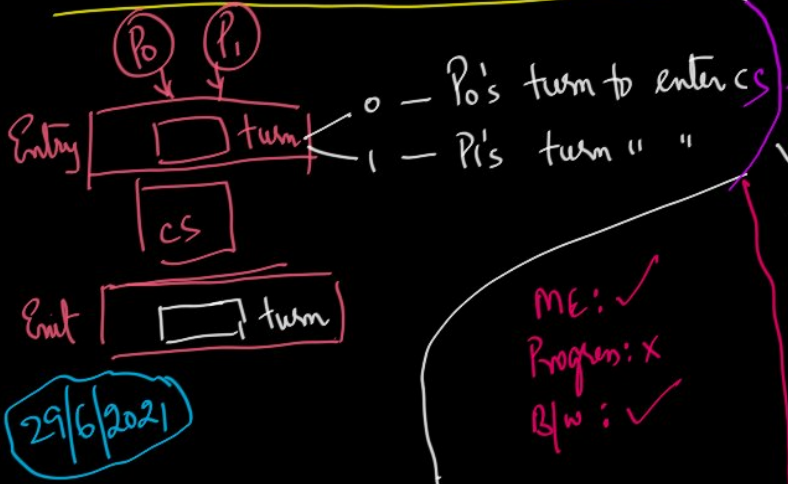
: P_1 : ✓

(iii) Bounded waiting: ✓

1 ✗ turn

P_0 : → CS → ···

P_1 : ↻



29/6/2021

ME: ✓
Progress: ✗
B/w: ✓

```
Void Process (int i)
{
  int j = NOT(i);
  While(1)
  {
    a) Non-CS();
    b) While (turn != i);
    c) <CS>   turn=j;
  } } d) turn=j;
```

Note: Strict Alt: is opposite to L:V in terms of Requirent Satisf.

3) Petersons Solution : → Combined approach of L·V + S·A :

- → Busy-waiting
- → S/w Soln Impl @ u/m
- → Two Process Soln (Pi|Pj)

```
#define N 2
#define TRUE 1
#define FALSE 0
int flag[N] = {FALSE}
int turn;
```

↳ Can take 'id'g Process.

M/E
Prog.
Bw

```
void Process(int i)
{
  int j = NOT(i);
  while(1)
  {
    a) Non_CS();
    b) flag[i] = T;
    c) turn = i;
    d) while(flag[j]=T && turn=i);
    e) <CS>
    f) flag[i]=F;
  }
}
```

Progress : always guaranteed

Bounded Waiting : Guarantees Bw always.

✓
- → 2-Processes
- → (B/wailing)
- → Mod. Architecture

RQ | P0 P1 |

CPU

flag[0] = F T
flag[1] = F T
turn = 0 1

(M.E is guaranteed always)

t_2 : P0 $i=0; j=1$: a; b; Prc

t_5 : P1 $i=1; j=0$: a; b; c; d

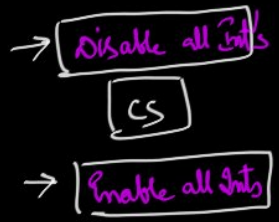
t_8 : P0 $i=0; j=1$: c; d

t_9 : P1 $i=1; j=0$; d : <CS>

II. Synchronization H/W:

a) Disable Interrupt:

→ Cause of Process is interrupt



→ Imp. OS functions have to get blocked.
→ Cannot be done in U/M
→ If multiple cpu's are available then Int's of all have to be disabled.

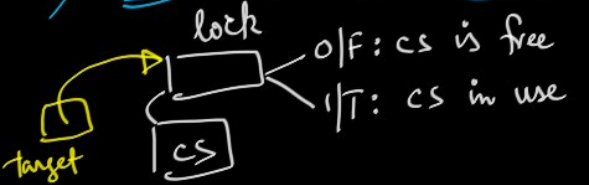
b) Special H/W Instructions:

Each processor supports special H/W Instructions that are atomic (without PreEmption)

(i) TSL (Test and Set Lock) } Multi process soln
(ii) Swap

→ H/W mechanisms
→ Busy waiting
→ Multiprocess Solution
→ Extension to Ideology of lock variable
<lock variable based>

1) TSL : Test and Set Lock :



Syntax : Bool TSL (&lock);

: {Return the current value of lock & Sets the value of lock to always True (1);

```
Bool TSL (Bool *target)
{
    Bool rv;
Atomic
    1. rv = *target;
    2. *target = TRUE;
    3. return (rv);
}
```

```
Bool Lock = F;
Void Process (int i)
{
    While (1)
    {
        a) Non_CS();
        b) While (TSL(&lock) == T);
        c) <CS>
        d) lock = F;
    }
}
```

Galvin
R.R

✓a) M/E will always be guaranteed.
✓b) Progress is guaranteed.
× c) This impl. does not guarantee Bound wait;

2) SWAP - Instruction:

: lock - key Mechanism

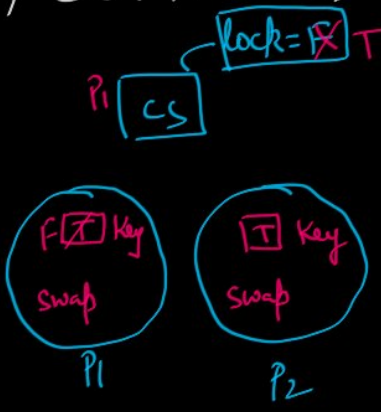
< CS is protected by lock; Each Process has its own key >

Atomic

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

→ guarantee ME always
→ Progress is guaranteed
→ Bound wait fails

```
Bool lock = F;
void Process(int i)
{
    Bool key;
    while(1)
    {
        a) Non_CS();
        b) key = T;
        c) while(key == T)
               SWAP(&key, &lock);
        d) <CS>
        e) lock = F;
    }
}
```



III. OS - Based :

a) Sleep() & Wakeup();

→ Blocking (Non Busy waiting)

→ Multi-process Soln

→ OS primitives

Impl. of (P) & (C) Problem using Sleep() & Wakeup();

```
void Producer (void)
{
  int itemp, in=0;
  while (T)
  {
    a) itemp = Produce_item();
    b) if (count==N) Sleep();
    c) Buffer[in] = itemp;
    d) in = (in+1) % N;
    e) count = count+1;
    f) if (count==1) Wakeup(Consumer);
  }
}
```

```
void Consumer (void)
{
  int itemc, out=0;
  while (T)
  {
    a) if (count==0) Sleep();   <- Pre
    b) itemc = Buffer[out];
    c) out = (out+1) % N
    d) count = count-1;
    e) if (count==N-1)
         Wakeup(Prod)
    f) Process_item(itemc);
  }
}
```

Count [ ] 2 1 0 -1 -2 -3

N=3

(i) Consumer: ...; Sleep() ↓ Pre

(ii) Producer:

Buffer[3]

a) Inconsistency

→ b) Deadlock

→ {SEMAPHORE}

SEMAPHORES:

- → Designed & Developed by E. Dijkstra
- → Is implemented in the OS Kernel as an A.D.T
- → Being an A.D.T, it support two operations, { DOWN ⟷ UP ; wait ⟷ signal / release ; P ⟷ V }
- → *Semaphore operations are atomic
- → Semaphore is an OS resource
- → It is a Synch. tool.
- → It is a General purpose utility
- → It is practically in use by many OS (UNIX / LINUX / WIN).
- → Its is a Blocking Construct / Multiprocess

30/6/2021

Defn: Semaphore is a variable (ADT: SEM) that takes on only Integral values.

→ integer Values

- COUNTING / GENERAL (CSEM) <-∞ to +∞>
- BINARY / MUTEXES (BSEM) <0 / 1>

1) Counting Semaphore (CSEM)

```
typedef
  struct
  {
    int value;        → State (init value)
    QueueType L;      → List of PCB's of those Processes that get blocked while performing 'DOWN' operation unsuccessfully
  } CSEM;
```

→ Positive value of SEM indicates that those many 'P' ops can be succ.

→ |SEM| when it is -ve indicates the no. of Blocked processes (unsuccessful Downs)

```
CSEM S;
S.value = 4;
S = 4;
:
DOWN(S);
<Si>
:
```

S = 4 3 2 1 0 -1

:1,1,1,1,

:1,

```
DOWN (CSEM S)
{
  1. S.value = S.value - 1;
  2. if (S.value < 0)
       put this process (PCB) in S.L() & Block (sleep());
     else
       return;
}
```

S = 1 0 -1 -2 -1 = 0 ; L: P2 P3

P1 P2 P3 P4

DOWN(S); → CS (P1) → UP(S); (P1)

P3

('Sem' can be used to solve the Problem of CS)

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

Q1) S = 5;

ops's: 1P; 3V; 8P; 7V; 12P; 4V

-10 -7 -15 -8 -20 -16

S = ?

-16 ✓

L P1 ... P16

Q2) S = x (Initial)

ops's: 10P; 6V; 8P; 2V; 4P; 1V

x-10+6-8+2-4+1 = -5

S = -5 (final)

Q3) 1000's clients

client requests

DOWN(s)

S = 50

LAN

MAN

Max-Load = 50

Data Base Server

resource (Disk)

UP(s) ✓

Semaphore Queues:

FIFO LIFO Priority

Strong other

Weak

Starvation

S = 1

P(s);

CS

V(s);

P1; P2 P3

P1

P2 P3

L

RQ

CSEM S=1;

Void Process(int i)

{ while(1)

{ NonCS;

P(s);

<cs>;

V(s);

} }

Pq Pk Pi, Pj

Priority FIFO

(i) M/E: ✓

(ii) Progress: ✓

(iii) Bo/Wait: ✓

(for 2 Processes)

2) Binary Semaphores (BSEM):

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

```
BSEM  S=1;
  :
  DOWN (S);
  <Si>
  :
```

```
DOWN(BSEM S)        OS
{
  if(S.value = 1)
  {
    S.value = 0;
    return;
  }
  else
  { put this process in
    S.L() & block it
  }
}
```

S=1 0

P1 P2 P3 P4 P5 P6

P1: DOWN(S);

P2 CS P5

P3: UP(S); P6

```
UP(BSEM S)   OS
{
  if (S.L() is NOT Empty)
  {
    Select a process from S.L()
    & wakeup();
  }
  else
    S.value = 1;
}
```

```
P(S)
Count --
V(S)
```
Count P

```
P(S)
Count +
V(S)
```
Count V

Cases of Binary Semaphores:

1) S=1;
   P(S);
   S = 0
   Status = Success

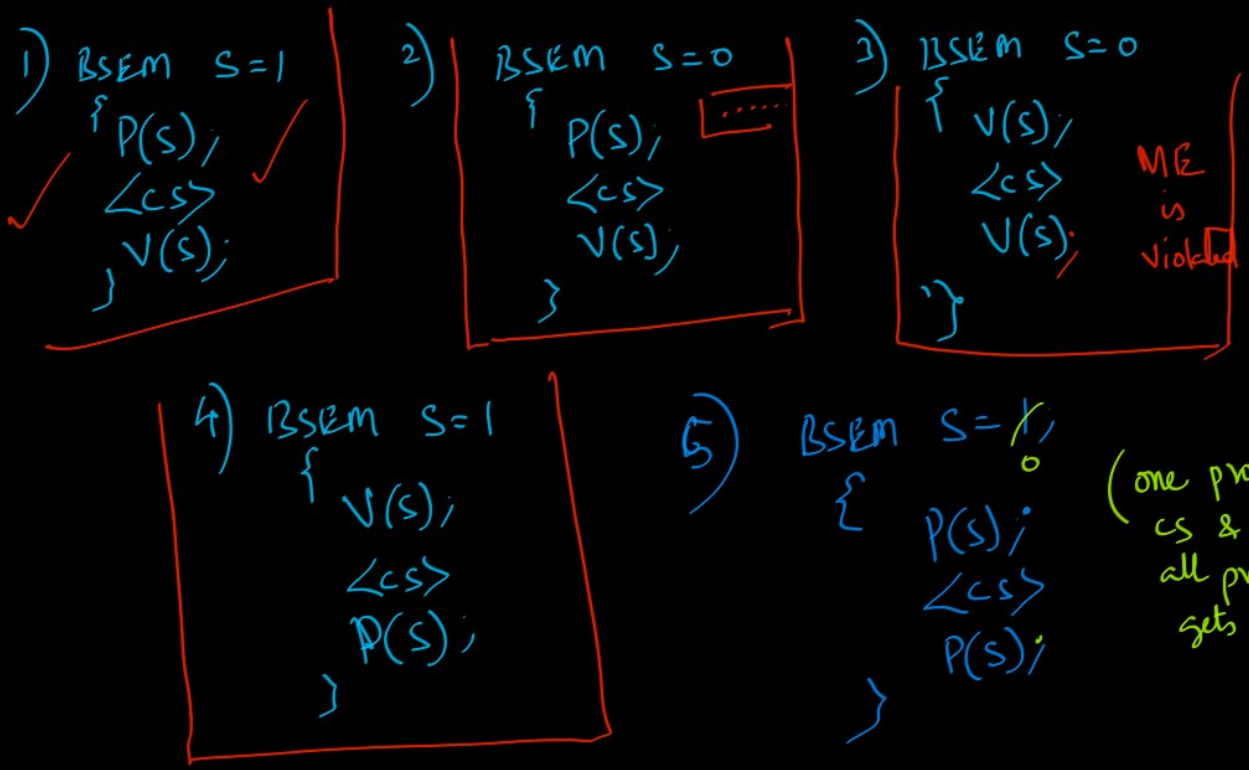
2) S=0;
   P(S);
   S=0
   Status = unsuccess

3) S=0
   V(S);
   S = 0 : 'Q' is NOT Empty
   S = 1 : 'Q' is Empty

4) S=1; ⇒ Q is Empty
   V(S);
   S=1
   Status = Success

5) S=0; (initial)
   V(S); (first opn)
   S=1
   Status = Success

Note: Process never gets blocked while Performing 'up' opn


1) BSEM S=1
{
P(S);
<CS>
V(S);
}
2) BSEM S=0
{
P(S);
<CS>
V(S);
}
3) BSEM S=0
{
V(S);
<CS>
V(S);
}
ME is violated
4) BSEM S=1
{
V(S);
<CS>
P(S);
}
5) BSEM S=1/0;
{
P(S);
<CS>
P(S);
}
(one process enters CS & thereafter all processes gets Blocked)

BSEM mutex = 1;

Pri i = 1,9

```
{
  while (1)
  {
    P(mutex);
    <cs>
    V(mutex);
  }
}
```

Prj j = 10

```
{ while (1) {
    V(mutex);
    <cs>
    V(mutex);
  }
}
```

Exit

M/E : L

$P_2 P_3 P_4 \cdots P_9$

P_1 P_2 P_8 P_3
CS

Max (n) = 3

Max (n) = 10

1/07/2021

Applications of Semaphores:

1. Implementation of classical IPC Problems:

a) Producer - Consumer:

```
#define N 100
int Buffer[N];
CSEM Empty = N;
```
<No. of Empty Slots in Buffer>

```
CSEM full = 0;
```
<No. of full Slots / Data-items>

```
BSEM mutex = 1;
```
<used for Mutual exclusion>

```
void Producer(void)
{
    int itemp, in = 0;
    while(1)
    {
        a) itemp = Produce_item();
        b) DOWN(Empty);
        c) DOWN(mutex);
CS      d) Buffer[in] = itemp;
        e) in = (in + 1) % N;
        f) UP(mutex);
        g) UP(full);
    }
}
```

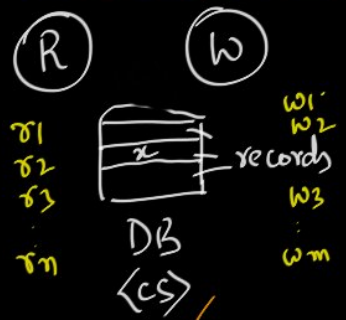
```
void Consumer(void)
{
    int itemc, out = 0;
    while(1)
    {
        a) DOWN(full);
        b) DOWN(mutex);
CS      c) itemc = Buffer[out];
        d) out = (out + 1) % N;
        e) UP(mutex);
        f) UP(Empty);
        g) Process_item(itemc);
    }
}
```

full = 0; Empty = N; mutex = 1; 0

Consumer: Blocked ✓

Producer: D(mutex); Blocked ✓ Deadlock

2) Readers-writers Problem:



a) First R-W: <Starvation to writers>

t_0: r_1 / w_1 : allow them

t_0: r_1 and w_1 : allow any one of them ✓

t_1: r_1 : ✓ r_1 in CS

t_2: r_2 : ✓

t_3: r_3 : ✓

(H/W): challenge question

b) First W-R | Second R-W: Prioritizing writer / Starvation to readers

t_0: r_1 / w_1 : allow

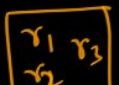
t_1: r_1 and w_1 : always allow writer

t_2: w_1 : → i t_3: ~~r_3~~

t_0: r_1

t_1: r_2

t_2: r_3



t_3: w_1

t_4: r_4

: "principle of turnstile"

"waiting for DB to become vacant so as to allow waiting writers to enter DB"

Implementation of First R-W using Semaphores :

```
BSEM db=1;
<indicates the availability of Database>
BSEM mutex=1;
<used by Rs to update "rc" as cs>
int rc = 0;
<Readers Count>
```

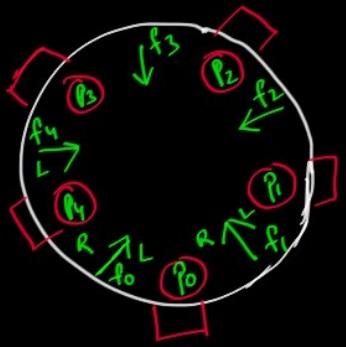

```
void writer(void)
{
  while(1)
  {
    DOWN(db);
    <write_DB>
    UP(db);
  }
}
```

```
void Reader(void)
{
  while(1)
  {
    DOWN(mutex);
    rc=rc+1;
    if(rc==1) DOWN(db);
    UP(mutex);
    <READ_DB>
    DOWN(mutex);
    rc=rc-1;
    if(rc==0) UP(db);
    UP(mutex);
  }
}
```

3) Dining Philosophers Problem

→ There are 'N' Philosophers ($N \geq 2$)

$N = 5$



CS is fork

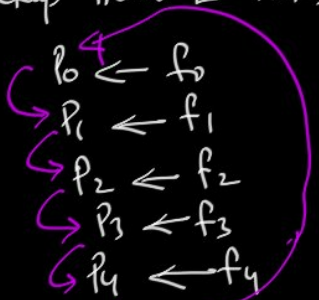
↑ Semaphore

```
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);
    }
}
```

Problem: Deadlock :

(i) All Pi's become hungry

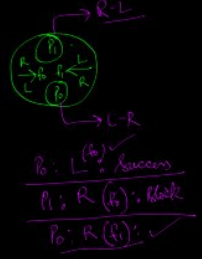
(ii) All pickup their 'L' forks



With no deadlock, max # of Pi's who can eat is (2)

How to Prevent Deadlock in D.P problem

- Semaphore based Solution
  - [Assignment] Code is in (Workbook)
- Non-Semaphore based Soln
  - { Let (N-1) P's follow L-R & 1 Pi, follow R-L }
  - { Let odd Numbd Pi's Pickup L-R & even numbd Pickup R-L }



(i) Sleeping-Barber Problem
<Barber Shop Simulation>

(ii) Cigarette-Smokers

Concurrency conditions :

Let S_i & S_j be two statements ;

Read-Set (S) : R(S)
Write-Set (S) : W(S)

S_i : $a = b + c$;
S_j : $d = b * c$;

S_i — $R(S_i)$, $W(S_i)$
S_j — $R(S_j)$, $W(S_j)$

I. $R(S_i) \cap W(S_j) = \emptyset$
II. $R(S_j) \cap W(S_i) = \emptyset$
III. $W(S_i) \cap W(S_j) = \emptyset$

IV. $R(S_i) \cap R(S_j) =$ May / May Not be $\emptyset$

" Bernsteins Concurrency conditions "

{R-W Problem}

a += ++b * --c ;

R = {a, b, c}
W = {a, b, c}

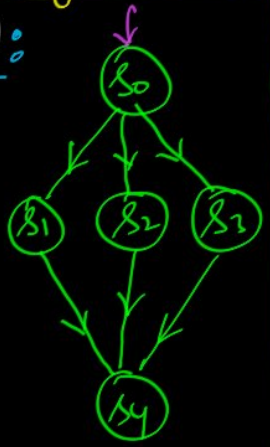
How to achieve Concurrency : Concurrency Mechanisms :

(i) Parbegin – Parend / Cobegin – Coend :

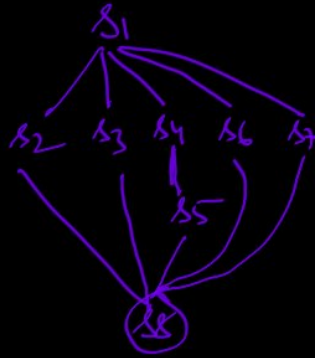
begin
S1;
S2;
S3;
End

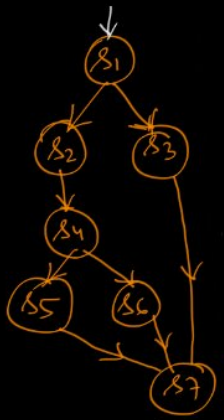
{
S1;
S2;
S3;
}

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



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






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

This graph is non Impl.

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

Assig. less than (7)

Using Semaphores to meet Conc. requirements

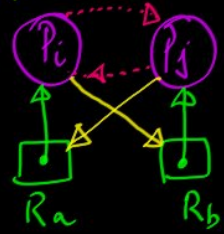
BSEM a, b, c, d, e, f, g = {0}

```
Parbegin
    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
Parend
```

Deadlocks:

⇒ Two/more processes are said to be in Deadlock, iff they wait for the happening of an event, which would never happen.

Infinite Blocking
Blocking forever

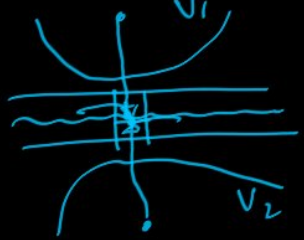


2/7/2021

Pi
P(S);
P(T); Block

Pj
P(T);
P(S); Block

⇒ Deadlock



Deadlock is the most undesirable feature of O.S

(i) it leads to Blocking of Processes
(ii) underutilization of cpu/devices

→ Hold resources & gets Blocked.

FB

Deadlock vs Starvation

- Starvation → Indefinite waiting (long waiting)
- Deadlock → infinite waiting

Livelock: is also deadlock:

Spinlock: Busy-waiting

```
while (~~~);
```

Note: In deadlock the states of the processes are Block. whereas in livelock the states of the Processes are not Block. (May be Ready / Running)

Priority Inversion Problem

: pre-prio based Scheduling
: Busy-wait Mech's

Characterization :

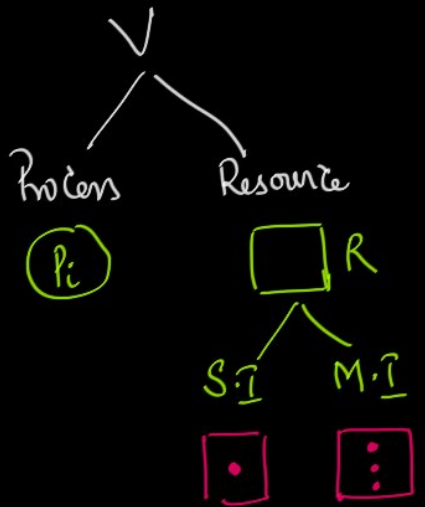
a) **Necessary Conditions :**

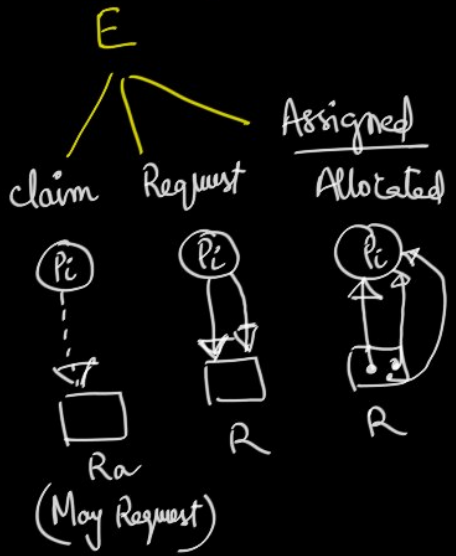
1) Mutual Exclusion :
2) Hold and Wait :
3) No Preemption of resources
4) Circular wait :

b) **Resource Allocation Graph (R.A.G)**

$G = (V, E)$

(Multi Graph)





Methods for Handling Deadlock : Deadlock Handling Strategies :

1) Deadlock Prevention
2) Deadlock Avoidance (Banker's Algorithm) — T_1 :. deadlock never happens
3) Deadlock Detection & Recovery (Doctor's Algo)
4) Deadlock Ignorance (Ostrich Algorithm) — T_2 :. deadlock occurs

---

4) Deadlock Ignorance (Ostrich Algorithm)

" No Strategy by O.S "

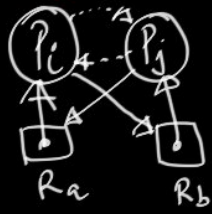
Risk ↓

↳ user/Programmer → { Power up / Restart }

1) Deadlock Prevention: "Dissatisfying one/more of the necessary conditions"

a) Mutual Exclusion: RQ [P1 P2 P3 ...] ← CS (CS4) ∴ M/E is non dissatisfiable.

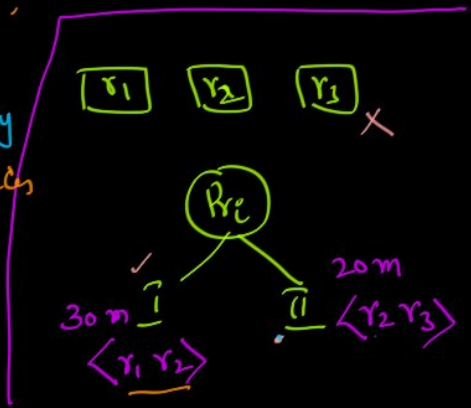
b) !(Hold and wait): Hold or Wait



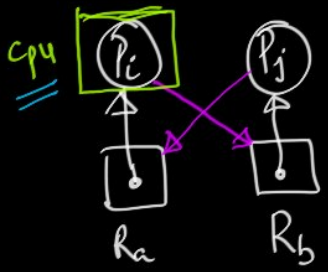
✓ 1) Process must Request & be allocated all resources Priori to its commencement.
- Starvation
- Resource inefficiency

✓ 2) Process must release all resources b/f making a fresh/new request.
- Starvation

{Gallium} ✓



c)! (No- Pre Emption):



Pre Emption of resources

forceful

(Running Process should never get Blocked)

(Greedy) (Selfish)

Self (voluntary)

gracious / Selfless

(Host)

Starvation:

"If a process is repeatedly getting pre Empted of its resources then such process will starve"

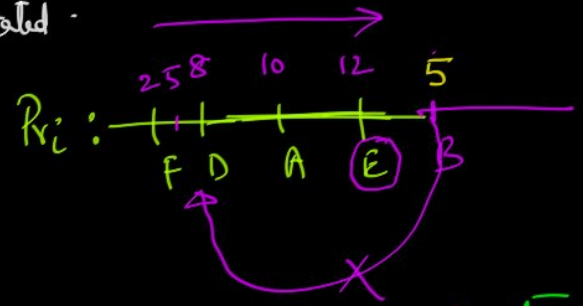
4) Circular wait : Implementing a total ordered relation among process for requesting resources.

Inc-order

a) Number all resources uniquely
b) Never allow a process to request a Lower numbered resource than the last one requested and allocated.

Ex:

	Resid
A	10
B	5
C	15
D	8
E	12
F	2
G	16



Soln: Release all Conflicting resources & make fresh request in order

⇒ (Starvation) ✓

II: Deadlock Avoidance:

↳ Concept of System State

Mode
- Safe: No-Deadlock
- unsafe: likely to go into deadlock (warning)

"The Basic objective of Avoidance Algorithms is to operate the System always in "SAFE" State"

Avoidance Algorithms / Approaches

I. (Resource-Allocation Graph Algo.)
Single Instance Resource

II. (Banker's Algorithm) (Multi-Instance Resource)
- Safety
- Resource-Request

1. Resource-Alloc. Graph Avoidance Algorithm: for S.I Resource

→ claim Edge (A Priori info.) ⇒ request Edge ⇒ Assigned edge.

⇒ Explained & understood ⇐

b) ⊛ Banker's Algorithm: (Multi-Instance Resource)

→ It is based on the ideology that process must demand a priori its maximum requirement of each Resource "

Data-Structures (System-State)

1) 'n' : no. of Process ($P_1 .. P_n$)

2) 'm' : no. of resources ($R_1 .. R_m$)

3) Maximum $[1..n, 1..m]$; $n \times m$

Max $[i, j] = K$

$P_i \xrightarrow[max]{} K(R_j)$

4) Allocation $[1..n, 1..m]$:

Alloc $[i, j] = a$

$P_i \xleftarrow[allo]{} a(R_j)$ $[a \leq K]$

5) Need $[1..n, 1..m]$: Max - Alloc

Need $[i, j] = b$

$P_i \xrightarrow[Need]{} b(R_j)$

6) Request $[1..n, 1..m]$:

Req $[i, j] = c$

$P_i \xrightarrow[req]{} c(R_j)$ @ 't' $[c \leq b]$

7) Total $[1..m]$: Total $[j] = z$

8) Available $[1..m]$: Avail $[j] = l$

There are $l(R_j)$ free avail @ 't'

Eg 1:

Total
29

$n = 5;\ m = 1;\ R = 29$

$$\text{Avail}(@t) = \text{Total} - \sum_{i=1}^{n} \text{Alloc}(P_i)$$

to:

	Max R	Alloc R	Need R	Avail R
P_1	10	5	5	3
P_2	15	8	7	~~5~~
P_3	5	2	3	~~10~~
P_4	8	5	3	~~16~~
P_5	12	6	6	21
				29

to: $\langle P_3; P_4; P_5; P_1; P_2 \rangle$

Safe Sequences — MCQ / MSQ

a)
b)
c)
d)

State { SAFE, UNSAFE } — Safety Algorithm

System is said to be Safe, iff the need of all Processes can be satisfied with avail resources in some order/seq.

• Example of Banker's Algorithm

- 5 processes P_0 through P_4;
  3 resource types:
  A (10 instances), B (5instances), and C (7 instances).
- Snapshot at time T_0:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	2 3 0
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	2 1 0
P_4	0 0 2	4 3 3	

Need
A B C
7 4 3 — P_0
1 2 2 — P_1 ✗
6 0 0 — P_2 ✗
0 1 1 — P_3 ✗
4 3 1 — P_4 ✗

"Note: Deadlock is avoided by always operating the System in Safe State. the Request of any Process will be granted only if the resulting State of System is Safe"

T_0: ⟨P_1; P_3; P_4; P_0; P_2⟩ "Safe"

T_1: P_1 :→ ⟨1,0,2⟩ : Req$_1$

⟨P_1; P_3; P_4; P_0; P_2⟩ : Safe : ✓ → Pr

T_5: P_4 :→ ⟨3,3,0⟩ : Req : Denied

T_6: P_0 :→ ⟨0,2,0⟩ : Req

⟨ Unsafe ⟩

∴ Req is denied & P_0 is Blocked

Resource - Request Algorithm :

Algo Res_Req(Ri, Reqi, Avail, Alloci, Needi)
{

1. Reqi ≤ Needi
2. Reqi ≤ Avail ;
3. [Assume to've Satisfied the Reqi]
   a) Avail = Avail - Reqi
   b) Alloci = Alloci + Reqi
   c) Needi = Needi - Reqi
4. Run Safety Algorithm
5. if the system is 'Safe' then Grant Reqi
   else deny the Reqi & Block the process

}

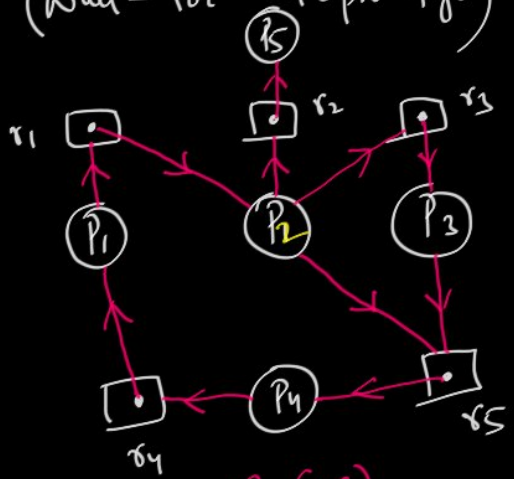
iv. Deadlock Detection & Recovery:

→ Majority of Process are blocked
→ Low cpu utiliz

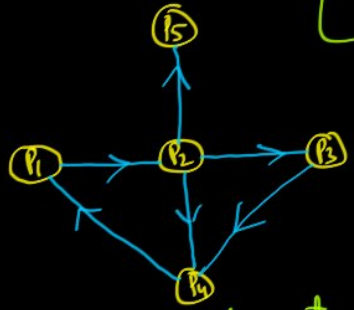
a) Detection Algorithm: <Safe; Deadlock>

1) Single Instance Resource:

(Wait-for-Graph Algo)



G = (V, E)



Run cycle det. Algo.

<P1 - P2 - P3 - P4 - P1>

OP <P1 - P4> : are in Deadlock

Recovery

2) Detection with multi-Instance Resource :

- Five processes (n) P_0 through P_4; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2 ✗	⟨0 1 0⟩
P_2	3 0 3	~~0 0 0~~ 0 0 1 ✗	⟨3 1 3⟩
P_3	2 1 1	1 0 0 ✗	⟨5 2 4⟩
P_4	0 0 2	0 0 2 ✗	⟨5 2 6⟩
			⟨7 2 6⟩

T_0 : ⟨P_0; P_2; P_3; P_4; P_1⟩ : Safe

T_1 : (P_2) : ⟶ ⟨0, 0, 1⟩ : Req — Blocked

⟨0 1 0⟩ ⟨P_0;

∴ deadlock :

OP: ⟨P_1 - P_4⟩ : deadlocked Process

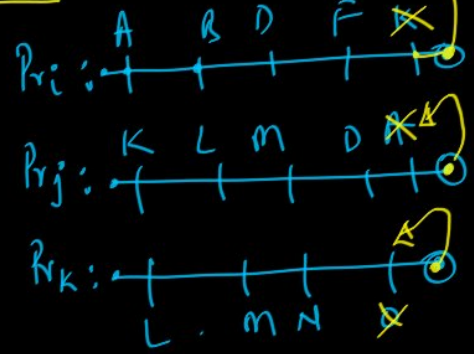
IP: Recovery

Note: The System is Safe (No deadlock) iff the Request of all Process can be satisfied with the avail resources in some order, otherwise the System is said to be deadlock.

Deadlock Recovery Strategies : $\langle P_1 \cdots P_N \rangle$:

P_1 : 95 %.
P_2 : 70 %.
P_3 : 25 %.
P_4 : 5 %.

- Process Termination <Abortion>
  - restarted
    - 1) KILL-ALL
      - → No Need to apply. detection Algo. again
    - 2) KILL ONE @ Time
      - → Apply. det. test Continuously until Deadlock cycle is removed
- 3) Resource PreEmption
  - {Rollback}
  - Starvation



Conceptual Problems:

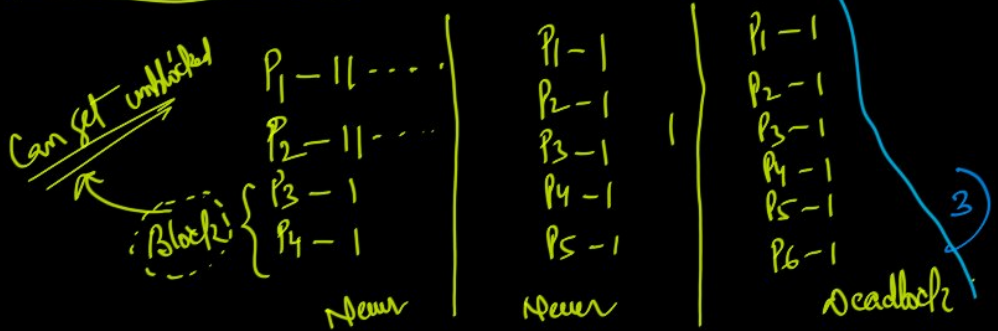
1) n : processes $(P_1 \cdots P_n)$
   R : 6 instances
   Pr_i : 2(R) to Complete

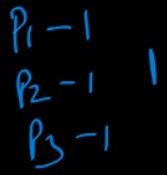
   a) Min(n) for deadlock : 6
   b) Max(n) for deadlock-freedom : 5

Dining Philosophers

2) $n = 3$ proteses
   $R =$
   $Pr_i = 2(R)$

   a) Max (R) to Cause deadlock : 3
   b) Min (R) for deadlock freedom : 4





3) n : Processes

R : 6 Copies

Pr_i : 3(R) to Complete

a) Min (n) for deadlock : 3

b) Max (n) for deadlock : 2 — freedom

$P_1 - 2$	$P_1 - 1$
$P_2 - 2$	$P_2 - 1$
$P_3 - 2$	$P_3 - 1$
	$P_4 - 1$

4) $n = 5$

$R =$

Max for deadlock

	Max R	
P_1	3	− 2
P_2	5	− 4
P_3	8	− 7
P_4	12	− 11
P_5	6	− 5

+1

29

Min (R) for deadlock freedom:

30 ✓

Q5) 'n' – processes; 'R' – Resource

P_i : 'K'(R) to Complete

How many copies min: to be available for deadlock freedom } : $n(K-1)+1$

$nK - n + 1$ ✓

P_1 – K-1

P_2 – K-1

P_3 – K-1

P_4 – K-1 +1

P_5 – K-1

P_n – K-1

$n(K-1)$

02. Consider a System with n Processes <P_1.....P_n>. Each Process is allocated x_i copies of R (resources) and makes a request for y_i copies of R. There are exactly 2 Processes A and B whose request is zero. Further there are 'k' instances of R free available. What is the condition for stating that System is not approaching deadlock (System is said to be not approaching deadlock if minimum request of Process is satisfiable? Also compute the total instances of R in System.

	Alloc / R	Req / R	Avail / R
P_1	x_1	y_1	K
P_2	x_2	y_2	
P_3	x_3	y_3	
P_A	x_A	0	
P_B	x_B	0	
P_n	x_n	y_n	

a) $Total(R) = \sum_{i=1}^{n} x_i + K$

b) T_0: $(K + x_A + x_B) \geq Min(y_i)$, $i \neq (A, B)$ ✓

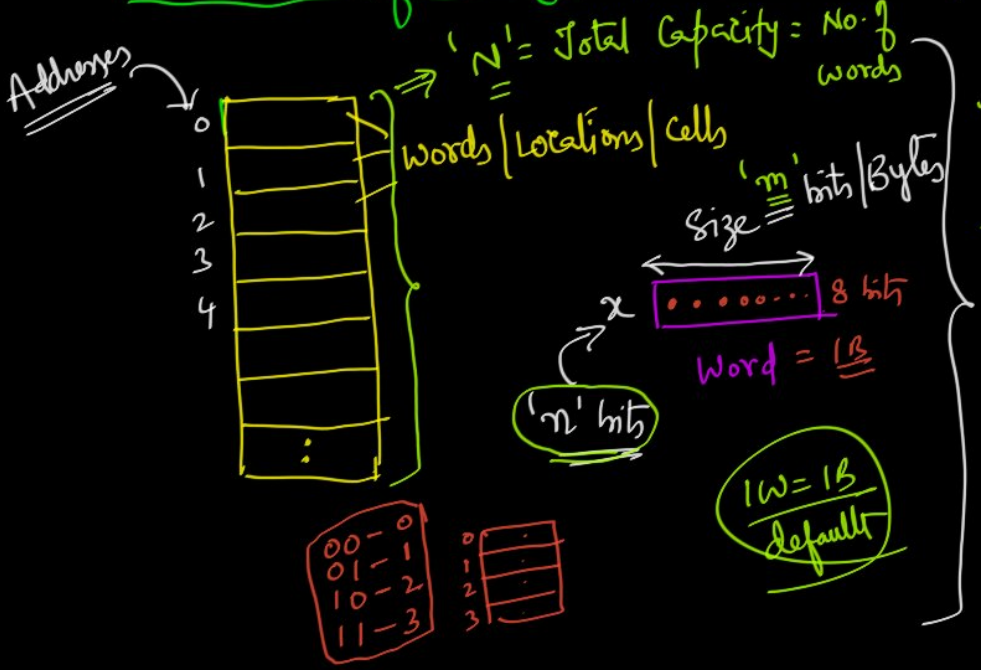
NAD

c) T: $(K + x_A + x_B) < Min(y_i)$: $i \neq (A, B)$ Deadlock

d) T_i: $(K + x_A + x_B) \geq Max(y_i)$: SAFE

II. Memory Subsystem/Management:

Abstract view of Memory: linear 1-D array of words (Instrs/Data units)



I. $N \propto n$

$N = 2^n W$

$n = \log_2 N$ bits

0000 - W0
0001 - W1
:
1111 - W15

1) $N = 16W$, $n = 4$ bits

2) $N = 256W$, $n = 8$ bits

3) $N = 16GB$, $n = 4 + 30 = 34$ bits

1W = 1B default

$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024 = 10^3 = 1K$
$2^{20} \sim 10^6 \sim 1M$
$2^{30} \sim 10^9 \sim 1G$
$2^{40} \sim 10^{12} \sim 1T$
$2^{50} \sim 10^{15} \sim 1P$

4) $N = 1000\,KW$

$n = 20$ bits

5) $n = 26$ bits

$N = 2^{26} = 2^6 \times 2^{20}$

$= 64\,MW$

6) $n = 35$ bits

$N = 32\,GW$

$N = 8$ bits

$m = 2$ bits

$N_W = 4$

7) $N = \text{Log} \times W$

$n = \text{Log}_2 \text{Log}_2 \times$ bits

8) $N = 2^y\,W$

$n = y$ bits

9) $n = 17$ bits ; $m = 16$ bits $= 2B$

$N_W = 2^{17}\,W = 128\,KW$

$N_B = 256\,KB$

$N_{Bits} = 256K \times 8$ bits $= 2^8 \times 2^{10} \times 2^3 = 2^{21} = 2M$ bits

10) $n = 32$ bits ; $m = 64$ bits $= 8B$

$N_{Bytes} = 2^{32} \times 8B = 2^{35} = 32\,GB$

$n_{word} = 32$ bits

11) $N = \frac{512\,G\text{ bits}}{8}$; $m = 128$ bits

$N_{Bytes} = 2^{39}/2^3 = 2^{36} = 64\,GB$

$n_W = 32$ bits ✓

Address

$N_{words} = \frac{512G}{128} = \frac{2^{39}}{2^7} = 2^{32}$

$= 4\,GW$

II. Loading & Linking:

```
10KB: main()
      {
        :
        if(Cond)
          f();
        :
      }

5KB : f()
      {
        :
        g()
        :
      }
```

```
g() : 30KB
{
  :
  h()
  :
}

h() : 15KB
{
  :
  :
20KB: Sqrt();
  :
}
```

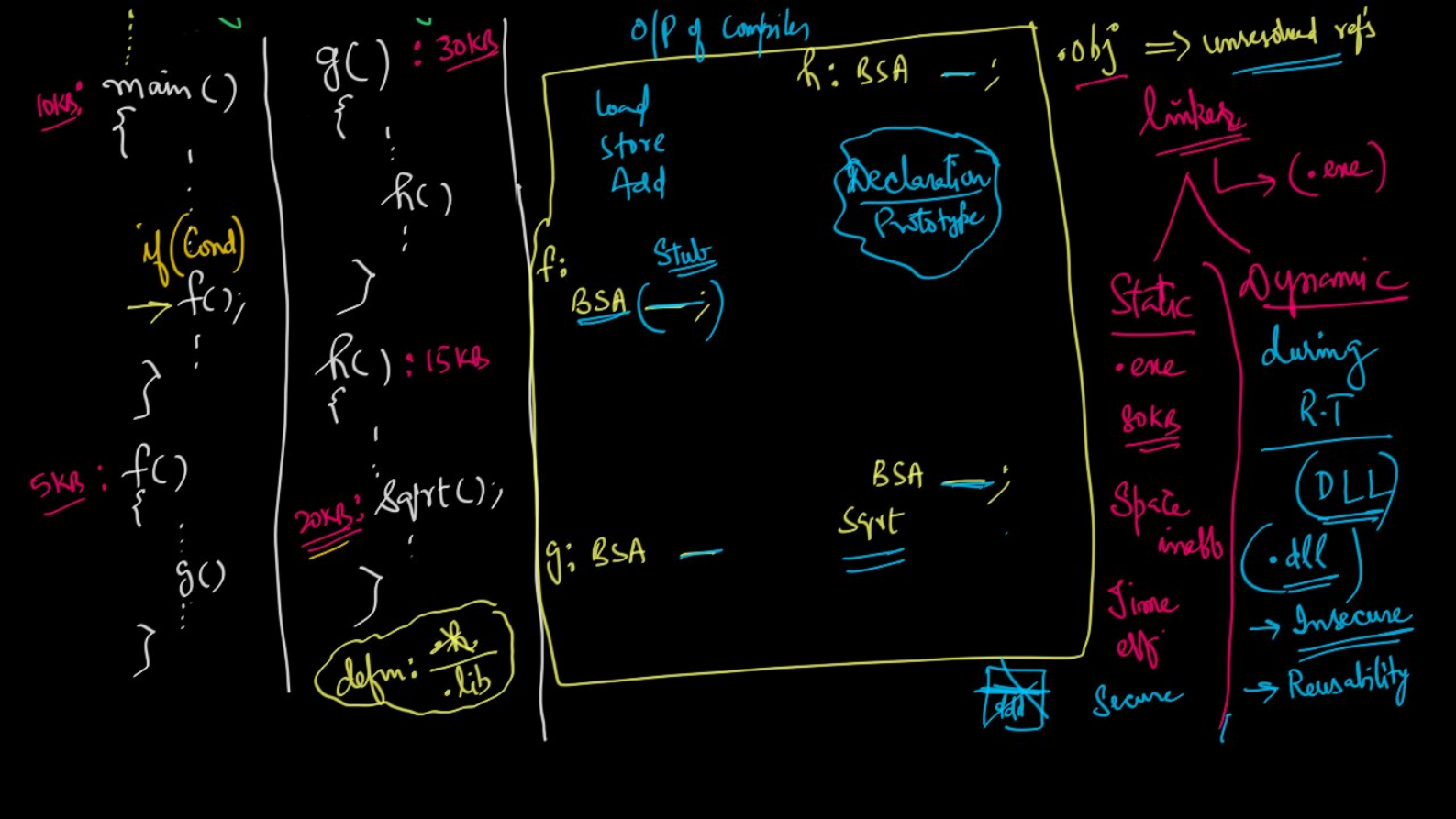
Total Prog-Size on disk = {80KB}

a) Loading: Disk → (Loader) Memory

- Static
  - All Modules are Loaded b/f R.T
  - 80KB
  - (Space ineffi)
  - → Time inefficient
- Dynamic
  - Loading the modules on demand @ R.T
  - → Space efficient

Linking: is the process of resolving external references:
- fns
- variable objects

Ex:
1) extern int x;
2) Func's which are defined later to the call / in other files


10KB
main()
{
if(Cond)
f();
}
5KB
f()
{
g()
}
g() : 30KB
{
h()
}
h() : 15KB
{
20KB
Sqrt();
}
defn: *.lib
O/P of Compiler
h: BSA —;
Load
Store
Add
Declaration
Prototype
f:
Stub
BSA (—;)
BSA —;
Sqrt
g: BSA —
.obj => unresolved refs
linker
(.exe)
Static
.exe
80KB
Space inefft
Time eff
Secure
Dynamic
during
R-T
DLL
(.dll)
→ Insecure
→ Reusability

Address Binding: Association of Program Instns & Data units to memory locations (Addresses) is Address Binding.

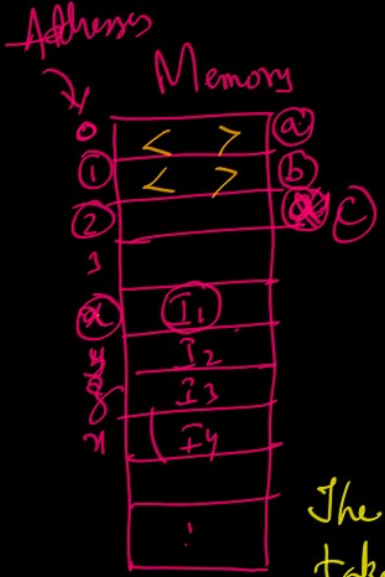
Eg:

int a, b, c; ← Data

c = a + b

↓ Instns

```
I1: Load R1, a
I2: Load R2, b
I3: Add  R1, R2
I4: Store c, R1
```

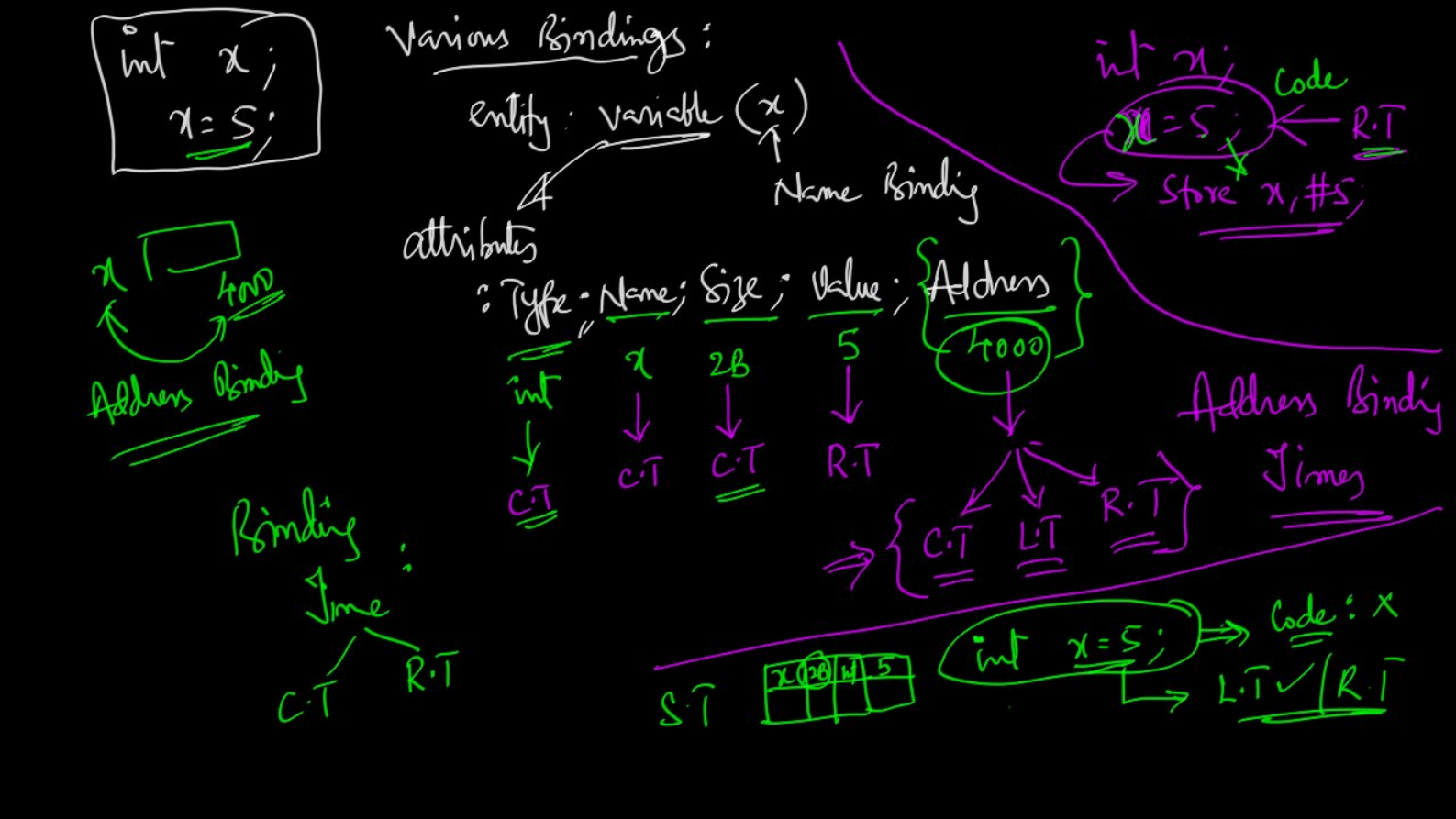


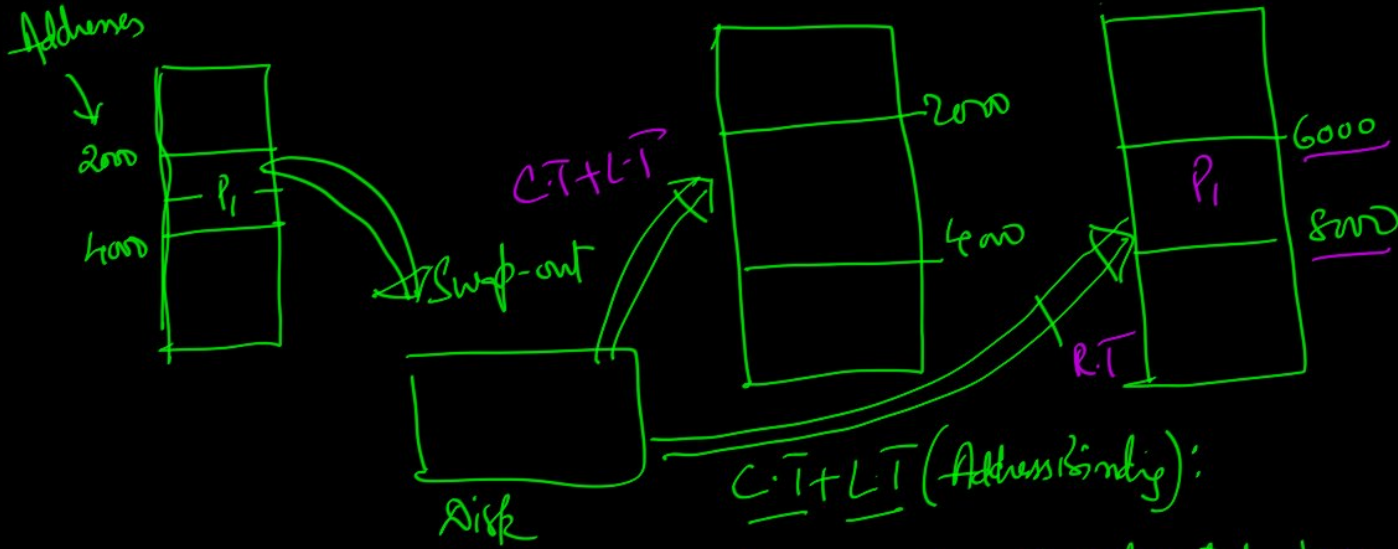
Addresses

I1 : x
I2 : y
I3 : z
I4 : k

a : 0
b : 1
c : 2

The Time @ which this Binding takes place is Binding Time;


int x;
x = 5;
Various Bindings :
Entity : Variable (x)
Name Binding
attributes
: Type : Name ; Size ; Value ; {Address}
int
x
2B
5
4000
C.T
C.T
C.T
R.T
C.T
int x;
Code
x = 5;
R.T
Store x, #5;
x
4000
Address Binding
Address Binding Times
Binding Time :
C.T
R.T
C.T
L.T
R.T
int x = 5;
Code : x
L.T ✓ | R.T
S.T
x
2B
4
5


Addresses
2000
4000
P_1
Swap-out
C·T+L·T
2000
4000
P_1
6000
8000
R·T
Disk
C·T+L·T (Address Binding):
R·T: Program can get relocated to a
different area

Memory Mgmnt Techniques:

- Contiguous (CA): Partitions; overlays; swapping
  - Buddy
- Non-Contiguous (NCA): Paging, Segmentation, Seg-Paging; V.M

Mem. Manager:

functions

- Allocation
- Protection
- Address Translation
- Free Space Mgmnt
- deallocation

Goals

- effective utilization [less wastage]
- Manage execution of large programs in small memory areas
  - Eg: {100 KB} ⇒ {60 KB} (V.M, overlays)
  - Inc. degree of M.Pr → Enhanced thruput

Fragmentation

- I
- E

Contiguous Allocation:

5/7/2021

1. Overlays: {allows execution of larger Program in small memory areas}

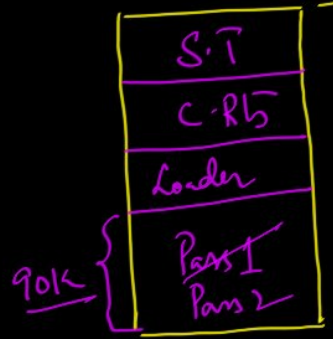
2 pass Assembler: is a Translator (ALP → Target m/c language Program)

Independent

- Pass 1: 70 KB
- Pass 2: 80 KB
- Symbol Table: 30 KB ✓
- Common Rts: 20 KB ✓
- overlay Loader / Driver: 10 KB ✓

Total Prog. Size = 210 KB

150K free Memory

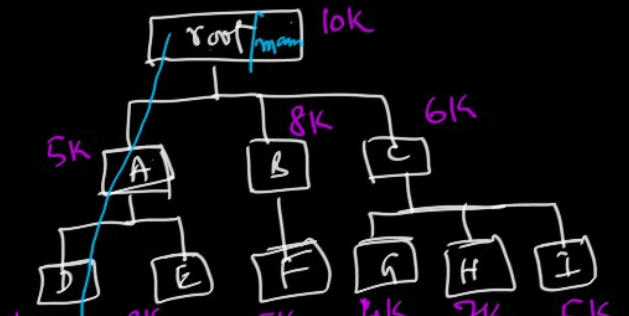


overlay: replace

: overlaying is always NOT possible.
: Prog. must be divided into Mut. exclusive modules.

Program : overlay Tree:

Prog. Size = 56K



Indep. { I, D }

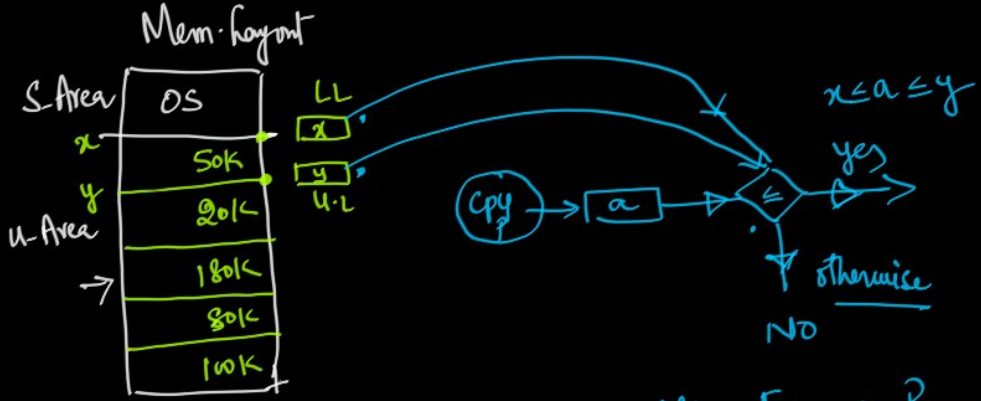
= max { 19K, 17K, 23K, 20K, 23K, 21K }

Min. Amount of Memory Needed to execute this program with overlaying = max { Path_length from root to leaf }

= { 23K } ✓

2) Partitions : — Fixed Partitions : MFT : Multiprograg with Fixed Task
Variable Partitions : MVT : " " " Variable "

a) Fixed Partitions (MFT) : Static Partitioning → u-Area is divided into Partitions
→ {1 partition = 1 program}



Allocation : Pmem : {15K}

Protection

L-L U-L

Part. Alloc. Policies

1. First Fit (FF)
2. Best Fit (BF)
3. Next fit (NF)
4. Worst Fit (WF)

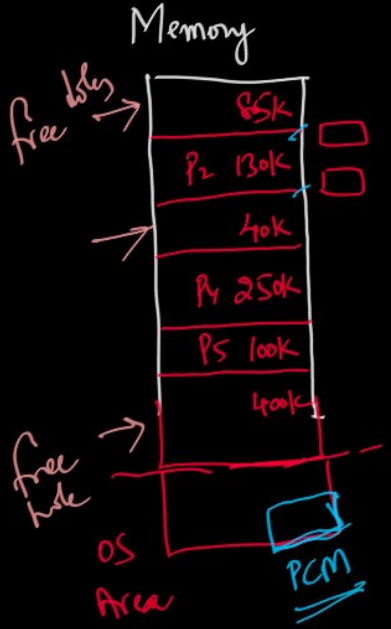
1. F.F: First free big enough {linear search | linear probing}
2. B.F: Smallest free " " (Best Available Fit)
3. Worst Fit(WF): largest free big enough
4. Next Fit(NF): Works like FF; except that Search fn free big enough Partition is made from the partition where Last allocation was made.

Adv.: NF may work faster to FF

Perf. Issues:

1) Internal Fragmentation (IF): ✓
2) External " (EF): ×
3) degree of M.Pr: Limited
4) Max. Process Size: Ltd to Max. Partition Size
5) Partition Alloc. Policy: B.F (due to less I.F)

2) Variable Partitions (MVT): Dynamic Partitioning



't': Incoming Process Req's: P1 85K; P2 130K; 40K; 250K; 100K P5-

Pnew: P6: {35K}

Perf. Issues:

1) Int. Frag: X
2) Ext Frag: ✓ ;
   < Contiguity >
3) Degree of M. Pr: Flexible
4) Max Process Size: Flexible
5) Alloc. Policy: W.F is Superior

(i) FF
(ii) BF:
(iii) NF
(iv) WF: ✓

Size of Process: P6: {410K}

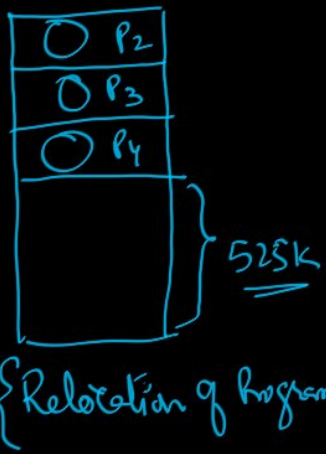
Total free Memory:

Solutions for External fragmentation :

- Non-CA Allocation :
  - Compaction | Defragmentation

Compaction : Merge all NCA free holes @ one place of Memory

Compaction



{Relocation of Programs}

Drawbacks | Limitation

(i) Lot of time Consumption

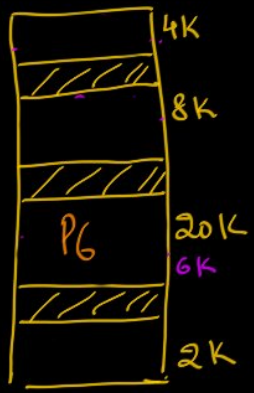
(ii) Compaction is Possible only if Programs support R.T Address Binding

05. Consider a System Using Variable Partition with no Compaction

Free holes	4K; 8K; 20K; 2K
Program size	2K; 14K; 3K; 6K; 10K; 20K; 2K
Time for Execution (B.T)	4; 10; 2; 1; 4; 1; 8

Using **Best Fit Allocation Policy** and **FCFS CPU Scheduling Technique**, Find the Time of Loading & Time of Completion of each program. The Burst Times are in Seconds.

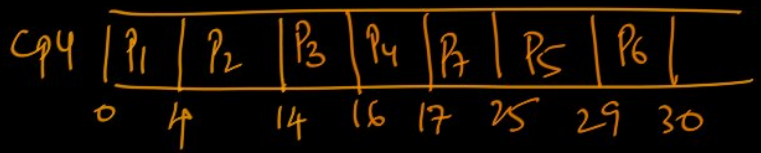
T_0:



$T_0L(P_1, P_2, P_3, P_4, P_7) = 0$

$T_0L(P_5) = 14$

$T_0L(P_6) = 29$

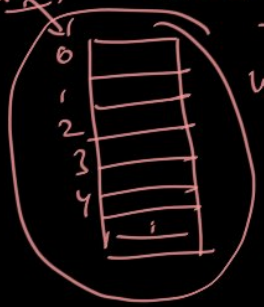


II. Non-Contiguous Memory Allocation :

a) Paging

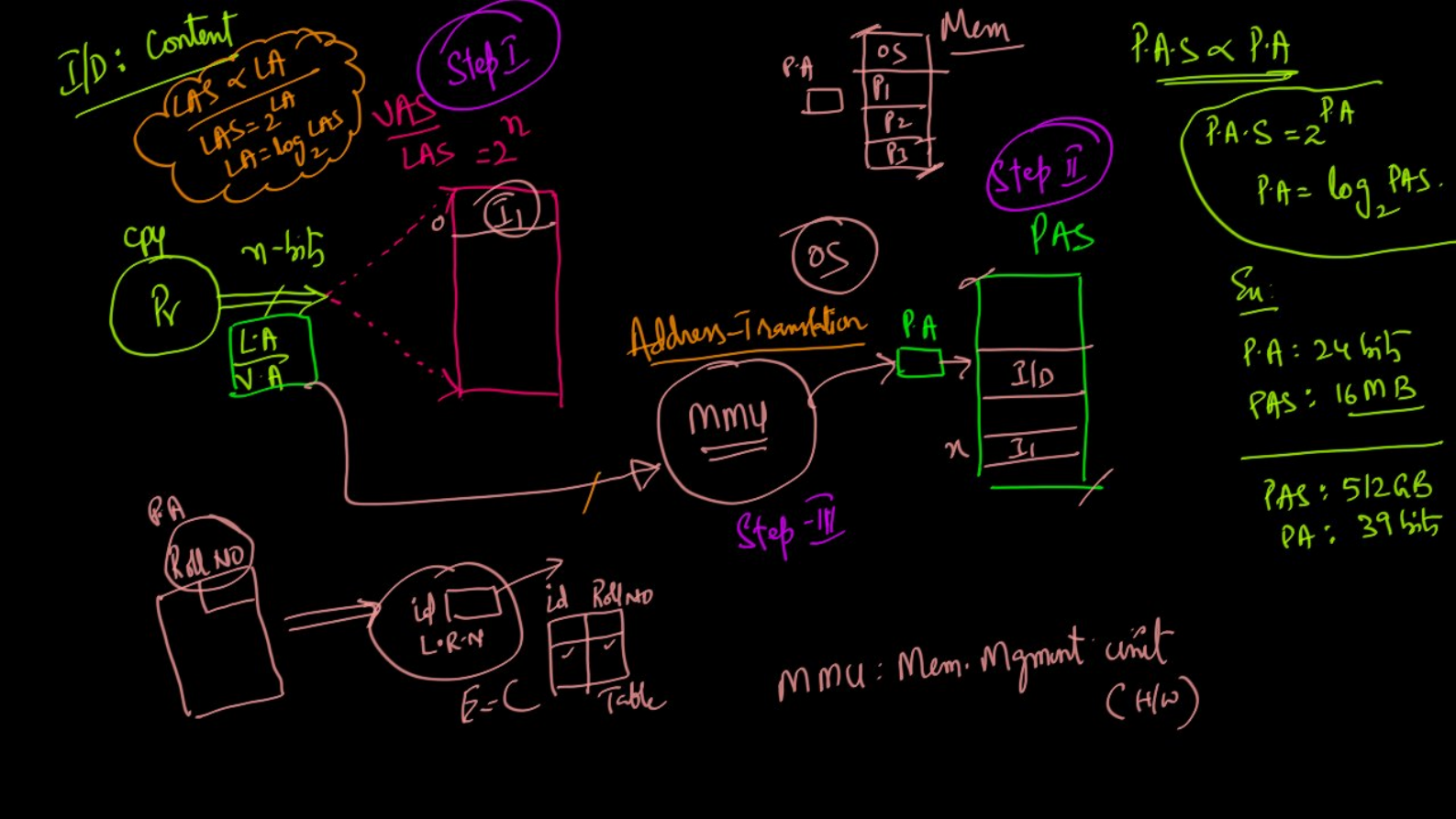
Logical (Virtual) (LAS/VAS) vs Physical Address Space (PAS)

Address-Space : { group/set / Space of words, where each word is associated with an addrs address }



Words / Bytes = P.A.S = Physical Mem / Main Memory

Physical Address / P.A = bits


I/D : Content
LAS ∝ LA
LAS = 2^LA
LA = log2 LAS
Step I
VAS
LAS = 2^n
CPU
n-bits
PC
L.A
V.A
I1
P.A
OS
P1
P2
P3
Mem
Step II
PAS
OS
Address-Translation
P.A
MMU
I/D
n
I1
Step -III
P.A
Roll NO
id
L.R.N
id
Roll NO
Table
E=C
MMU : Mem. Mgment unit
(H/W)
P.A.S ∝ P.A
P.A.S = 2^P.A
P.A = log2 PAS.
Eg:
P.A : 24 bits
PAS : 16 MB
PAS : 512 GB
PA : 39 bits

I. Simple Paging :

L.A.S = 8KB ; P.A.S = 4KB ; Page-Size (PS) = 1KB = F.S

L.A = 13 bits ; P.A = 12 bits

a) organization of LAS/VAS :

→ L.A.S is divided into equal size units called Pages.

→ Page-Size (P.S) is generally a power of 2 (2^KB)

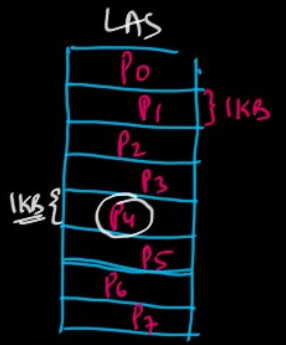
→ No. of Pages (N) = $\frac{LAS}{PS}$ — (1)

→ Page No (P) = $\log_2 N$ bits

$N = 2^P$

→ Page offset (d) = $\log_2 PS$ bits

$PS = 2^d$



→ LA : | P | d |

Format 3 10

13 bits

1KB { 0 1 ⋮ } PS = 10 = d $P.S = 2^{10} = 1KB$

Q1) L.A.S = 2MB ; P.S = 2KB

$N = 2^{21}/2^{11} = 2^{10} = 1K$

P = 10

d = 11

LA = 21 bits

LA. format = | P | d |

10 11

21 bits

Q2) System has 512 pages with a page offset of 11 bits; What is the size of LAS?

$N = 512$; $d = 11$ bits $\Rightarrow P.S = 2^d = 2^{11} = 2KB$ ✓

$P = 9$ bits

$L.A.S = ?$

(i) $L.A.S = N * P.S$
$= 512 * 2KB$
$= 1024\ KB$
$= 1\ MB$ ✓

(ii) L.A: | P | d |
9 11
20 bits

$\Rightarrow LAS = 2^{20} = 1MB$

Q3) System supports a L.A of 32 bits and 2K Pages; Calculate P.S:

N

P = 11

LA: | P | d | (32 bits)
11 21

$\therefore d = 21$ bits

$\Rightarrow P.S = 2^d = 2^{21} = 2MB$ ✓

B = Bytes
b = bits

b) Orgnz. of P.A.S :

PAS = 4KB; FS = PS = 1KB

→ P.A.S is divided into equal size units known as frames (Page-frames)

→ Frame_Size = Page_Size

→ Each frame holds a Page.

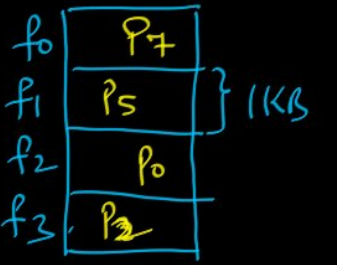
→ Any frame can hold any Page (∵ NCG)

→ **No. of frames (M) = PAS / PS**

→ **Frame No. (f) = $\log_2 M$ bits**

$M = 2^f$

PAS = 4KB



→ **frame_offset = Page offset = d**

→ P.A : | f (2) | d (10) |

Q4) L.A.S = 4 MB ; P.A.S = 32 KB ;

P.S = 2KB

L.A = 22 bits ✓

P.A = 15 ✓

$N = 2^{22}/2^{11} = 2^{11} = 2K$

$M = 2^{15}/2^{11} = 2^4 = 16$

p = 11

f = 4

d = 11

L.A format = | p (11) | d (11) |

P.A format = | f (4) | d (11) |

Q5) System supports 512 Pages & 64 frames; If LA is 18 bits then Calculate the Size of PAS.

Linear Algebra

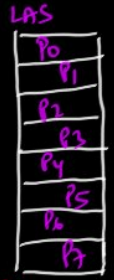
N = 512 ⇒ P = 9 ; M = 64 ⇒ f = 6

LA: | p (9) | d (9) | — 18

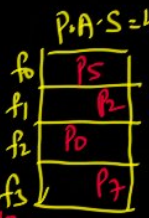
P.A: | f (6) | d (9) | — 15

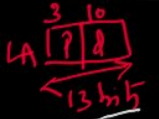
$\Rightarrow P.A.S = 2^{15} = 32KB$ ✓

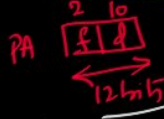
L.A.S = 8KB; P.A.S = 4KB; P.S = 1KB;



06/07/2021







c) orgn. of M.M.U (Memory Mgmnt. unit)

Page Table (P.T)

→ Each process has its own P.T

→ P.T's are stored in Memory

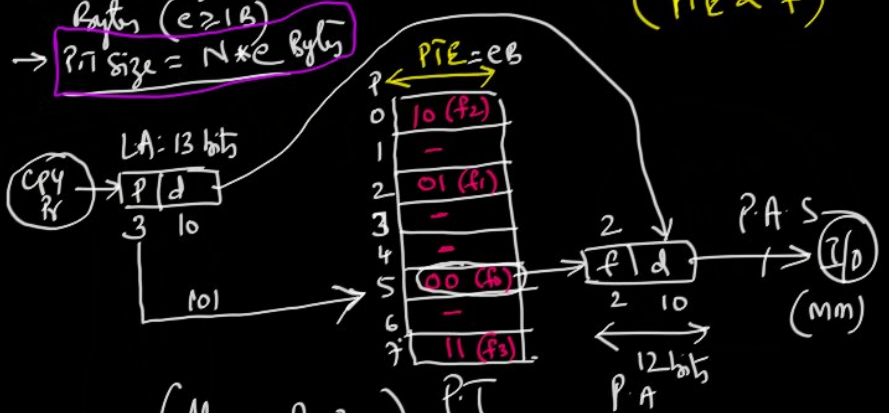
→ P.T is organized as set of entries, known as Page Table entries (PTE)

→ No. of entries in P.T = No. of Pages in A.S

→ P.T.Es contain the frame no. in which the referred Page is present.

→ P.T.E Size (e) is generally in multiples of Bytes ($e \geq 1B$) ($PTE \propto f$)

→ P.T Size = N * e Bytes



(Algo of Paging)

07. A Computer System using **Paging Technique** implements an 8KB Page with a **Page Table of Size** 24MB. The Page Table Entry is 24 bits. What is the **length of Virtual Address** in this System?

08. A Computer System using Paging technique has a Virtual Address of length ℓ. The Number of Pages in the Address Space are 'Z'. There are 'H' Frames in PAS. Calculate the number of bits in Page Offset and the size of PAS.

Q7 P.S = 8KB ; PTS = 24MB ; PTE = e = 24 bits = 3B

LA = ? d = 13

= | p | d |

23 13

LA = 36 bits ✓

PTS = N * e

24MB = N * 3B

$N = \frac{24MB}{3B} = 8M$

∴ N = 8M

P = 23 bits

Q8 V.A = 'ℓ' bits

N = Z

M = H

V.A: | p | d | (p: log Z, d: ℓ − log Z)

∴ $d = (\ell - \log_2 Z)$ bits ✓

$P = \log_2 Z$

P.A: | f | d | (f: log H, d: ℓ − log Z)

∴ P.A.S = $2^{\log H + \ell - \log Z}$ Bytes ⇒ $2^{\ell} \cdot 2^{\log_2 \frac{H}{Z}}$

⇒ $\left(\frac{H}{Z}\right) * 2^{\ell}$ Bytes PAS

09. Consider a computer system with 40-bit virtual addressing and page size of sixteen kilobytes. If the computer system has a one-level page table per process and each page table entry requires 48 bits, then the size of the per-process page table is_____ megabytes.

10. Consider a System using **Simple Paging Technique** with Logical Address (LA) of 32 bits. [redacted] Page Table Entry (PTE) 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)?

Q9) L.A = 40 bit ; P.S = 16 KB ;

PTE = 6 B

e = 6 B

PTS = N * e

$= 2^{26} * 6B$

$= 64\,MB \times 6$

$= 384\,MB$

$N = \frac{2^{40}}{2^{14}} = 2^{26}$

Q10 LA = 32 bits ; PTE = 32 bits = 4B

Let P.S = 2^x By = 128 KB

frame-size = 2^x By

PTS = N * e

$= \left(\frac{2^{32}}{2^x}\right) * 4B = \binom{32-x}{2} 4B = \left(2^{34-x}\right) By = 2^x\ By$

PTS

$34 - x = x$

$34 = 2x$

$x = \frac{34}{2} = 17$

$\therefore PS = 2^{17} = 128KB$

Performance of Paging :

a) Temporal Issues : ∴ Reduce EMAT

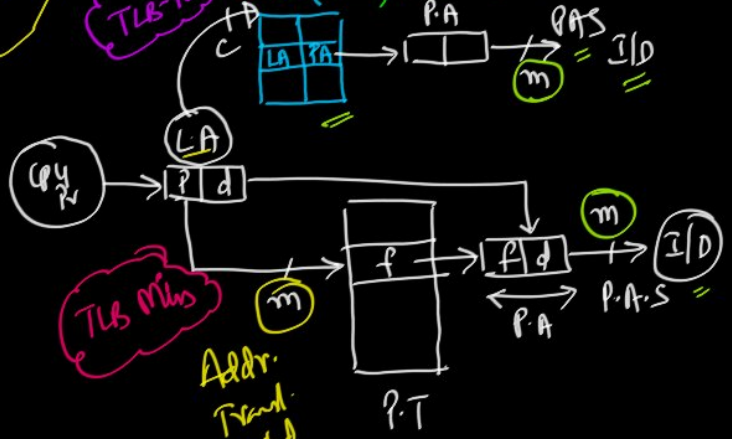
→ Main Memory access_time (MMAT) = 'm'

→ Effective " " " (EMAT) = '2m'

Paging with TLB

TLB-Hit

Cache (TLB) : Translation Look Aside Buffer



(p) (d)

→ TLB AT = 'c' (c ≪ m)

→ TLB Hit ratio (x) = No. of hits / Total Refs

→ TLB Miss ratio = (1 − x)

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

Q: c = 20 ns ; m = 100 ns

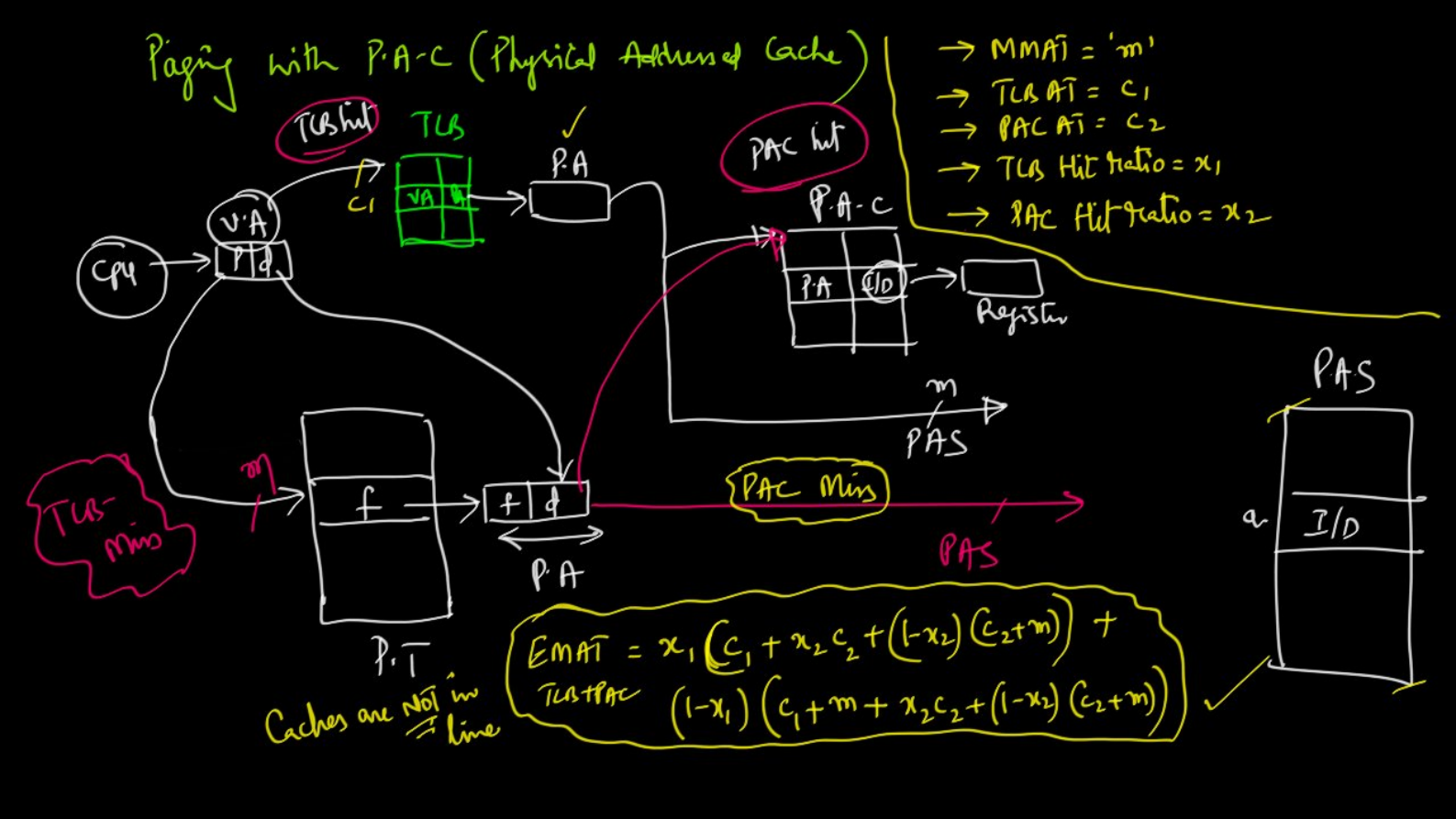
1) x = 90%

EMAT = 0.9 (120) + 0.1 (220) = 130 ns

2) x = 10%

EMAT = 0.1 (120) + 0.9 (220)

= 210 ns


Paging with P·A-C (Physical Addressed Cache)
→ MMAT = 'm'
→ TLB AT = c_1
→ PAC AT = c_2
→ TLB Hit ratio = x_1
→ PAC Hit ratio = x_2
TLB hit
TLB
P·A
PAC hit
P·A-C
V·A
CPU
p
d
c_1
VA
PA
PA
I/D
Register
m
PAS
TLB-Miss
m
f
f
d
PAC Miss
PAS
P·A
P·T
Caches are NOT in line
$EMAT_{TLB+PAC} = x_1(c_1 + x_2 c_2 + (1-x_2)(c_2+m)) + (1-x_1)(c_1 + m + x_2 c_2 + (1-x_2)(c_2+m))$
PAS
a
I/D

12. 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 the CPU time is devoted to loading the Page Tables?

Time to load the P.T = (Time to load one entry) * No. of entries

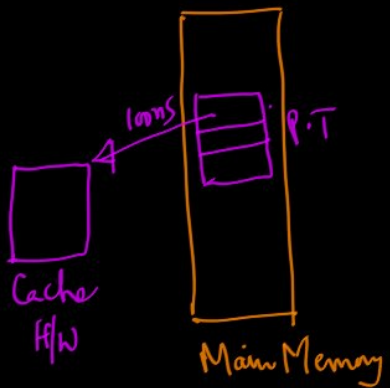
$= 100ns * (N)$

$= (100 \times 512 \times 1024\ ns)$

$N = \frac{2^{32}}{2^{13}} = 2^{19} = 512K$

% CPU_time $= \left(\frac{100 \times 512 \times 1024 \times 10^{-9}}{100 \times 10^{-3}}\right)$

$\simeq 50\%$



P.T.S = N * e

$= 512K * 4B = 2MB$ ✓

2) Spatial Issue:

L.A = 32 bits; P.S = 4KB

P.T. Size approximately = ?

$N = \frac{2^{32}}{2^{12}} = 2^{20} = 1M$

objective:

To Reduce (P.T. Size)

= ?

PT = 1M entries

if e = 4B

PTS = 4MB ✓

$N = \frac{LAS}{PS} \Rightarrow \left(N \propto \frac{1}{PS}\right)$

$P.T.S = N * e$

$\Rightarrow \{PTS \propto N\}$

---

a) Increase Page Size:

$\left\{PTS \propto N \propto \frac{1}{PS} \Rightarrow \left(P.T.S \propto \frac{1}{PS}\right)\right\}$

Ex: Prog = 1026 By (Not desirable)

P.S

L {1024B} — 2 pages — I.F: 1022B

S 2B — 513 pages — I.F: 0B

If always occur in last Page

⇒ **Optimal Page Size:** ? What should be the ideal size of Page such that the P.T.S & I.F is less ?

Let V.A.S = 'S' By ; P.S = 'P' By ; PTE = 'e' By

$$P.T.S = \left(\frac{S}{P}\right) * e \text{ By} \quad ; \quad I.F = \frac{P}{2} \text{ By}$$

$$\frac{d}{dP}\left(\frac{S}{P} * e + \frac{P}{2}\right) = 0$$

$$\frac{-Se}{P^2} + \frac{1}{2} = 0$$

$$\Rightarrow \boxed{P = \sqrt{2 \cdot Se} \text{ By}}$$

Optimal Page-Size

Note: Smaller Pages can accomodate Programs Comfortably. with less I.F

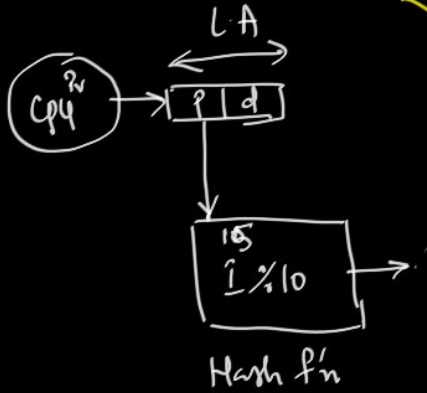
2) Hashed Paging : Process will be associated with a hashed P.T;

Collission: $(I_1, I_2) \Rightarrow$ 'i'

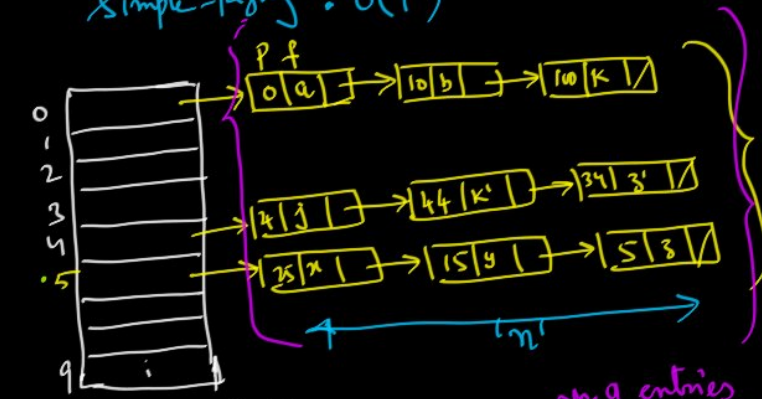
Linear Probing

chaining ✓

<0, 10, 100, 25, 15, 5, 34, 4, 44>



Simple-Paging : O(1)



→ Searching time in H.P.T chain : O(n) ✓

No. of entries in the (Nodes) Hashed P.T chains = No. of Pages in Memory

* 3) Multi-Level Paging / Hierarchical Paging / Recursive Paging : (Obj: is to assoc. smaller P.Ts with the process)

: Paging on Page Table is MLP :

7/7/21

L.A = 32 bits ; P.S = 4KB ; $N = \frac{2^{32}}{2^{12}} = 2^{20} = 1M$

In Memory we will have only O.P.T and the required pages of I.P.T.

I.T.Size ovhd:

$(1K + 2 \times 1K) \times 4B$

$= 12KB$ ✓

1st level

1K

0	x (4B)
1	
1023	y

Outer P.T (OPT)

(1M − 5) = blank

Let P.S of P.T = 1K

2nd level

1M entry

P0	a
1	b
2	c
3	d
	1K
	1K
	P1023
	z

I.P.T

$N_{pages} = \frac{1M}{1K} = 1K$

P.A.S

Pages of VAS

f0	P0
f1	P1
f2	P2
x	P0
	P3
fm	Pn
y	P1023

4KB

VAS = 4GB

Code	P0
	P1
	P2
Data	P3
	P31
Stack	

4KB

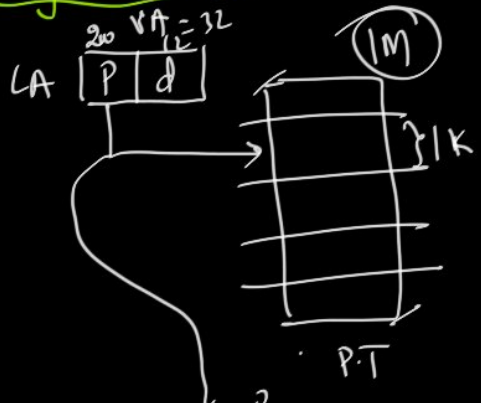
Min: 8KB

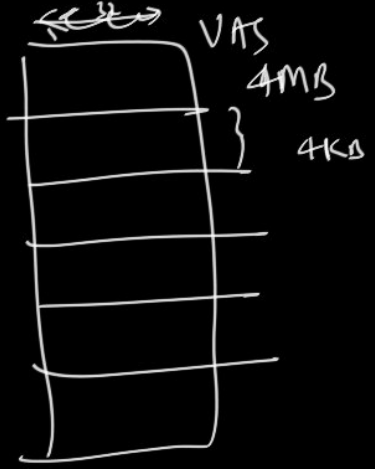
Max: 24KB

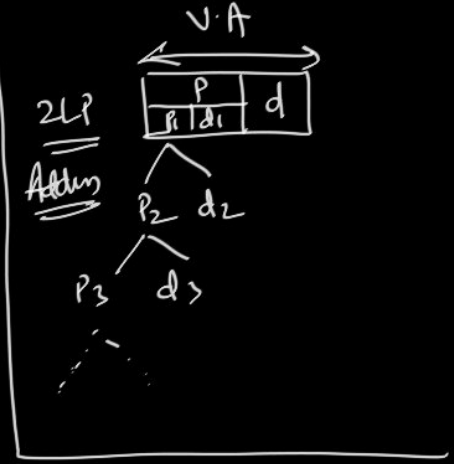
NCG

5 Pages (2 code; 2 Data; 1 Stack)

Addrng in MLP:



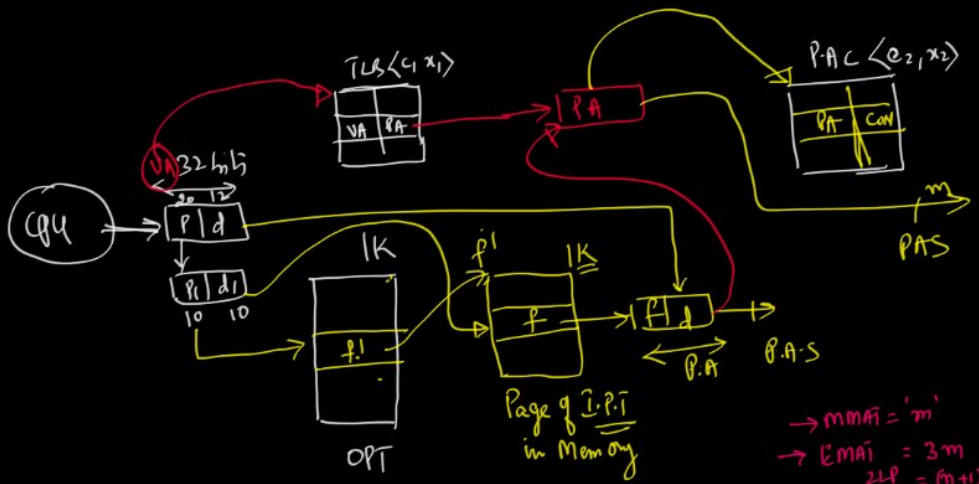




→ P.S of I.P.T

→ No. of Pages in I.P.T = No. of entries in O.P.T

Address Translation : In 2-level Paging



→ MMAT = 'm'

→ EMAT = 3m

$\overset{2LP}{MLP} = (n+1)\,m$

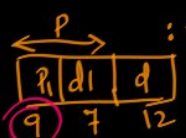
$$\overset{2LP}{EMAT} = x_1\left[c_1 + x_2 c_2 + (1-x_2)(c_2+m)\right] + (1-x_1)\left[c_1 + m + m + x_2 c_2 + (1-x_2)(c_2+m)\right]$$

$$\overset{2LP}{MLP} = (1-x_1)\left[c_1 + n*m + 1\right]$$

15. 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 Address 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?

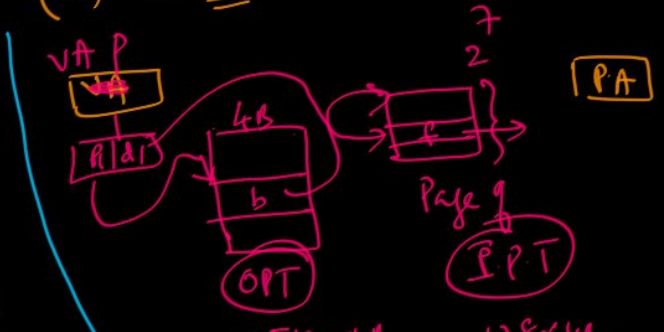
(ii) If P.T.E at both levels is 32 bits in size then what is the Space Overhead needed to translate Virtual Address to Physical Address of an Instruction or Data Unit?



(i) $P.S(VAS) = 2^{12} = 4KB$ ✓

$N\ pages\ in\ VAS = 2^{16} = 64K$

(ii) $e = 4B$



$= 512 \times 4B$

$128 \times 4B$

$2^{16} \times 4 = 2^{18}$

$256KB$

512
128
640

$640 \times 4B$

$2560\ B = 2.56KB$

16. Consider a Computer System using 3 Level Paging Architecture with a 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 access all the three levels of Page Tables and the Page offset of Virtual Address Space.

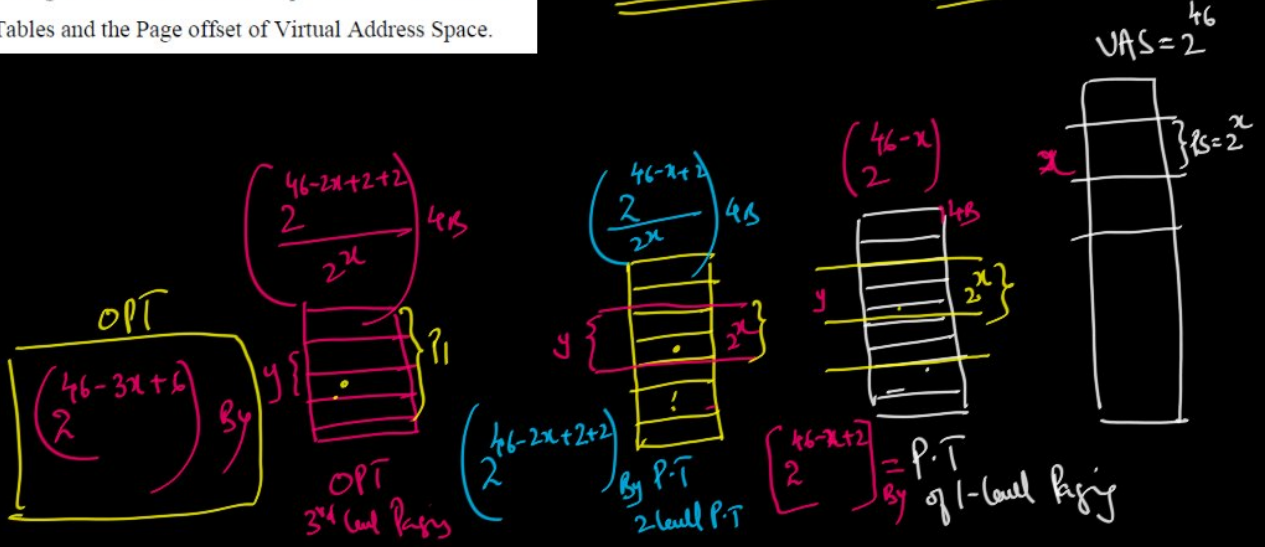
Let $P.S = 2^x$ B

$F.S = 2^x$

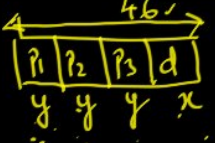
$\therefore P.S = 2^{13} = 8$ KB

$2^{46-3x+6} = 2^x$

$\therefore x = 13$



Address Format :



$\therefore 3y + x = 46 \quad -(1)$

$y = 11$

$3(x-2) + x = 46$

$3x - 6 + x = 46$

$4x = 52$

$x = 13$

$\therefore PS = 2^x = 2^{13} = 8\,KB$

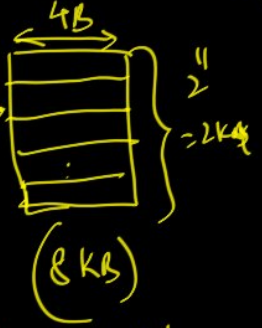
Let $PS = 2^x$ By

OPT should fit in one frame (2^x)

Size of OPT $\Rightarrow$ $(2^y)\,4B = 2^x$ By

$y + 2 = x$

$y = x - 2$



13 bits

(In the P.T Addressing will be done to the entry of PT)

11. SEGMENTATION:

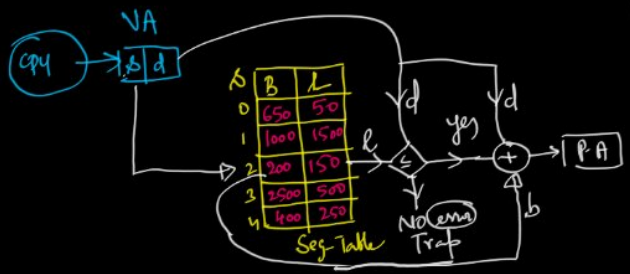
→ Paging does not preserve user's view of Memory allocation to Programs

→ As per user's view of Mem. allocation, program is divided into logical units, known as segments. A segment may represent a function/Procedure, O.S, S.T, ...

→ These segments are assumed to be stored in PAS @ non c.g locations.

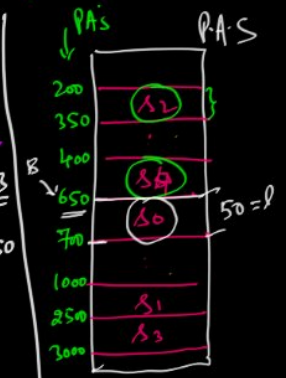
S = Seg-No
B = Base Address
L = Length of Seg

Program



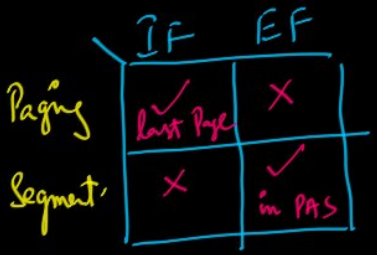
Eg:

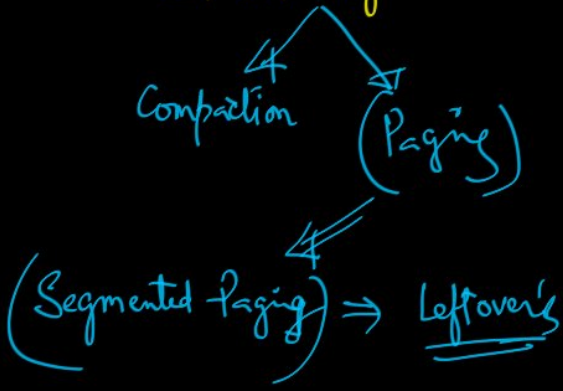
L.A.	PA
⟨2, 23⟩	200+23 = 223
⟨4, 50⟩	400+50 = 450
⟨1, 1850⟩	Trap
⟨3, 5⟩	2505
⟨6, 2⟩	Trap



How is PAS organized in Segmentation: as variable partitions:

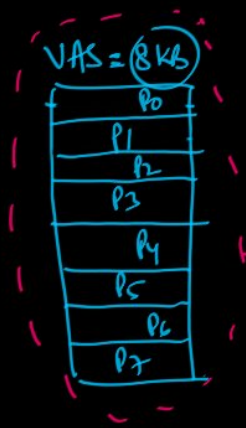
↳ ext. Fragmentation





Virtual Memory (VM) = { gives illusion to the Programmer that a huge amount of memory is available for executing Programs greater than the size of given available Physical Memory }

Ex: L.A.S / VAS = 8KB; ; P.A.S = 4KB; P.S = 1KB



(Simple Paging)

(Demand Paging (DP))

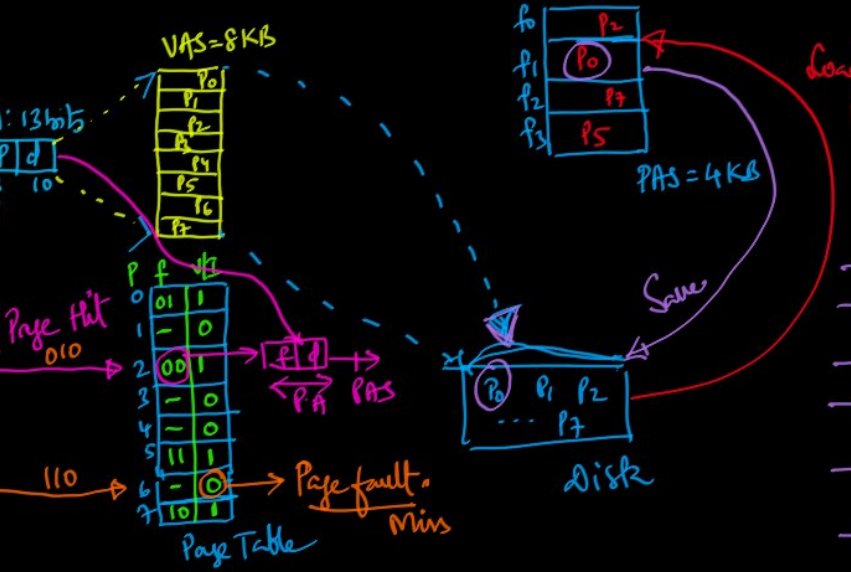
- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
- Virtual memory can be implemented via:
  - Demand paging ✓ (DP)
  - Demand segmentation – Theoretic

DP: Loading the pages of the program on demand @ Run Time from Disk to Memory.

8/7/2021

L.A.S = 8KB; PAS = 4KB; P.S = 1KB; N = 8 Pages; M = 4; P = 3; f = 2; d = 10 bits

V/I : Valid / Invalid bit



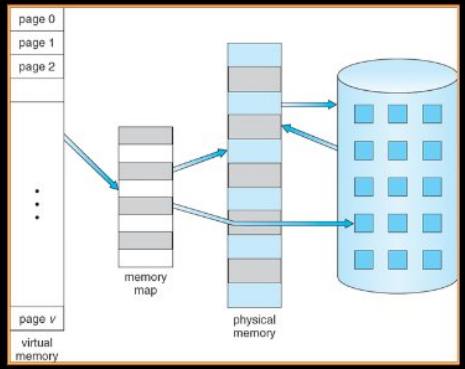
→ Process gets Blocked
→ OS VM M
→ VM → D.M
→ D.M → locate the Page
→ Load the Page into Memory
→ Page replacement

Demand Paging (DP)

Pure (DP)

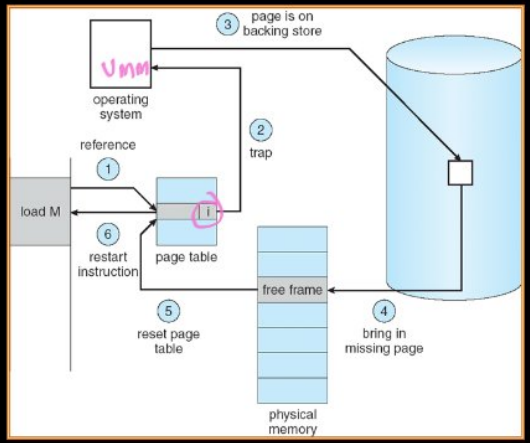
(Execution of Program starts with all Empty frames)

Prefetch (DP)



{The Time to serve the Page fault is known as Page-fault Service Time}

{PFST = ms}



Steps of Page fault Service

Performance of V.M:

→ Temporal Issue:

→ MMAT = 'm'

→ PPST = 'S' [S >> m]

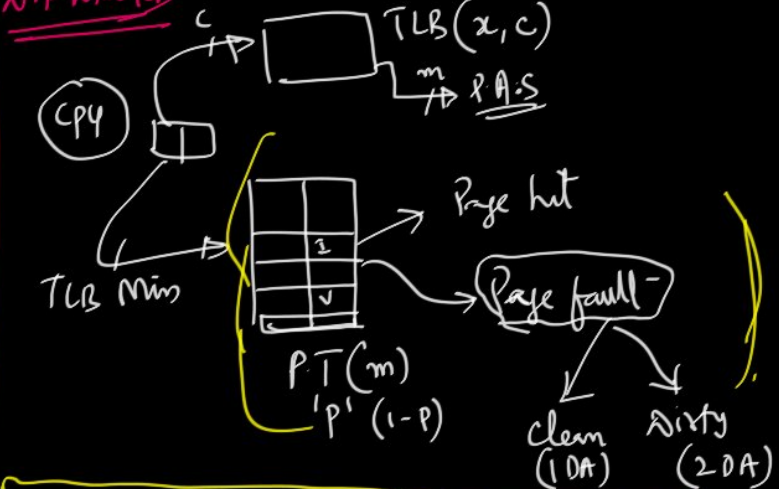
→ Page fault rate = 'P' [0 ≤ P ≤ 1]

→ Page-Hit ratio = '1-P'

E.M.A.T = (1-P) m + P * S

DP ✓

D.P with TLB



E.M.A.T = x(c+m) + (1-x)[c + (1-P) m + P * S] ✓

TLB+DP

→ Page-Replacement:

→ Reference_string = {Set of {successively unique} pages referred in the given list of V.As}

P.S = 100 P_i → V.A's = { 704; 823; 012; 056; 084; 122; 182; 801; 755; 964; 533; 550 }

Ref. string = { 7; 8; 0; 1; 8; 7; 9; 5 }

l = length = 8

n = no. of unique Pages = {0, 1, 5, 7, 8, 9} = 6

704

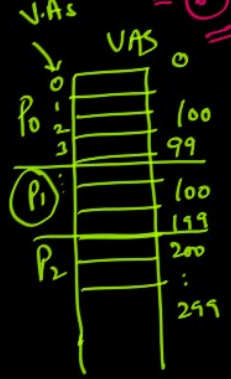
VA | P | d

P = VA/PS

d = VA % PS

Process_demand_frames = 6

Frame Allocation Policies:



Frame Alloc. Policies:

- → no. of Processes = $P_1 \cdots P_n$
- → Total Frames Avail = M
- → Demand of P_i = 'S_i'
- → Frames alloc. to P_i = a_i = ?
- → Total Demand = $D = \sum_{i=1}^{n} S_i$

Eg: $n = 5$; $M = 45$

	S_i	a)	a_i b)	c)
P_1	10	9	5	
P_2	25	9	12	
P_3	5	9	2	
P_4	35	9	17	
P_5	15	9	7	
	$D = 90$			

✓ a) Equal Alloc:

$a_i = M/n$

✓ b) Proportionate Alloc:

$a_i = \left(\frac{S_i}{D}\right) * M$

✓ c) 50% rule

$a_i = \left(S_i/2\right)$

Page Replacement Techniques :

Ref. string = < 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 >

$l = 20$ ✓

$n = 6$

$K = 6$

P.D.P

L.R.U ~ optimal

Impl of {L.R.U} / LRU Approximations

(i) FIFO :

Criteria: T.O.L

3 Frames = 15 ∴ $P = \frac{15}{20} = 75\%$

4 Frames = 10 ; ∴ $P = \frac{10}{20} = 50\%$ ↓

(ii) optimal Replacement:

{Replace the Page which will not be referred for the longest duration of time in future references}

3F : 9

4F : 8 "Benchmark"

(v) Counting Algo's

a) L.F.U : 13

b) M.F.U : 12

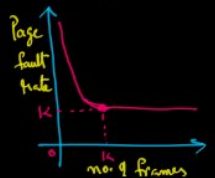
Criteria : C.O.R

Conflict : FIFO

(iv) M.R.U [most recently used]

3F : 16

4F : 12 ↓



↓

Criteria : T.O.R

(iii) L.R.U [Least Recently used]

Not used for longest duration of time in the past.

3 Frames : 12 ✓

4 Frames : 8 ✓ ↓

Ref. string: <1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5>

(i) FIFO:

3F: 9 ↑
4F: 10

(ii) Optimal:

3F: 7 ↓ (No Belady's Anomaly)
4F: 6

(iii) LRU:

3F: 10 ↓
4F: 8 (No Belady's Anomaly)

(iv) MRU:

3F:
4F:

Belady's Anomaly:

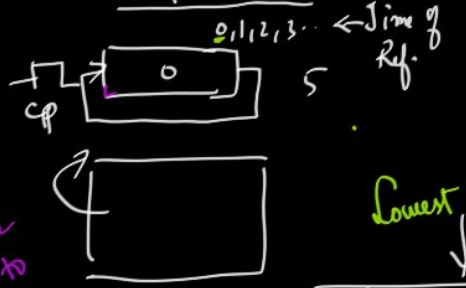
{ With the increase in no. of frames to the Process, Page fault rate also sometimes increases }

only FIFO and FIFO-Based Algo. Suffers from Belady's Anomaly

LRU Implementations:

(i) Counter Method:

n-bit-Counter



LI: Once the counter reaches last Count value Then it resets to '0' & Algo. fails

Lowest value is LRU

P	f	V/I	T.O.R
0	a	1	15
1	b	1	12
2	-	0	-
3	c	1	8
4	d	1	10

P. fault → (2)

(ii) Stack Method:

: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, ...

Never fail

3-frames

ouhd: '2n' Stack opn's even if no Page fault.

"2n"

'n'

STACK Page No's are Pushed onto Stack.

LRU Approximations : { class of replacement techniques that pretend to work like LRU }

a) Reference Bit :

Criteria : R : 0 : Page NOT refered so far during the present Epoch ;

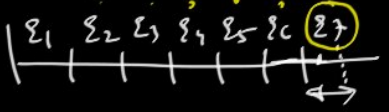
1 : Page is refered atleast once during " " ;

LRU

P	f	V/I	ToL	R
0	a	1	5	1
1	b	1	3	1
2	-	0	-	-
3	c	1	0	1
4	d	1	2	1

Page-Table

All Ref. bits are cleared to zero @ the end of Epoch



b) Additional Ref. bits :

C.E

$R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8$

- P_i : 1 1 1 0 1 0 1 1
- P_j : 1 1 0 0 1 1 1 1
- P_k : 1 1 1 1 1 1 1 1

P_x

ovhd

8 SL ops

3) Second chance / clock Algo:

Criteria: (T·O·L+R)

P	f	V/I	TOL	R	M
→ 0	a	1	5	0	1
1	b	1	3	1	0
(2)	–	0	-	-	
✓ 3	c	1	0	0	1
4	d	1	2	1	0

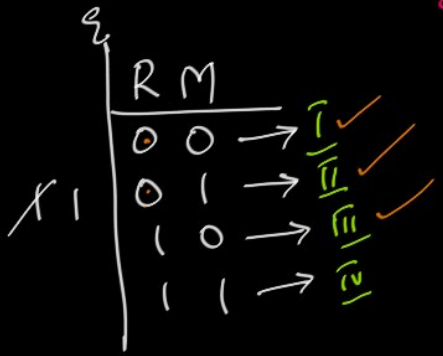
Page-Table

FIFO - Based

{ R: 1
Second-chance: 4 }

4) Enhanced Second chance / NOT Recently Used (NRU)

Criteria: R+M (Modified bit / dirty) ← 0 – clean, 1 – dirty



Thrashing : { Excessive/High paging activity }

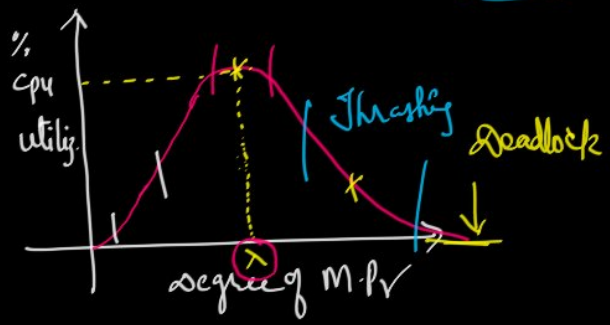
Paging activity = { The act of Loading & Saving the pages in the event of Page fault }

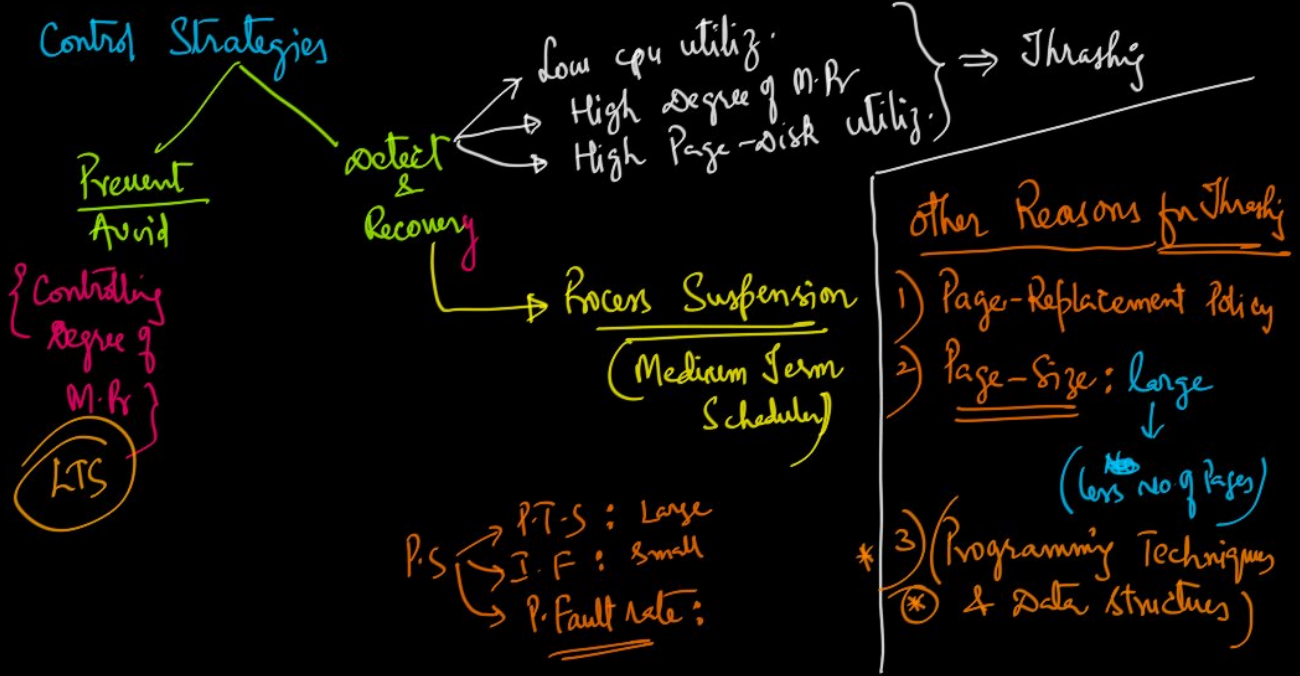
→ High Page fault rate = Thrashing [like deadlock is undesirable]
it Leads under utiliz. of cpu
drop in throughput

→ Reasons for Thrashing :

Primary { 1) Lack of frames (Memory)
2) High degree of M.Pr

1GB → 50|55|100




Control Strategies
Prevent
Avoid
{ Controlling Degree of M.Pr }
LTS
Detect & Recovery
Low cpu utiliz.
High Degree of M.Pr
High Page-Disk utiliz.
⇒ Thrashing
Process Suspension
(Medium Term Scheduler)
P.S
P.T.S : Large
I.F : small
P. Fault rate :
Other Reasons for Thrashing
1) Page-Replacement Policy
2) Page-Size : Large
(less No. of Pages)
* 3) (Programming Techniques
(*) & Data Structures)

Programming Techniques & Data Structures : Storage Structures : RMO / CMO

Case-Study-I

09/07/2021

P.S = 128 W ; Npages = 128 ; P.D.P ; FIFO

```
int A[1..128, 1..128];
```

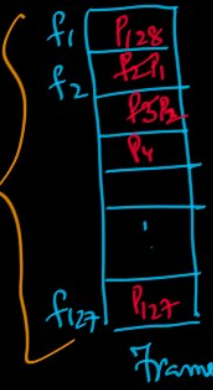
No. of Page-faults

I:
```
for i ← 1 to 128
    for j ← 1 to 128
        A[j,i] = 1;
```

$(128)^2 = 2^{14} = 16K$

II.
```
for i ← 1 to 128
    for j ← 1 to 128
        A[i,j] = 1;
```

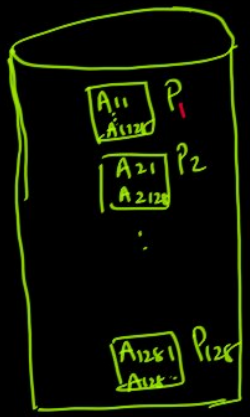
(128) obey L.O.R <Loc. of Reference>



128 frames very low

Frames

1 frame



Disk

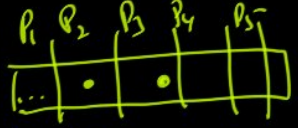
⇒ If the matrix is stored in Memory in R.M.O, then try to access in RMO only.

Case-Study-II :

Linear Search vs Binary Search [for very large-array]

→ Binary Search may generate more P.F's

→ Every subsequent Comparison may lead to P.F



obey Locality of reference

" Programs are said to obey Locality of reference, iff they make references to Pages present in Memory "

Case Study - III :

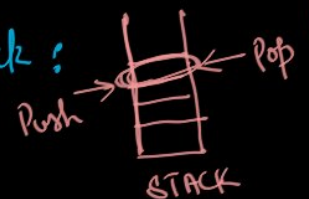
Linked list vs Array : → Locality

(NCG) CG

Case Study - IV :

usage of Stack :

- Good -



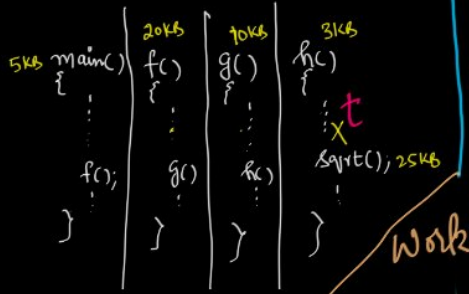
Case Study : Hash-Tables : Bad

Working Set Strategy : { helps control thrashing & utilize Memory effectively }

L is based on Principle of L.O.R

Prog-Size = 63 KB ; P.S = 1 KB ; N = 63 pages

50% rule



Ref-Str : 3, 8, 12, 7, 8, 10, 12, 22, 24, 23, 24, 22, 38, 40, 52, 60, 51, 50, 52, 40, 38, 41, ... 30

R_t

Δ=10 Δ=10 t

Working Set-window = { Set of unique Pages referred during the past 'Δ' references } Δ = integer = guess

(WSW; t)

WSW_i^t (Δ=10) = { 38, 40, 41, 50, 51, 52, 60 }

$WSWS_i^t$ = '7' (demand for frames)

WSW is a dynamic moving window with time

Δ : Success

Small → Inc. P.F's

large → ineffective utiliz. of Mem.

Framework:

→ n: no. of Process

→ S_i: WSWS$_i$: demand of P_i in frames

→ 'S': Total demand of all Process

$$S = \sum_{i=1}^{n} S_i$$

→ Available Frames: M

Scenario 1: M = S : No Thrashing

Scenario 2: M > S : No Thrashing (Inc. degree of M-Pr)

Scenario 3: M < S : Thrashing

↳ apply Process Suspension (Swap-out)

File System & Device Management:

(Visible part of OS)

Appl.

CPU <Electronic>

H/W

Disk <Electro-Mechanical>

File Manager (OS)

Device Driver (vendor)

Controller Card <SCSI; SATA; IDE>

Two - Movements

Linear

Angular

(H/D) Disk characterization <C.O>

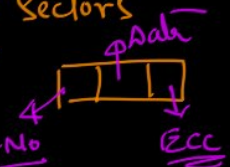
Physical Geometry

→ Disk is a collection of Platters

→ Each platter has 2 surfaces

→ Surface is divided into Tracks

→ Track is divided into Sectors



R/W operation time: unit of Tranfer is sector

: Seek_time (S.T) + Rotational Latency (L.T) + Transfer Time (TT)

S.T: ms (More)

L.T: $\left(\frac{R}{2}\right)$ ms

TT: $\left(\frac{X \cdot R}{Y}\right)$ ms (less)

R = Rotation_time (rpm)

3600 r — 60 s

1 r — ?

$\frac{60}{3600}$ s = 16.6 ms

Sector_Size = 'X' Bytes

Track_Size = 'Y' Bytes

Rot_time = R

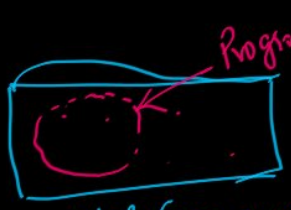
Y By — R ms

X By — ?

03. How long does it take to load a 64 Kbytes Program from a disk whose Average Seek time is 30 ms Rotation time is 20 ms, Track Size is 32 Kbytes, Page Size is 4 Kbytes. Assume that Pages of the Program are distributed randomly around the disk. What will be the % saving in time if 50% of the Pages of program are Contiguous?

Time_to_load Program =
(Time-to-Load a Page) * N
$= (S \cdot T + L \cdot T + T \cdot T)$
$= (30 + 10 + 2.5) * 16$
$= 42.5 * 16 = 680\ ms = 0.68s$

T.T =>
32KB – 20ms
4KB – ?
$\frac{20}{8} = 2.5ms$



Disk (ST: 30ms
RT: 20ms
TS: 32KB)
PS: 4KB

N = 16 pages
8 Pages are CG

NCG

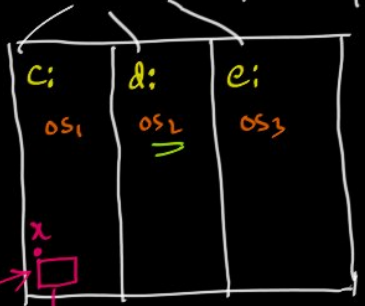
Time_to_load_Prg = $8 * (S\cdot T + L\cdot T + T\cdot T) + (S\cdot T + L\cdot T + T\cdot T \times 8)$
$= (8 * 42.5) + (30 + 10 + 20)$
$= 400ms = 0.4s$

% Saving = $\frac{0.28}{0.68} = \frac{28}{68} = (41.1\%)$

Logical Structure of disk: (Formatting Process)

(FAT32; Ext4; NTFS; ZFS; UFS ...)

Partitions / volumes / mini disk : fdisk / df



(Master Boot record) → MBR ✓

Multi-Boot System

Partition Table

Boot Loader

Partitions

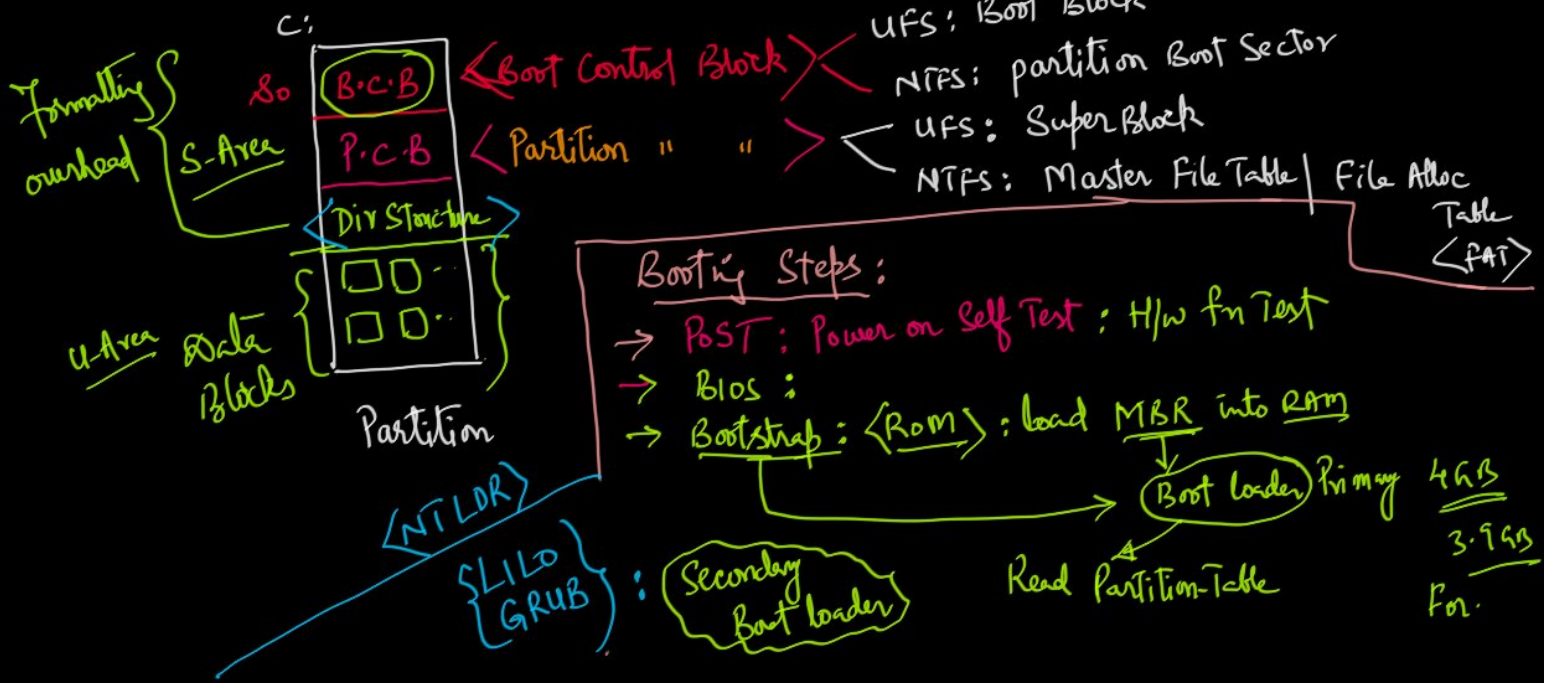
- Primary (Bootable) (OS + Data)
- Extended Non-Bootable (Data)

→ Every Partition must be formatted with some File System.

(putting infrastructure onto disk)

{Generic way} (R.R.t Priority)

Partition structure due to formatting; with a File System:



File System Interface :

→ Concept of File : { Collection of logically related records of an entity }

File as an A.D.T : < Defn, Repr/structure, operations, Attributes >

operations

- create
- open
- R/W/Tr/Modify
- seek
- close
- cp/Ren/MOV
- delete

Command API

Raw Structure / Flat
< series of Bytes >

Record Structure
< series of records >

↓

Hierarchical Structure
< B-Trees ... >

Attributes: < Name, Id, Type, owner, Size, Mode, Date & Time, permissions, + Blocks-in
Gen-Attribs | location/path | Loc-Attribs >

F.C.B

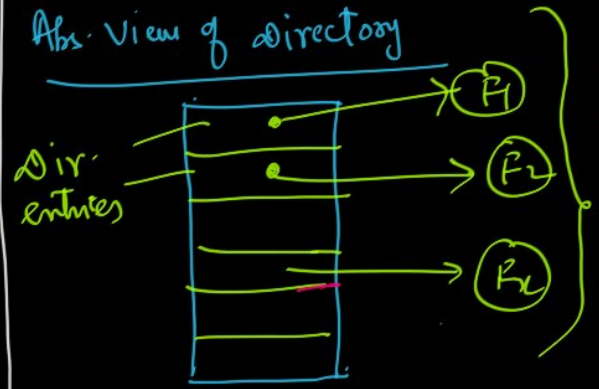
<File Control Block>

U.F.S
I-Node

NTFS | FAT32
(directory entry)

Directory: { is a special file; Metadata of files }
Folder

Abs. View of directory



Dir. opns
→ list
→ traverse

** File System Implementation Issues :

a) Allocation Methods :

Data Blk } DB
Disk "

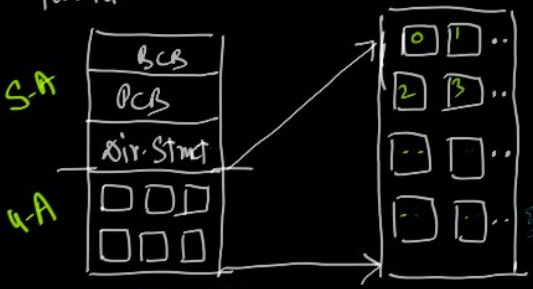
{ → Synch. Mech.
→ M.L.P + V.M
→ F.S.I }

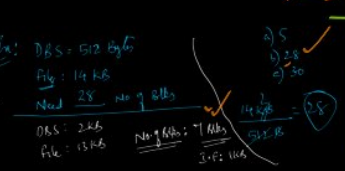
{File System}

DBA : Disk Block Address (bits)

DBS : Disk Block Size (Bytes)

Partition





"DB" is a unit of allocation to files, directories or any Data Structure that gets stored on disk

Ex: DBA : 16 bits ; DBS : 1 KB

→ Max. File Size = Disk-Size

Disk-Size : $2^{16} \times 1KB = 64MB$

Disk Size : $2^{DBA}_{Max} \times DBS$ By

2M

File Allocation Methods :

(i) Contiguous Allocation Methods :

if a File Needs 'K' blocks, then Contigious 'K' blocks of Disk are allocated.

Gen Attribs		Loc. Attributes	
File Name	....	Start D&A	No. of Blks
KK.C	....	2	4
AK.C	....	7	5

Directory Structure

11/7/2021

Performance Issues :

1) Int. Fragmentation : ✓
   <last Block>
2) External Fragmentation : ✓
   <Defrag : Heavy x Time>
3) Type of access : Both
   (Faster) (Seq & Random)
4) Increasing existing : Inflexible File Size

5 of 6

5
10
x

2) Linked Allocation/NCG: (DOS/WIN)

Gen Attributes		Loc. Attributes	
File Name	....	Start DBA	end DBA
KK.C	....	3	11

end-DBA

- (To Know the Consistency of file access)
- (Inc. existing file size)

Perf. Issues:

1) I. Frag: ✓
A2) Ext Frag: X
3) Type of access: only Sequential (Slow)
A4) Inc. existing file: Flexible
5) links of file Consume Disk Space
6) ptrs (links) are vulnerable
7) If link breaks file gets truncated.

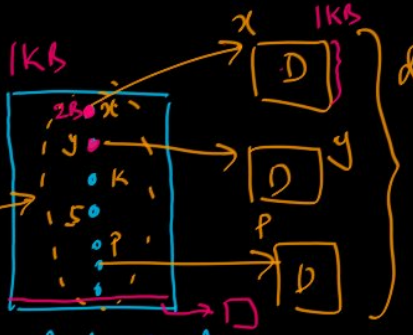
3) Indexed Allocation : <CA+NCA> : Each File is associated with an Index Block. Index Block is one of the IBlock of Disk; Index Block contains addresses of Data Blocks in use by the File.

(UNIX/LINUX)

Gen Attribs		Loc Attribs
File	...	I-block
K.C	...	7

→ IF: ✓
→ EF: X
→ Inc. File Size: Flexi
→ Type Access: Both

Address (DBA) of the Data Blocks in use by File



No. of Addrs in IB = 1KB / 2B = 512

Q: DBA: 16bits; DBS: 1KB;
: Max file size with Index Block : 512KB ✓

Case Studies :

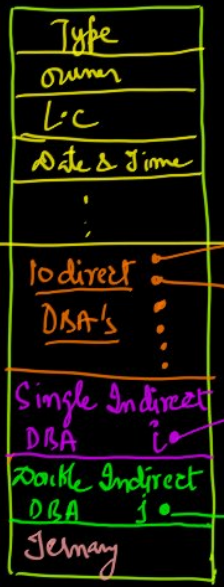
I: UNIX / LINUX : Directory Implementation :

Each file is associated with an I-Node : I-Node Contain Attributes of file

$ ls -i

File Name	I-Node
KK.c	23

UNIX Directory



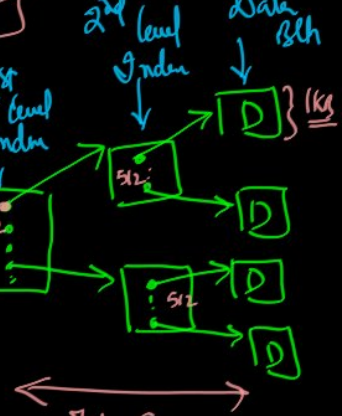
DBA : 16 bits
DBS : 1KB

Max FS : 64MB

Is this File Size really possible = 64MB

Max File Size : 10KB + 512KB + 256MB

{ : 256.522 MB }



$2^{28} = (256\text{MB}) = 512 \times 512\text{KB} = 2^9 \times 2^9 \times 2^{10}$

11. DOS/WINDOWS :

Gen-Attributes			Loc-Attributes
FileName	- -	- - -	First DBA
K·C		- - -	3

Dir. entry structure $\langle 3, x, 0, K \rangle$

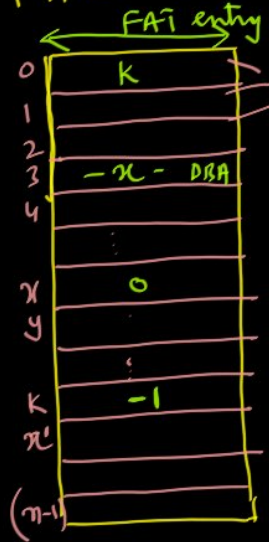
→ No. of entries in FAT = No of Blocks on Disk

→ FAT entry Contains Addres of Next DB in use

→ ∴ unless otherwise $\langle$FAT entry -Size = DBA$\rangle$

FAT - 8, 12, 16, 32

FAT = File Alloc Table (MFT)



(Tabular linked Alloc.)

05. Consider a Unix I-node structure that has 8 direct Disk Block Addresses and 3 Indirect Disk Block Addresses, namely Single, Double & Triple. Disk Block Size is 1Kbytes & each Block can hold 128 Disk Block Addresses.

Calculate (i) Maximum File Size with this I-Node Structure?

(ii) Size of Disk Block Address?

(iii) Is this File Size possible over the given Disk?

File Size: $\underbrace{8*1KB}_{\text{Direct}} + \underbrace{128*1KB}_{\text{S.I}} + \underbrace{128*128*1KB}_{\text{D.I}} + \underbrace{128*128*128*1KB}_{\text{T.I}}$

8KB, 128KB, 2^{21} = 16MB, 2GB

136KB = 0.136MB

$\Rightarrow$ 16.136MB

$\Rightarrow$ 0.016136GB

$\Rightarrow$ (2.016) GB

(2^{31}) By

Max. Size $= 2^{64} \times 2 = 2^{74}$ B

DBA: 1KB, 128, DB

$\frac{1KB}{128}$ = DBA

$\frac{2^{10}}{2^7} = 2^3 = 8$

DBA: 8 Bytes = 64 bits

07. A File System with 300 G Byte Disk uses a File descriptor with 8 Direct Block Addresses, 1 Indirect Block Address and 1 Doubly Indirect Block Address. The size of each Disk Block is 128 Bytes and the size of each Disk Block Address is 8 Bytes. The maximum possible File Size in this file System is

(a) 3 K Bytes
(b) 35 K Bytes
(c) 280 K Bytes
(d) Dependent on the size of the disk

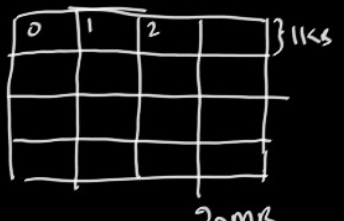
Disk Free Space Management:

Consider a disk : 20MB

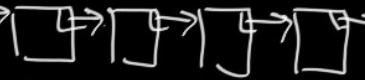
DBS : 1KB;

DBA : 16 bits; (2B)

No. of Blks = $\frac{20MB}{1KB} = 20K$



free

(i) Free linked list : 

link all free Blks to form a list.

(ii) Free list : Blocks of Disk Contains Addresses of Free Blocks.

No. of Blks of free list need to store all 20K free Blocks = 40

$\frac{20K}{512} = 40$

1 Blk – 512

? – 20K

(iii) Bit Map; Associate Binary bit with each Block;

Bit Map = Vector

0 — Free
1 — in use

20K free Blocks:

DBA: 16 bits

: 1 0 1 1 0 1 0 1 0 1
1 0 0 0 0 1 1 1 1 1
0 0 0 1 1 1 1 1 0 1
. . . . . .

Bit Map = 20K bits

Block = 1KByte
1K * 8 bits
= 8K bits

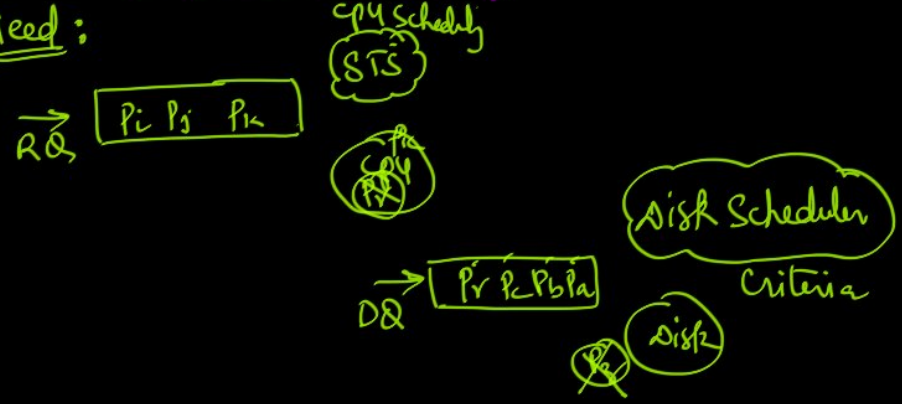
2) How many blocks are needed to store the Bit map?

1 Block = 8K bits
? — 20K bits

$\frac{20K}{8K} = 2.5 = \{3 \text{ Blks}\}$

Disk/Device Scheduling :

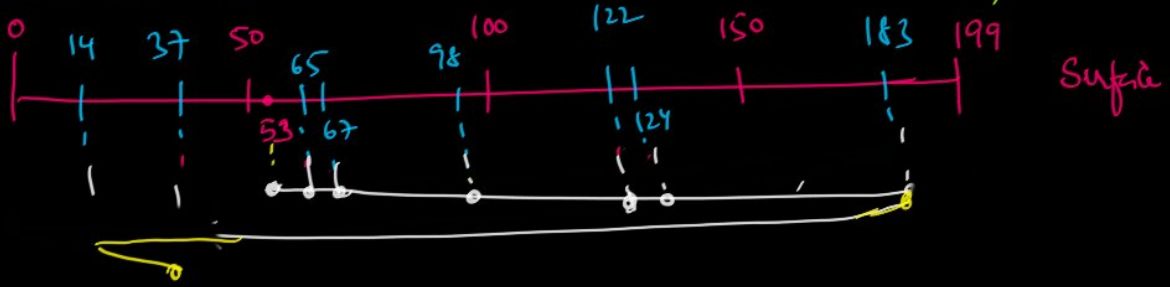
Need :



Generally the Process requests come in the form of Track No / cylinder No.

R/W Head @ : 53

Process Requests : < 98 183 37 122 14 124 65 67 >



(i) FCFS:
Total No. of Seeks : 640

(ii) Shortest Seek-Time First (SSTF) / Closest Track Next:
No. of Seeks : 236

(iii) SCAN/Elevator:
No. of Seeks: (199-53) + (199-14)
Starvation = 331

(iv) Look:
: (183-53) + (183-14)
= 299

(v) C-SCAN / Circular-SCAN
: (199-53) + (199-0) + (37-0) = 382

(vi) C-Look:
: (183-53) + (183-14) + (37-14)
: 322

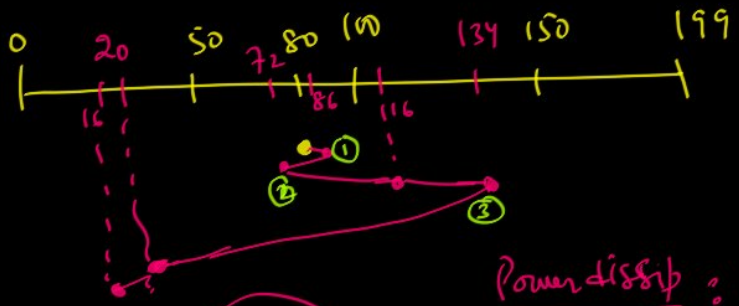
17. Consider a storage disk with 4 platters (numbered as 0,1,2 and 3), 200 cylinders (numbered as 0, 1, .., 199), and 256 sectors per track (numbered as 0, 1, , 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0] [212, 86, 3], [56, 116, 2], [118, 16, 1]

Currently the head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts.

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 requests using the Shortest Seek Time First disk scheduling algorithms is_____

Reqs: 72, 134, 20, 86, 116, 16



Power dissip / 100 Seeks : 20 mW

Total Seeks: 200

Power dissip: 40 mW

Revers : 3 x 15 = 45 mW

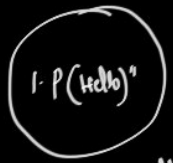
85 mW

Leftovers:

(i) Fork System Call : Case-Study:

Creating a new/child Process:

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

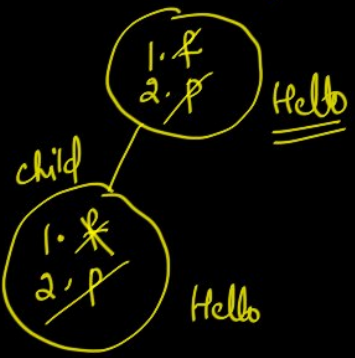


Hello

```
2) main()
   {
     1. fork();
     2. printf("Hello");
   }
```

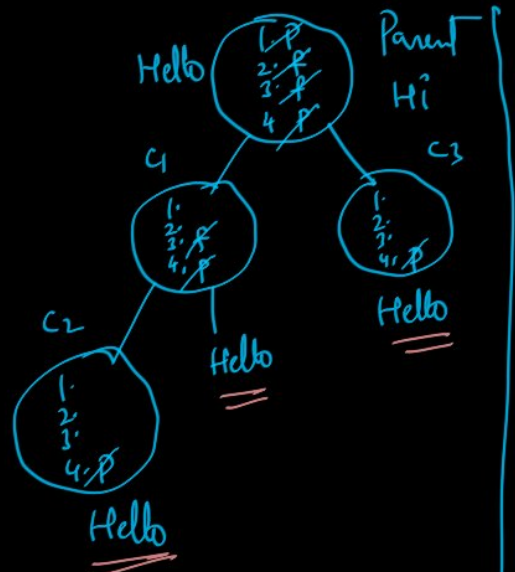
child is a replica of Parent, in terms of (Code/Text/Data)

Process (Parent



(Execution in child, starts from next statement after fork)

```
main()
{
  1. printf("Hi");
  2. fork();
  3. fork();
  4 printf("Hello");
}
```




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

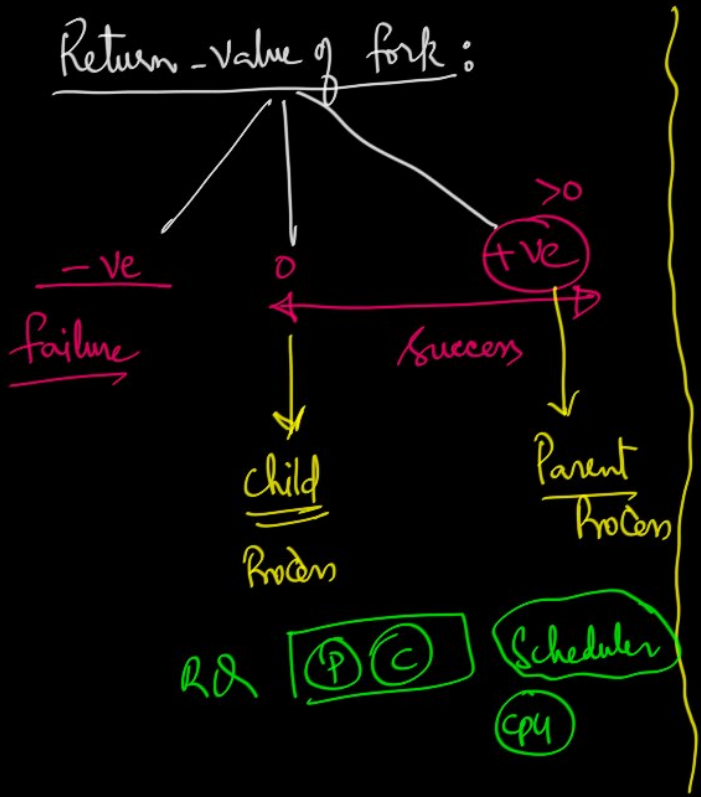
Total Processes : 8 → Hello

No. of New child : 7

No. of forks in Series	No. of children	Total Processes
1 f()	1	2
2 f()	3	4
3 f()	7	8
4 f()	15	16
⋮	⋮	⋮
n-f()	(2^n-1) ✓	2^n

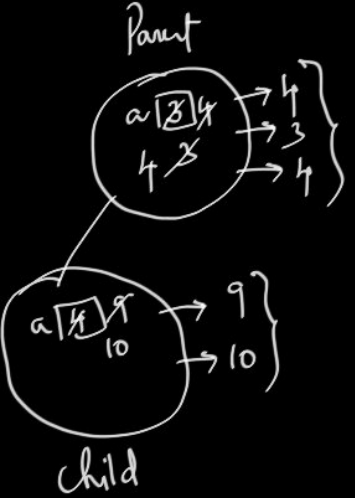
```
for (i=1; i<=n; ++i)
    fork();
```

No. of children $= (2^n-1)$




```
main()
{
   int id; // Parent //
   id = fork(); if (id<0) exit();
   if (id == 0)
   {
      // child //
   }
   else
   {
      // Parent //
   }
   // child + Parent //
}
```

① main()
{
int a=3;



a++;
print(a);
if (fork()==0)
{
a=a+5;
print(a);
++a;
}
else
{
a=a-1;
print(a);
++a;
}

print(a);
}

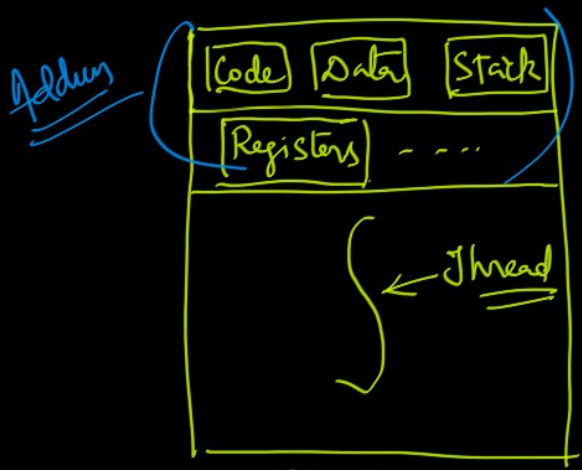
② main()
{
int i, n;
for(i=1; i<=n; ++i)
if (fork()==0);
print('*');
}

No. of times "*" get Printed = $(2^n - 1)$
" " " " " = (2^n)

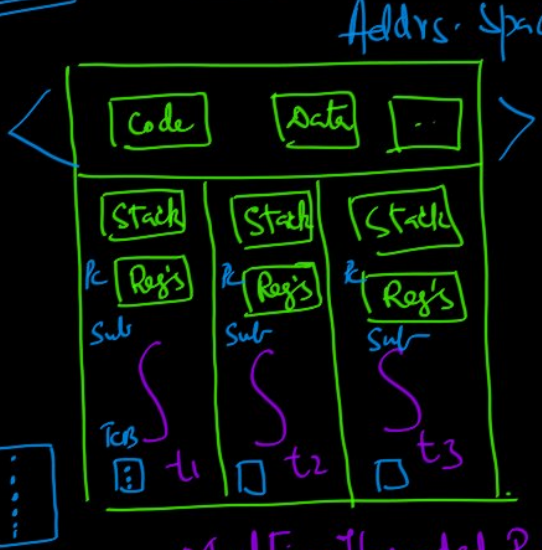
12/7/2024

Threads and Multithreading

: Thread is a light weight Process;

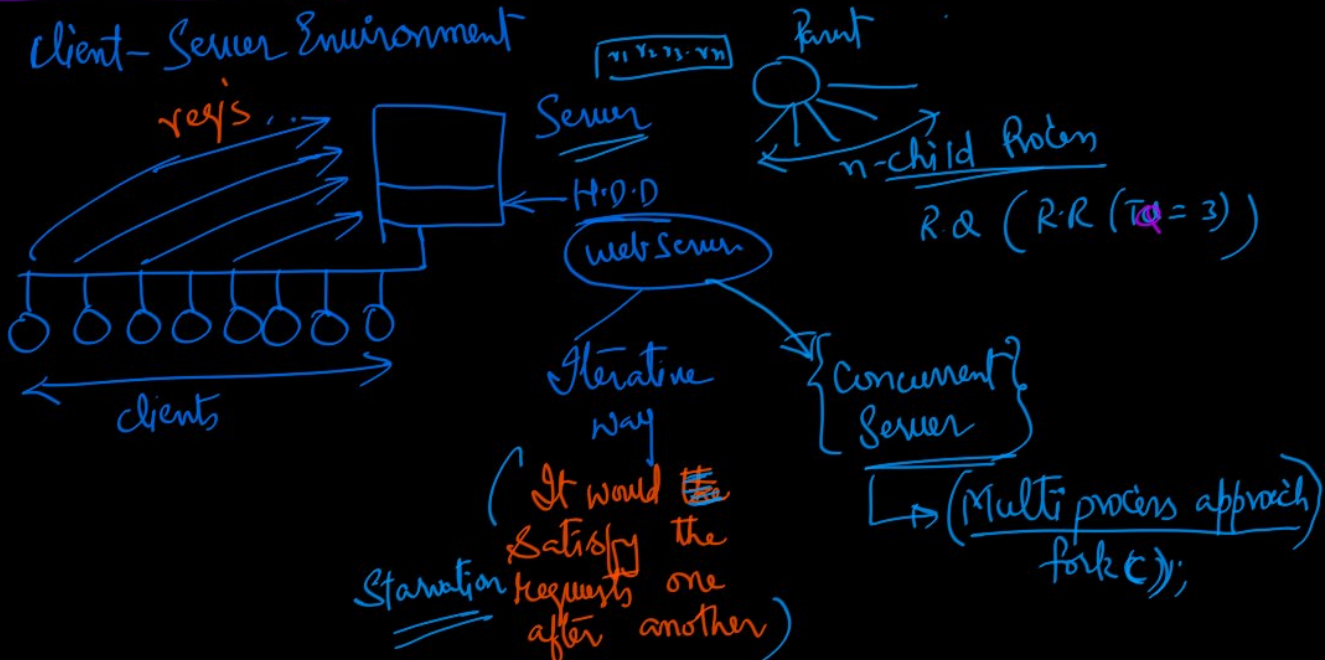


Single Threaded Process
Traditional / conventional



Multi-Threaded Process

Need for multi-Threading:



→ Thread like a process is a unit of cpu utilization
→ Thread " " " is an active entity
→ " " " " is a schedulable unit.

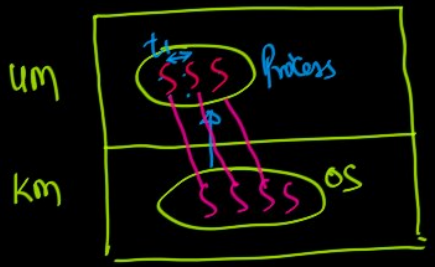
Benefits of Threads :

(i) Resource Sharing
(ii) Cost effective (Economical)
(iii) |TCB| < |PCB|
⇒ Less Context switch ovhd (Light-weight)
(iv) Improved Performance.
(v) Can achieve Computational Speedup by exploiting multi-core/cpu architectures

QuadCore (cpu)

(1 core is like one cpu)

Types of Threads < User-level-Threads (ULTs) : Threads created and managed @ the user level by spl. Thread Packages / Libraries.
Kernel-level " (KLTs)



→ Kernel-level-Threads (KLTs) : Win32
Modeling : 1-1, M-M
→ We can Schedule KLT independently.

(i) Benefits of ULTs :
→ Flexibility
→ ULT switching is Superfast
→ Examples: PThread, Java Threads, Green Threads
→ existence of Threads in Proces is Transparent to O.S

Drawback of ULT :
→ If one thread needs to carry out IO / Sys call then the whole process gets Blocked.

Monitors: is a collection of Procedures, variables and data structures that are all grouped together in a special kind of module/Package.

(Class (encapsulation / Abstraction))

→ Procedures that run outside the monitor cannot access monitor internal variables & data structures. They can activate monitor member functions.

→ Monitors have an important property that only one function/Process can be active in the Monitor.

→ Monitors include special condition variables that support wait() and signal() operations. They are used to meet the Synch. requirements.

```
monitor monitor name
{
   /* shared variable declarations */

   function P1 ( . . . ) {
      . . .
   }

   function P2 ( . . . ) {
      . . .
   }

         .
         .
         .
   function Pn ( . . . ) {
      . . .
   }

   initialization_code ( . . . ) {
      . . .
   }
}
```

Constructor

Figure 5.15 Syntax of a monitor.

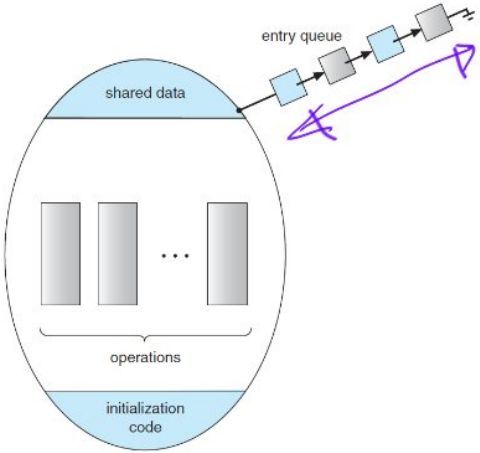


Figure 5.16 Schematic view of a monitor.

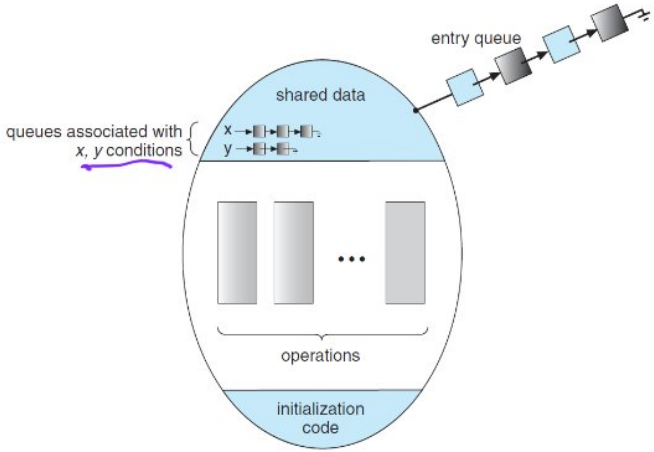


Figure 5.17 Monitor with condition variables.

5.8.2 Dining-Philosophers Solution Using Monitors

Next, we illustrate monitor concepts by presenting a deadlock-free solution to the dining-philosophers problem. This solution imposes the restriction that a philosopher may pick up her chopsticks only if both of them are available. To code this solution, we need to distinguish among three states in which we may find a philosopher. For this purpose, we introduce the following data structure:

```
enum {THINKING, HUNGRY, EATING} state[5];
```

Philosopher *i* can set the variable `state[i] = EATING` only if her two neighbors are not eating: (`state[(i+4) % 5] != EATING`) and (`state[(i+1) % 5] != EATING`).

L

R

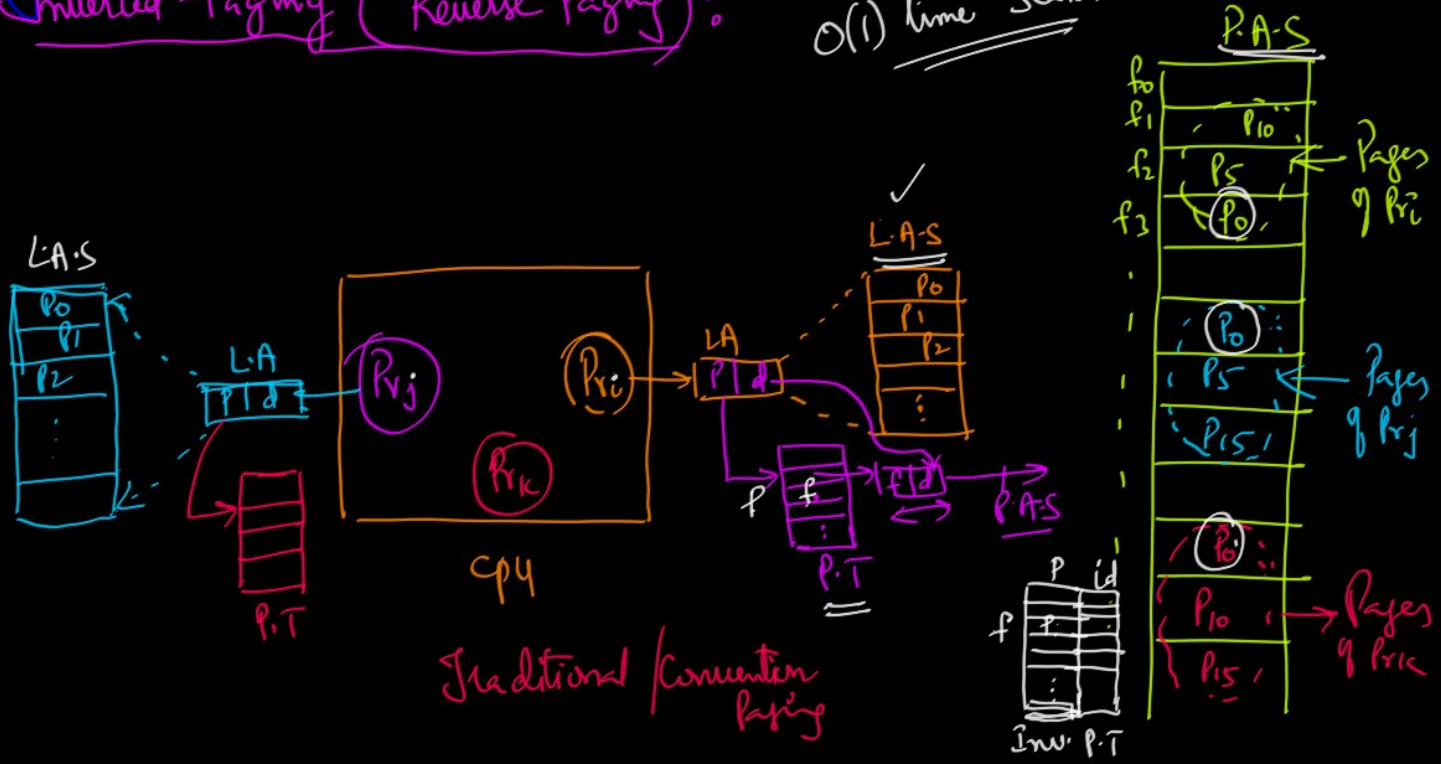
```
monitor DiningPhilosophers
{
   enum {THINKING, HUNGRY, EATING} state[5];
   condition self[5];

   void pickup(int i) {
      state[i] = HUNGRY;
      test(i);
      if (state[i] != EATING)
         self[i].wait();
   }

   void putdown(int i) {
      state[i] = THINKING;
      test((i + 4) % 5);
      test((i + 1) % 5);
   }

   void test(int i) {
      if ((state[(i + 4) % 5] != EATING) &&
       (state[i] == HUNGRY) &&
       (state[(i + 1) % 5] != EATING)) {
         state[i] = EATING;
         self[i].signal();
      }
   }

   initialization_code() {
      for (int i = 0; i < 5; i++)
         state[i] = THINKING;
    }
}
```

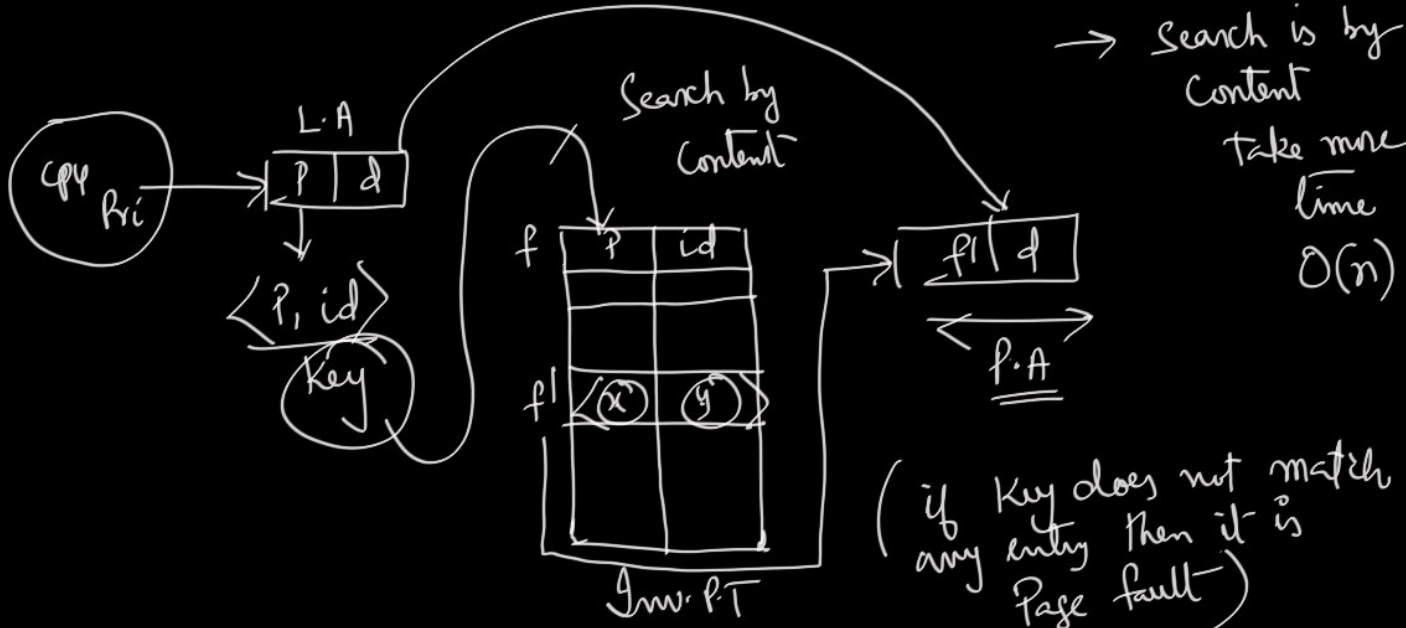
Figure 5.18 A monitor solution to the dining-philosopher problem.

```
DiningPhilosophers.pickup(i);
...
eat
...
DiningPhilosophers.putdown(i);
```

Inverted - Paging (Reverse Paging):

O(1) Time Search





→ All process Share one Inv P.T

→ Search is by Content take more time O(n)

(if Key does not match any entry then it is Page fault)

Q) Consider a System with LA : 33 bits; PA : 25 bits
P.S = 4 KB; e = 32 bits;
Calculate

(i) Conv. P.T. Size = $\left(2^{33}/2^{12}\right) * 2^2 = 2^{35-12} = 2^{23} = 8$ MB

(ii) Inverted P.T. Size = $\left(\frac{2^{25}}{2^{12}}\right) 2^2$ B = $2^{27-12} = 2^{15}$

= 32 KB

(Used by all Process)

one Process

Segmented - Paging : To overcome external fragmentation in Segmentation

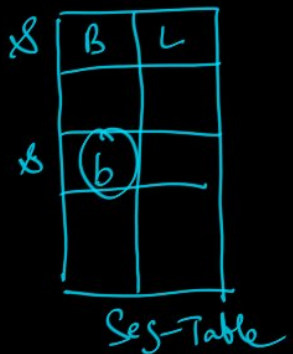
V.A = 34 bits : $\langle s, d \rangle = \langle 18, 16 \rangle$ bits

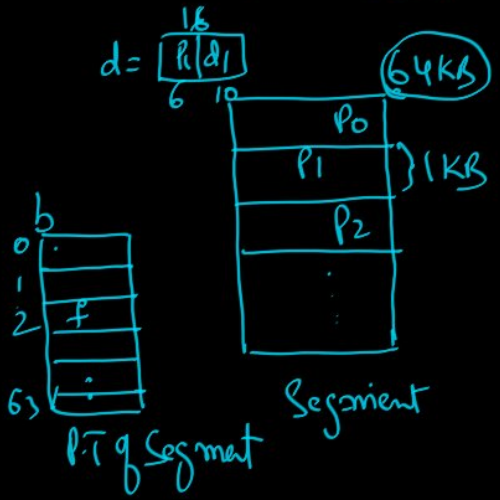
Max. Seg-Size : $2^{16} = 64KB$

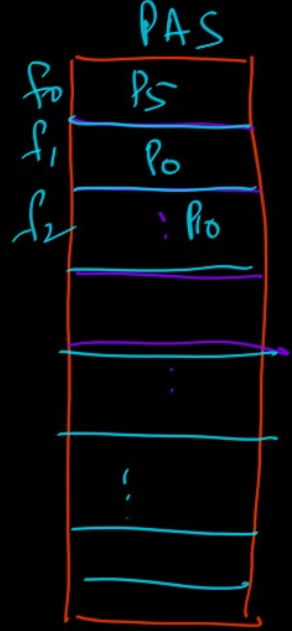
No. of Segms : $2^{18} = 256K$

B = Page Table Address

Let P.S = 1KB

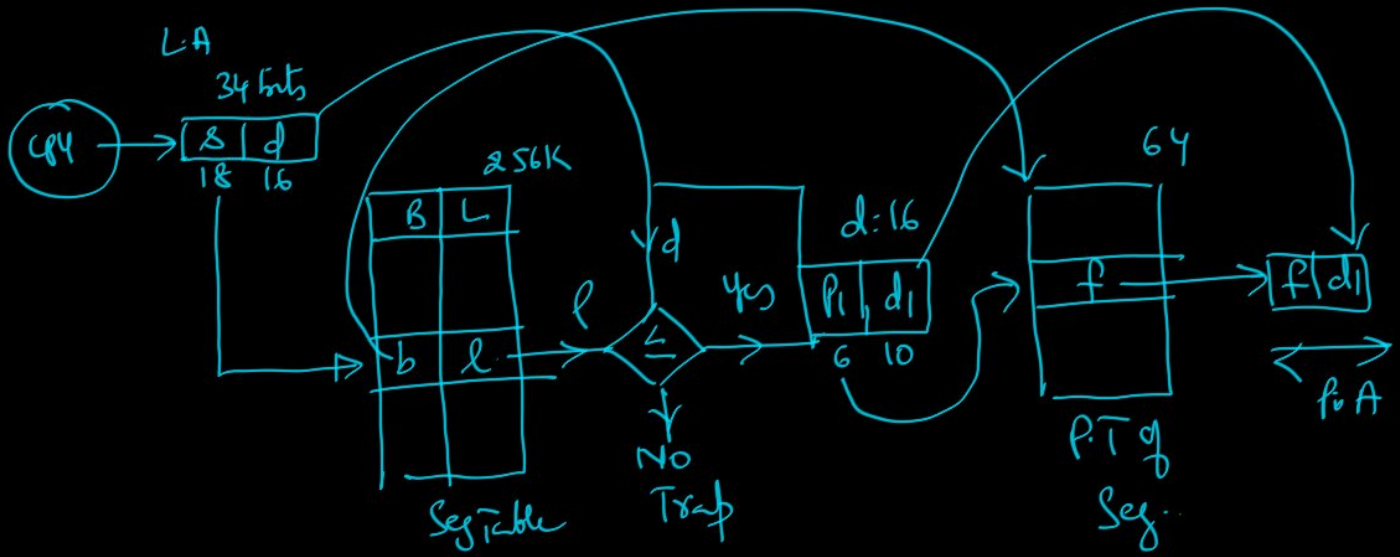






(When Seg-table become large then we can apply Paging)

{Segmented - Paging}



18. Consider a System using Segmented-Paging Architecture with V.A.S = P.A.S = 2^{16}Bytes. The VAS is divided into 8 equal sized non-overlapping Segments. Paging is applied on Segment, with a Page size being a power of 2 in Bytes. The Page Table Entry size of the Page Tables

of the Segment is 16 bits. What must be the Page Size of the Segment such that the Page Table of the Segment exactly fits in one frame of Memory.

$VAS = 2^{16}$ By

$Seg\text{-}Size = \frac{2^{16}}{8} = 2^{13} = 8KB$

Let $P.S = 2^x$ By

$F.S = 2^x$

$e = 16$ bits $= 2B$

$\therefore P.T.S = (2^{13-x}) \cdot 2B$

$= (2^{14-x})$ By

Seg : 8KB

$N = \frac{2^{13}}{2^x} = 2^{(13-x)}$

$\therefore 2^{14-x} = 2^x$

$\therefore x = 7$ $\therefore P.S = 2^7 = 128$ By

19. Consider a System using Segmented-Paging Architecture. Paging is applied on Segment. The System maintains a 256 entry Page-Table per Segment. The Page Size of Segment is 8Kbytes. The Virtual Address Space supports 2K Segments. Page Table Entry Size is 16 bits while the Segment Table entry is 32 bits in size.

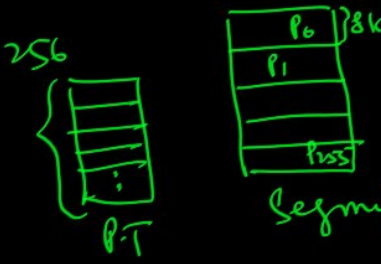
Calculate

a) Size of Virtual Address Space

b) Address Translation Space Overhead in Bytes.

c) The number of levels of Memory accesses required for Address Translation.

256 entry P.T per Seg: P.S = 8KB



Segment Size = 256 × 8KB

$= 2^8 \times 2^{13} = 2^{21}$

= 2MB

1) V.A.S = 2K * 2MB

= 4GB ✓

2) S.T + P.T

2K * 4B + 256 × 2B = 8KB + 512B

= (8.5 KB) ✓

(S.T + 2K * 512B)

3) 2-level.

(S.T + P.T + final access)

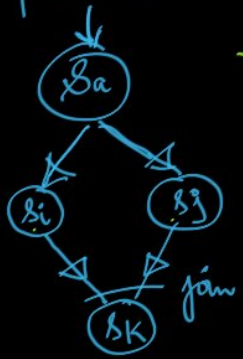
fork - Join Concurrency Construct

Count = 2

Syntax:

```
Sa;
fork L;
Si;
goto L1;
L: Sj;
L1: Join(Count);
Sk;
```

Execution of fork statement results in starting two computations concurrently. (i) @ label L (ii) immediately after fork



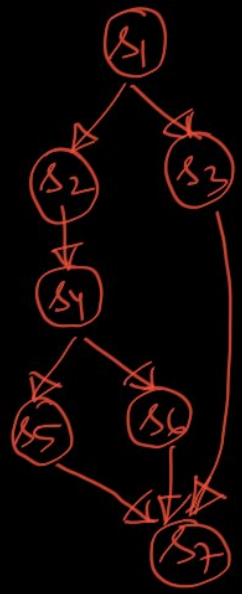
Any statement that joins 2/more computations, must have a join statement

Syntax: join (Count)

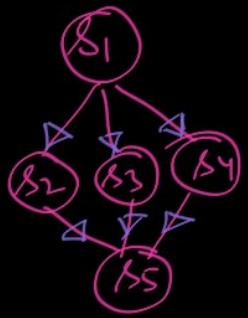
Count = No. of comp.'s it joins

```
join(int Count)
{   Count--;
    if (Count ≠ 0) exit;
    else return;
}
```

atomic

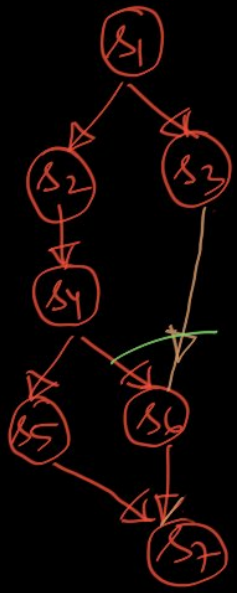



```
Count = 3;
  S1;
  fork L;
    S2;
    S4;
  fork K;
    S5;
    goto Z;
L:  S3;
    goto Z;
K:  S6;
Z:  Join(Count);
    S7;
```




```
Count = 3;
  S1;
  fork L
    S2
    goto K;
L:  fork Z
    S3;
    goto K;
Z:  S4;
K:  Join(Count);
    S5;
```

```
S1;
fork L
  fork Z
    S3;
L:  S2;
Z:  S4;
```




```
C1=2; C2=2;
S1;
fork L
   S2;
   S4;
fork K;
   S5;
   goto Z;
L: S3;
   ↓
K: Join(C1);
   S6;
   goto Z;
```

```
Z: Join(C2);
   S7;
```