Operating System :

NAME: CHANDRABHUSHAN MISHRA
ROLL NO: 12213139
CLASS: IT(B)-07

In every Computer System.
RAM
| OS | Running Program |

Intro :
- Software abstracting hardware
- Interface b/w user and hardware.
- set of utilities to simplify application development / execution
- Control program.
- Act like a government of Computer System.

Defn : It is a System S/w that manage & handle the hardware & software resources of a Computer System.
It provide interaction b/w user of Computers & Computer hardware.
It is responsible for managing & controlling all the activities & sharing of Computer resources.
It is low level s/w that include all fast the basic functions like memory management, Error detection, Communication (Inter-process Communication), Resource Allocation Accounting, Protection & security.

Services - User Interface, Program execution, I/o operation, File-system manipulation, Error detection.

Goals - Convenience (user-friendly), Efficiency, Portability Reliability, Scalability, Robustness.

Parts -

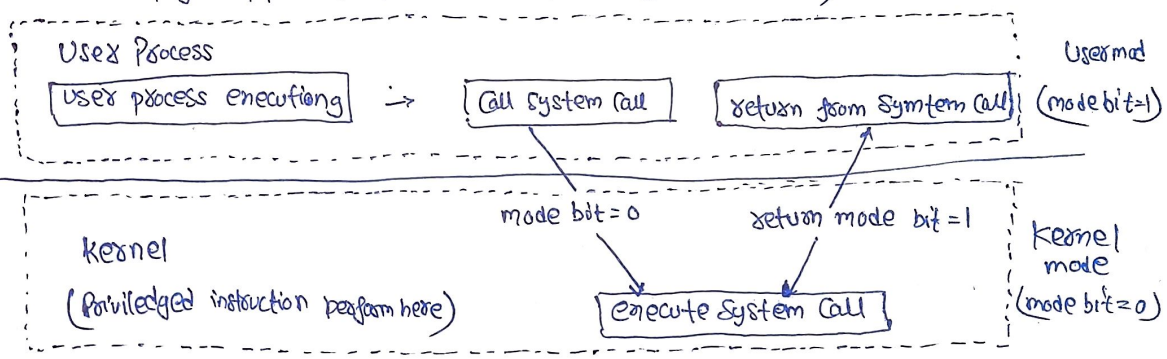
Interface < GUI / CLI (Common-Line-Interpreter)
all functionalities of OS.

System Call - To call function → to do any operation
A system call is a way for programs to interact with the OS

Types SC: Process Control, file manipulating, device manipulation, communication, info maintenance, protection

Dual Mode of operation - used to implement protection.
Process / user call → to perform any operation that process can't do → OS

& mode:
- User mode (mode bit = 1)
- Kernel / system / privileged / Supervisor mode (mode bit = 0)

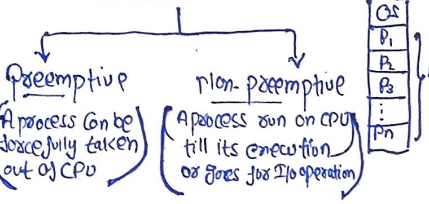


Types of OS :

① **Uni-programming OS** - OS allows only one process to reside in main memory.

Single process can't keep CPU & I/o devices busy simultaneously.
Not a good CPU utilization.

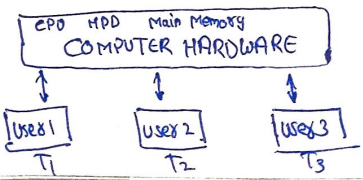
② **Multi-programming OS** - OS allows multiple processes to reside in main memory.



- Better CPU utilization than Uniprogramming.
- Degree of Multiprogramming.
  No. of running Program (Process) in main memory.
  Degree (Multiprogramming) ↑ → CPU utilization ↑

③ **Multi-tasking OS** - Extension of Multiprogramming OS in which processes ~~execute~~ upto certain limit execute in round-robin fashion
execute each process for limited time duration & then next process come & next process come and so on.

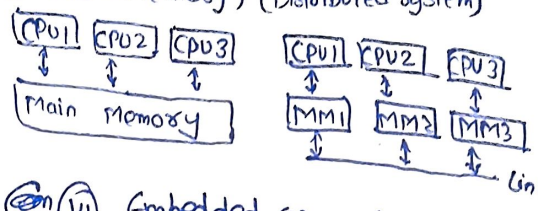
④ **Multi-user OS** -



This OS allows multiple users to access single system simultaneously
T - keyboard, mouse, display
Window → No multiuser OS
Linux, Unix → Yes.

(v) **Multi-processing OS** : This os is used in Computer System with multiple CPU

Tightly Coupled System (Shared-memory) | Loosely Coupled System (Distributed System)



Main Memory

(vi) **Embedded OS** : An os for embedded Computer Systems.
Designed os a specific purpose to increase functionality and reliability for achieving a specific task.
User interaction with os is minimum.

(vii) **Real-Time OS (RTOS)** : are used in environments where a large no. of event mostly external to the Computer system. must be accepted and processed in short time or within certain deadline.

- Hard Real Time (strict about deadline)
- Soft Real Time (Some Relaxation in deadline)

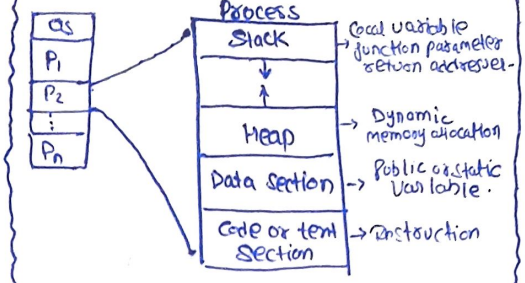
e.g. os used for Rocket Launching.
Every Process has → deadline.

(viii) **Hand-Held Device OS** : os used in hand-held device

Process : A program under execution (Code → set of instruction) (Program + Run time Activity ⟹ Process)
An instance of a program.
In term of CPU – Schedulable / Dispatchable unit
" " " OS – unit of execution
" " " OS – Flows of Control

Process As Data Structure :

- Definition — Code or Only Program
- Representation / Implementation — How process store in memory
- Operation
- Attribute

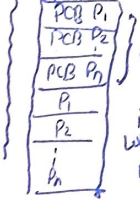


- Create (Resource Allocation)
- Schedule Run
- Wait Block
- Suspend Resume
- Terminate (Resource DeAllocation)

(OS will do all this operations on Process.)

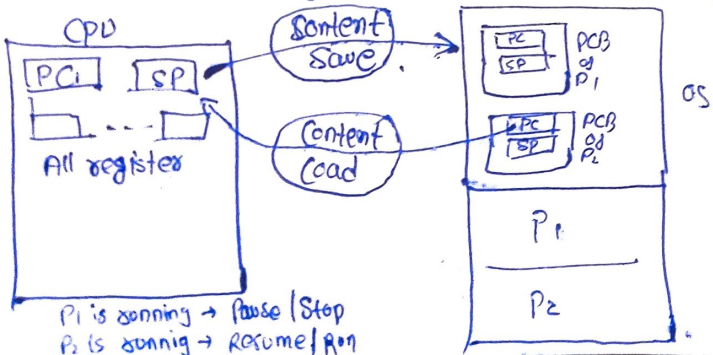
- Process ID (PID)
- PC (Program Counter)
- GPR (General purpose Register)
- List of Device
- Type
- Size
- Memory limit
- Priority
- State
- List of File

(Context) OS will maintain all the attribute into Data structure form called PCB
(Process Control Block or Process descriptor)



Content Switch : two thing
- A program in execution.
- It is done by os dispatcher in os → Program.

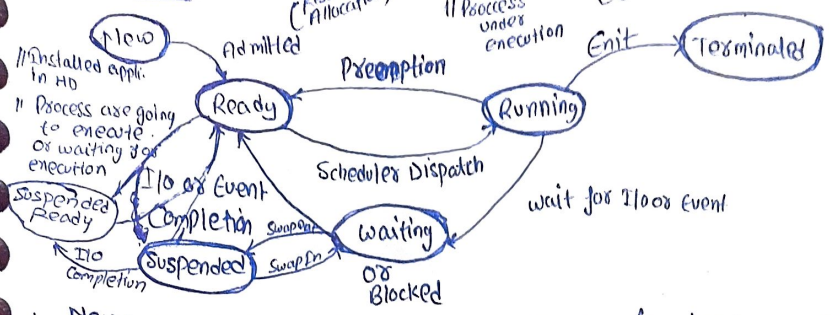
Content Save, Content Load



Notes : the Content of PCB of a process are collectively known as Content of a process

PCB were not accessed by anything (Program, Process, CPU) except os

Process State: Preemptive



- **New**: An installed process are known to be in new state
- **Ready**: All processes which are waiting to run on CPU are known to be in ready state
- **Running**: A process which is running on CPU has its state as running
- **Terminated**: A completed ~~comp~~ process has it state as terminated.
- **Blocked**: All process which are waiting for any I/O or event.

Process State transition:

- New to Ready : when a process is admitted by OS.
- Ready to Running : " " " " dispatched to CPU.
- Running to Terminated : " " " " completed
- " to Blocked : " " " goes for I/O or even. } By Process.
- Running to Ready : " " " is preempted.
- Blocked to Ready : " " " Complete I/O or even.

(these ~~Process~~ Transition is voluntary)

Q. n processes, m CPU | $m < n$. ⇒

		min	max
1	Running	0	m
2	Ready	0	n
3	Blocked	0	n

Process State : Non-Preemptive System.

Remove this transition from upperdig

Types of Process

① CPU Bound (use more CPU)
② I/O Bound (use more I/O)

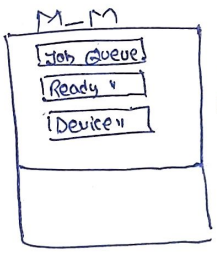
① **CPU Bound**: if the process in intensive in term of CPU operations.
② **IO Bound**: if the process in intensive in term of I/O operations

Notes: A process which has just terminate but has to relinquish its resource. is called zombie process

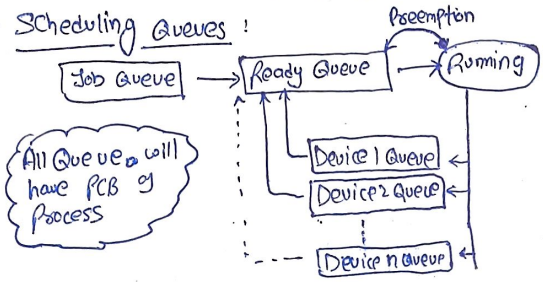
Process Scheduling: Better resource utilisation.

OS keeps the process into 3 Queue.

- Job Queue → All process which are in new state.
- Ready Queue → " " " " in Ready state.
- Device Queue → " " " " waiting for a specific device



* Scheduling Queues :



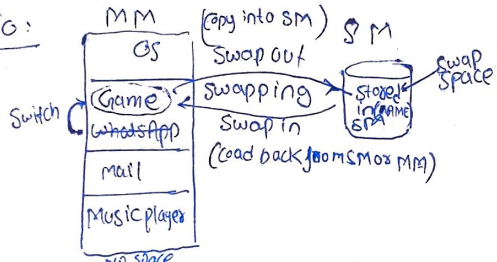
Scheduling Criteria.
CPU utilization.
throughput
TAT
WT
RT.
Fairness
Priorities

* Types of Schedulers :

Names are like that because of frequency of changes

1. Long-term Scheduler (Job) → From new state to ready state (Resource allocation)
2. Short-term " (CPU) → From ready to running (select one of all ready process)
3. Mid-term " (medium-term) → Does Swapping.

Scenario:



Swapping is known as rolling also if swapping is done based on priority.
State – Sospended. either from Ready or waiting

CPU Scheduling – Short-term Scheduler (one of the process selected from ready state to CPU)

Function: Make a selection

Goals: minimize waiting time & turn around time.
Maximize CPU Utilization (throughput)
Fairness.

Preemptive | Non-Preemptive

Scheduling Time:

(i) Arrival Time (AT): The time at which the process arrives in the system
(ii) Burst Time (BT): It amount of time for which process runs on CPU
(iii) Completion Time (CT): It time at which the process complete the execution
(iv) Turnaround Time (TAT): Time from arrival to complete.
(v) Waiting Time (WT): Time for waiting Process
(vi) Response Time (RT): Amount of time from arrival till first time Process gets the CPU
(vii) Deadline (D):
(viii) ~~Throughput~~:
(viii) Scheduling length (L): Total duration of time in which short-term Scheduler was Scheduling Process & process were executing
(ix) throughput (operation/sec): no. of process we have executed in per unit of time.

- TAT = CT − AT
  - TAT = AT + BT + CT
- WT = TAT − BT
- L = max(CT) − min(AT)
- throughput = n / L (n = no. process)

Assume many process to learn these Concept!

Note: Every Process has no any I/O operation (Assumption in all algorithms)

FCFS (First come first serve)

- Criteria: Arrival time (AT) {if two process has same arrival time} ↓ Tie breaker: Smaller process id first ← the time breaker (two process has same Arrival time then)
- Type: Non-preemptive.

Q.

Process	AT	BT	CT	TT	WT	RT	L	Throughput
P1	0	30	30	30	0	0	max(CT) − min(AT) = 45 − 0 = 45	3/45
P2	0	5	35	35	30	30		
P3	0	5	45	45	35	35		

(Same → non-preemptive)

Ready Queue = P1 P2 P3
Avg TAT = (30+35+45)/3 = 35
Avg WT = (0+30+35)/3 = 21.66.

Note: Gantt chart ⇒ time-line diagram, from when to when a process will be execute → describe by gantt chart
For non preemptive algo ⇒ RT of process = WT of that process. (RT is equal to WT)

Q.

Process	AT	BT	CT	TAT	WT	Scheduling length (L)	Throughput
P1	0	4	4	4	0	max(CT) − min(AT)	6/21
P2	1	2	6	5	3	21 − 0	
P3	2	3	9	7	4		
P4	3	5	14	11	6	21	0.2
P5	4	6	20	16	10		
P6	5	1	21	16	15		

(avg TAT = 59/6 = 9.833) (avg WT = 38/6 = 6.33)

Gantt chart: (P2 P3 P4 P5 arrive; P1 P6 P5 arrived)

P1 | P2 | P3 | P4 | P5 | P6
0 4 6 9 14 20 21

Ready Queue = P1 P2 P3 P4 P5 P6

Advantages:
1. Easy to implement
2. No complex logic
3. No starvation

Disadvantages:
1. No option for preemption
2. Convey effect makes the system slow

Q.

Process	AT	BT	CT	TAT	WT	RT
P1	4	4	8	4	0	0
P2	8	2	23	15	13	13
P3	6	3	14	8	5	5
P4	5	3	11	6	3	3
P5	2	1	3	1	0	0
P6	7	7	21	14	7	7

L = 23 − 2 = 21
throughput = 6/23

Gantt chart: (P2 P3 P4 P6 Arrived | Base on arrival time execute each processes)

P5 | P1 | P4 | P3 | P6 | P2
0 2 3 4 8 10 13 18 23

Time	Ready queue = P5, P1, P2, P3, P4, P6
0	
2	
3	

Convoy effect : If a large process is scheduled first then it slows down the system's performance this effect only occurs in FCFS. Problem in FCFS.

SJF (Shortest Job First)

- Criteria : Burst Time (smallest BT process first)
  Tie Breaker : FCFS. (sol^n)
- Type : Non - preemptive.

- **Advantages :**
  1. minimum avg. waiting time among non-preemptive scheduling
  2. Better throughput in continue run
- **Disadvantages :**
  1. No practical implementation b/c BT is not known in advance.
  2. No option of preemption
  3. Longer ~~preemption~~ processes may suffer from deadlock

Q.

Process	AT	BT	CT	TAT	WT
P_1	0	30	40	40	10
P_2	0	5	5	5	0
P_3	0	8	10	10	8

∵ On the Basis of BT - P_2 & P_3 are going to execute first $\xrightarrow[\text{is who's first}]{\text{But problem}}$ FCFS.

GANTT CHART

P_2	P_3	P_1

0 5 10 40

Q.

Process	AT	BT	CT	TAT	WT
P_1	0	6	23	23	17
P_2	0	3	3	3	0
P_3	1	4	12	11	7
P_4	2	2	6	4	2
P_5	3	1	4	1	0
P_6	4	5	17	13	8
P_7	6	2	8	2	0

Scheduling length (L) = max(CT) - min(AT)
= 23 - 0
= 23

throughput = $\frac{7}{23}$

Gantt chart :

P_2	P_5	P_4	P_7	P_3	P_6	P_1

0 3 4 6 8 12 17 23

Ready Queue

Time	
0	P_1 (P_2) ← less BT
3	$P_1 P_3, P_4$ (P_5)
4	P_1 P_3 (P_4) P_6.
6	P_1 P_3 P_6 (P_7)

Q.

Process	AT	BT	CT	~~WT~~	TAT
P_1	0	4	4	0	4
P_2	4	3	7	4	3
P_3	6	1	8	7	2
P_4	3	5	15	10	12
P_5	8	2.	10	8	2.

Gantt chart :

P_1	P_2	P_3	P_5	P_4

0 4 7 8 10 15

time	Ready Queue
0	(P_1)
4	(P_2) P_4
7	P_4 (P_3)
8	P_4 (P_5)

Problem in SJF : If there is a large process is Arrived only one & it is a at 0 then it must execute first. and we can't achieve the faster system as we want

SRTF (Shortest Remaining Time First).

If there is a current running Process among ~~arrived process~~ available processes already arrived if a new process arrive & that process having less BT then we will preempt this current running process & schedule new process.

- Criteria : BT
  Tie Breaker : FCFS
- Type : ~~Non~~- preemptive (AT→1CPU)

Advantages
1. among all Scheduling algo
2.

Disadvantage
1.
3. Starvation

Q. How many context switch happen across entire process execution within this = 7

Process	AT	BT	CT	TAT	WT	RT
P_1	0	6	17	17	11	0
P_2	1	4	9	8	4	0
P_3	2	1	3	1	0	0
P_4	3	2	5	2	0	0
P_5	4	1	6	2	1	1
P_6	5	3	12	7	4	4

	Ready Queue
at time 0	$P_1 = 6$ P_1
at time 1	$P_1 = 5$ ($P_2 = 4$)
at time 2	$P_1 = 5$ $P_2 = 3$ ($P_3 = 1$)
time 3	$P_1 = 5$ $P_2 = 3$ ($P_4 = 2$) → FCFS
time 4	$P_1 = 5$ $P_2 = 3$, ($P_4 = 1$, $P_5 = 1$)
time 5	$P_1 = 5$ $P_2 = 3$ ($P_5 = 1$)

① Preempt ② Preempt ③ ④ no Preempt ⑤ ⑥ ⑦ → Continue

P_1	P_2	P_3	P_4	P_4	P_5	P_2	P_6	P_1

0 1 2 3 4 5 6 9 12 17

(execute · execute · execute)

~~| P_1 | P_3 | P_5 | P_4 | P_6 | P_2 |~~

~~0 — 6 — 7 — 8 — 10 — 13 — 14~~

~~Time | Ready Queue~~
~~0 | P_1~~
~~6 | P_2 P_3 (BT) P_4 P_5 P_6.~~

FCFS & SJF Parallely.

$P_1 = 6/5$
$P_2 = 4/3$
~~$P_3 = 1$~~
$P_4 = 2/1$
$P_5 = 1$
$P_6 = 3$

Problem with SJF & SRTF.

1. Starvation - If shortest process are keep coming & one longer process are waiting ~~in~~ long (as indefinite wait)
2. No fairness -
3. Practical Implementation is not possible

HRRN (Highest Response Ratio Next)

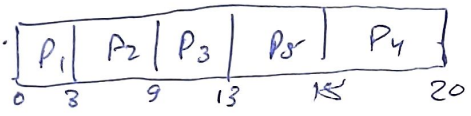
Object: Not only favours short Jobs but decrease the WT of longer job.
Criteria: Response Ratio
Tie-breaker: BT (if two process has same Response Ratio)
Type: Non-preemptive.

$$RR = \frac{WT + BT}{BT}$$

$RR \propto WT$
$RR \propto \frac{1}{BT}$ (Burst time = service time (s))

Q.

Process	AT	BT
P_1	0	3
P_2	2	6
P_3	4	4
P_4	6	5
P_5	8	2.



time	Ready Queue
0	P_1
3	P_2
9	P_3 P_4 P_5
13	P_4 P_5

(Whenever we have more the one process in Ready queue then we need algo.)

Cal @ time 9: $RR[P_3] = \frac{5+4}{4} = 2.25$ (it was waiting for long time)

$RR[P_4] = \frac{3+9}{5} = 1.3$

$RR[P_5] = \frac{1+2}{2} = 1.5$

Cal @ time 13: $RR[P_4] = \frac{7+5}{5} = 2.4$

$RR[P_5] = \frac{5+2}{2} = 3.5$ (waiting since long time)

Priority Based Scheduling:

- Advantages: ~~Better rest~~ Better response for real-time situation
- Disadvantage: Low priority Processes may suffer from Deadlock

(also suffer from Starvation)

If higher priority process keep arriving then low "indefinite" time may wait till

Criteria: Priority
Tiebreaker: FCFS
Type: < Non-preemptive / Preemptive

Priority Type < Smaller / Larger } mentioned in question

1. Static (fixed) 2. Dynamic (may increase or decrease)

Q.

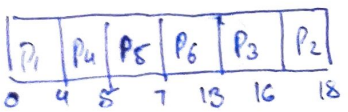
Process	AT	BT	Priority
P_1	0	4	4
P_2	1	2	5
P_3	2	3	6
P_4	3	1	10 (highest)
P_5	4	2	9
P_6	5	6	7

CT	TAT	WT
4	4	0
18	17	15
16	14	11
5	2	1
7	3	1
13	8	2
~~avg~~	avg=8	avg=5

CT	TAT	WT	RT
18	18	14	
15	14	12	
14	12	9	
4	1	0	
6	2	0	
12	7	1	
	avg = 9	avg = 6.	

→ for Non-Preemptive:

time	Ready Queue
0	P_1
4	P_2 P_3 P_4 P_5
5	P_2 P_3 P_5 P_6

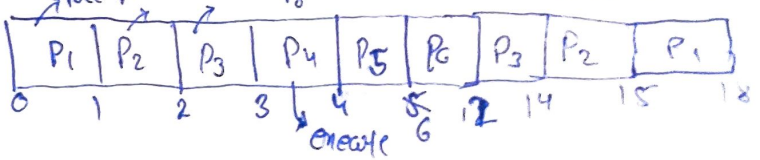


→ for Preemptive:

time	Ready Queue
0	P_1
1	P_1 P_2
2	P_1 P_2 P_3
3	P_1 P_2 P_3 P_4 P_5

4 Preempt

P	~~4~~ 3
P_2	~~2~~ 1
P_3	~~3~~ 2
P_4	T
P_5	~~2~~ X
P_6	~~6~~ 5



Soln of Starvation — Ageing (↑ Priority after certain unit of time)

Ageing is a predefine time period increase priority of all waiting Process by 1

Applicable only for dynamic priority system.

Round Robin :

Objective : Provide Interactiveness, Provide fairness.

Advantages :
(i) Fairness
(ii) Interactive for time sharing system
(iii) BT of process should not be known prior by

Criteria : AT + Q (Q = Time Quantum)

Tie-breaker : Process Id (smaller Id first)

Type : Preemptive.

(1).

	AT	BT	CT	TAT	WT	RT
P_1	0	3	9	9	6	0
P_2	0	6	17	17	11	2
P_3	0	4	13	13	9	4
P_4	0	8	18	18	13	6

Q = 2 avg: $\frac{31}{4}$ $\frac{39}{4}$

Gantt Chart :

P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4	P_2	P_4
0	2	4	6	8	9	11	13	15	17	18

time	Ready - Queue
0	~~P_1~~ P_2 P_3 P_4
2	~~P_2~~ P_3 P_4 P_1
4	P_3 P_4 P_1 P_2

~~$P_1 = 3 - 2$~~

$P_2 = 6 - 2 = 2$

~~$P_3 = 4 - 2$~~

$P_4 = 8 - 2 - 1$

No. of Content Switches = 9.

multitasking computing system $\xrightarrow{\text{Round Robin}}$ more effective

(2).

	AT	BT	CT	TAT	WT	RT
P_1	0	4	8	8	4	
P_2	1	5	20	19	14	
P_3	2	6	24	22	16	
P_4	3	3	19	16	13	
P_5	4	2	12	8	6	
P_6	8	4	22	17	13	
				$\frac{90}{6}$	$\frac{66}{6}$	

Q = 2

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_3	P_4	P_2	P_6	P_3	
0	2	4	6	8	10	12	14	16	18	19	20	22	24

time	Ready Queue
0	~~P_1~~
2	~~P_2~~ P_3 P_1
4	~~P_3~~ P_1 P_4 P_5 P_2
6	P_1 P_4 P_5 P_2 P_6 P_3

1st ~~$P_1 = 4 - 2$~~

2nd ~~$P_2 = 5 - 2$~~

$P_3 = 6 - 2$

3rd ~~$P_4 = 3$~~

2nd ~~$P_5 = 2$~~

5th ~~$P_6 = 4 - 2$~~

No. of Content Switches = 12

TRICK : for P_1 = 2 times CPU = $(\frac{BT}{Q})$

$P_2 = 3$, $P_3 = 3$, $P_4 = 2$, $P_5 = 1$, $P_6 = 2$ } 13 times CPU.

12 times content switching

What should be the Quantum Value

Very-2-small — CPU efficiency ↓ CPU spends more time in Context Switch & very less time in Program execution

Small — System Performance Interactive

Large — less Interactive

Very-2-large — Round-Robin degrades to FCFS.

Multilevel Queue (MLQ) Scheduling : ✗

multiple Ready Queue ↓ handle by OS.

- System Processes — Queue 1 ⇒ 60% time
- foreground Processes — Queue 2 ⇒ 30%
- Background Processes — Queue 3 ⇒ 10%

Priority high ↓ Priority low

Scheduling among Queue :

1. Fixed priority preemptive scheduling method → Suffer Starvation
2. Time slicing → no Starvation

Advantages of RR
1. All processes execute one by one so no starvation
2. Better interactiveness
3. BT is not required to be known in advance

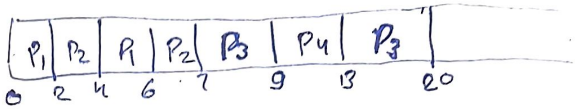
Disadvantage
1. avg. WT & TAT is more
2. Can degrade to FCFS.

Q. Queue1 : RR with Q=2 => Higher Priority.
Queue2 : FCFC

	AT	BT	Q
P_1	0	4	1
P_2	0	3	1
P_3	0	9	2
P_4	9	4	1

~~At time=0~~

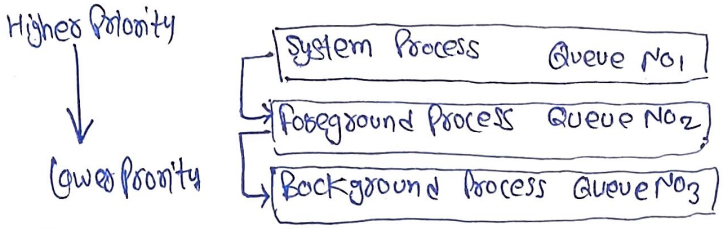
Time	Ready Queue 1	Ready Queue 2
at 0	P_1 P_2	P_3
at 9	P_4	P_3



1st ~~P1 = 1 2~~
2nd ~~P2 = 3 4~~
P3 = ~~2~~ 7
3rd ~~P4 = 4~~

Disadvantages :
1. Some process may starve for CPU if some higher priority queues are never becoming empty
2. It is inflexible in Nature

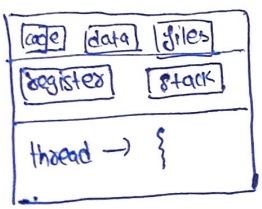
Multilevel feedback Queue (MLFQ) Scheduling.



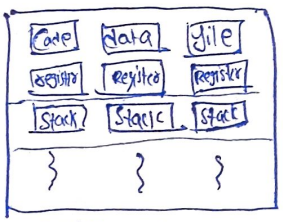
- Advantages : flexible ~~flexible~~
- Disadvantage : Some process may starve for CPU if some higher priority queues are never becoming empty
  - Soln → moving a process to higher level queue after certain duration.

Thread :
- User level thread :
- kernel level thread :

- Component of Process
  or
  Lightweight of Process
- Provide a way to improve application performance through parallelism.
- Shared Among Thread
  - Code section
  - Data section
  - OS Resource
  - Open file & Signals
- Unique for each thread
  - thread ID
  - register set
  - Stack
  - program counter
- Advantages
  1. Responsiveness
  2. Fastest Context Switch
  3. Resource Sharing
  4. Economy
  5. Communication
  6. utilization of multiprocess Architecture



Single thread process



Multi thread processes

Types:

User thread

- Multithreading in user process
- Create without kernel intervention
- Context switch is very fast
- If one thread is blocked, os blocked entire process
- Generic and can run on any os
- Faster to create and manage

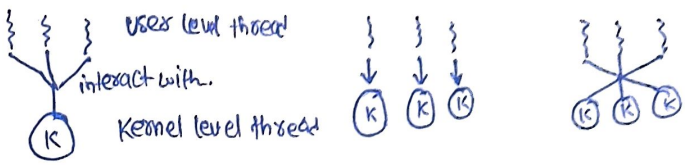
Kernel thread.

Multithreading is kernel process.
Kernel itself is multithreaded
Context switch is slow
Individual thread can be blocked.

specific to os
Slower to create and manage.

Multithreading Model.

1. Many to one Model
2. One-to-one Model
3. Many to Many Model



Types of Process.

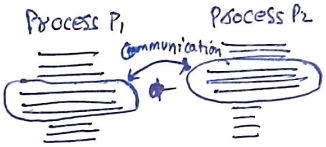
1. Independent – no any communication with any other process. → no synchronization required
2. Cooperating / coordinating / communicating
   ↳ can affect other process and affected by other process. → synchronization required.

Need of synchronization.

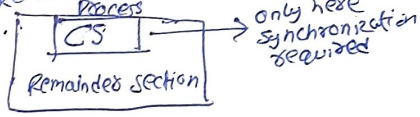
↳ In communicating process.
↳ Problem without synchronization: → Inconsistency, → Loss of data, → Deadlock.

Each instruction is not necessary to involve in communication.

Critical Section: It is a code segment where the shared variables can be accessed.



So Process has two type of section
↳ Critical section
↳ Remainder section

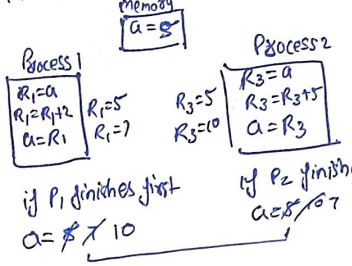


Race Condition:

Because of lack of synchronization

It is undesirable situation it occurs when the final result of concurrent process depends on the sequence in which the process complete their execution

e.g.



if P1 finishes first
a = 7 ✗ 10

if P2 finishes first
a = 10 ✗ 7

In both cases, result are diff.

Soln for Critical Section Problem:

Process
- Entry Section
- Critical Section
- Exit Section

Requirements of Critical section problem solution.

1. Mutual Exclusion: If one process is using C.S. then other process can't use C.S.
2. Progress: If no any process in CS & atleast one process wants to entry in C.S then it should be allow
3. Bounded waiting: waiting of process should be Bounded.

Soln: Using lock.

```
Boolean lock = false;
while (true)
{ while (lock);
  lock = true;
    CS
  lock = false
    RS
}
```

```
P1
while (true)
{ while (lock)
  lock = true;
    CS
  lock = false
    RS
}
```

- Does not satisfied Mutual Exclusion
  If a process preempted after while (lock) ; statement
- Progress is satisfied.

Soln 2: using turn.

```
int turn = 0
P1
while (true)
{ while (turn != 0)
    CS
  turn = 1
  RS
}
```

```
P2
while (true)
{ while (turn != 1)
    CS
  turn = 0
  RS;
}
```

→ Satisfied Mutual Exclusion
→ ~~satisfied~~ Progress. is not satisfied
→ Bounded waiting is here.

These two process will execute only in strict alternate manner.

Soln: Peterson's Soln

```
Boolean flag[2]  → flag[0] = false
                   flag[1] = false
int turn → P0 for priority.
P0
while(true){
  flag[0] = true;
  turn = 1
  while(flag[1] && turn == 1)
    CS;
  flag[0] = false
    RS;
}
```

```
P1
While (true) {
  flag[1] = true;
  turn = 0
While (flag[0] && turn == 0)
  CS;
  flag[1] = false;
  RS;
}.
```

→ ME is satisfied
→ Progress is satisfied.
→ Bounded waiting is satisfied.

Synchronization Hardware: Instruction support by CPU ⇒ Can be used to provide Synchronization
↓
Privilaged Instruction (executed in kernel mode)

1. Test AND Set()
2. Swap ()

TestAndSet () : Return the current value ylag and set it to true.

```
Boolean Lock = false.
Boolean TestAndSet(Boolean * tag)
{ boolean rv = * tag;
  * tag = True;
  Return rv;
}
↓
atomic
```

```
While (true)
{ while (Test And Set (& lock));
    CS
  lock = false;
}.
```

Busy Wailing : A process execute on CPU but can't go further because of CS Solution then process is known to be in busy waiting

Swap () :

```
Boolean key, lock = false:
Void Swap (Boolean *a, Boolean*b)
{
  boolean temp = *a
       *a = *b
       *b = temp;
}
↓
atomic
```

```
Po
while(true)
{ key = True;
  while (key == True)
    Swap (&lock, &key).
  key = True;
    CS
  lock = false;
    RS
}
```

No mutual Exclusion if preemption is happen

Synchronization Tool :

1. semaphore
2. Monitor

Semaphore : It is synchronization tool which are integer value which can be accessed using following functions only.

- wait() / P() / Degrade() / lower() / down() → Decrement upto 0 (atomic function)
- Signal() V() upgrade() / up → Increment.

Binary — Only two values 0 & 1

Counting — Unlimited domain of value (any integer)

```
wait(S)                    Signal(S)
{ while (S<=0)             {
    S--;                       S++;
}                          }
```

→ at Exit Section

At Entry Section ← atomic function.

no impact of half execution cycle preemption. it start from initially.

Binary Semaphore : It is used to implement the soln of critical section problem with multiple processes. ⇒ for mutual exclusion.

Counting Semaphore : It is used to control access to a resource that has multiple instances.

- CS Soln : S=1

```
P0                          P1
while (True)                while (True)
{ wait(S)                   { wait(S)
    CS.                         CS.
  Signal(S)                   Signal(S)
}.                          }.
```

Characteristics.

- used to provide Mutual exclusion.
- used to control access to resource.
- soln using semaphore can lead to have Deadlock
- " " " " " " " Starvation → indefinitely waiting
- " " " " busy waiting soln.
- semaphore may lead to a priority Inversion.
- semaphore are machine-independent.

Q. Consider a semaphore S initialized with value 1 Consider 10 processes $P_1, P_2 \dots P_{10}$ All processes have same code as given below but one process P_{10} has signal(S) in place of wait(S) if all process to be executed only one, then max. no. of process which can be in critical section together.

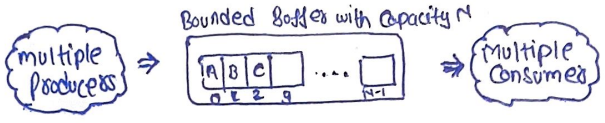
```
P0 .... P9.                 P10
while (True)                while (True)
{ wait(S)                   { signal(S)
    CS                          CS
  Signal(S)                   Signal(S)
}                           }
```

Ans let Assume P_{10} execute first for max. CS.

S = 1 → after getting Room → 2 → Now 3 process can execute together in CS. (P_{10} + other 2 P)

Ans if process to be execute multiple times executes P_{10} till S becomes equal to 10 ⇒ (10)

Bounded Buffer Soln :




```
Producer()                  Consumer()
{ wait(empty)               { wait(full)
// produce an item
  wait(mutex)                 wait(mutex)
// add item on buffer
  Signal(mutex)               signal(mutex)
  Signal(full)                Signal(empty)
}.                          }.
```

- Known as producer - Consumer problem also.
- Buffer is the shared resource b/w producer & consumer
- Producer must block if the Buffer is full.
- Consumer must block/stop if the Buffer is empty
- variable -

  Mutex : Binary semaphore to take locks on buffer (Mutual Exclusion)

  full : Counting semaphore to denote the no. of occupied slots in buffer

  empty : Counting semaphore to denote the no. of empty slots in buffer

Readers – Writers Problem: Consider a situation where we have a file shared b/w many people.

- If one of the people tries editing the file no other person should be reading or writing at the same time otherwise change will not be visible to him/her.
- However, if som person is reading the file then other may read it at the same time.

Soln:

- Variable:
  - mutex : Binary semaphore to provide mutual exclusion.
  - wrt : " " " restrict readers & writers if writing is going on
  - readcount : Integer variable denote no. of active readers.
- Initialization:
  - mutex : 1
  - wrt : 1
  - readcount : 0

Writer() Process.

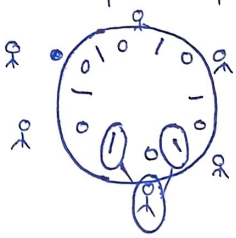
```
writer() {
{
   wait (wrt)
   Signal (wrt)
}
```

Reader() Process:

```
Reader() {.
{ wait(mutex)
  readcount++;
  if (readcount == 1)
    wait (wrt)
  Signal (mutex);
  // perform reading
  wait (mutex)
  readcount--;
  if (readcount == 0)
    Signal (wrt);
  Signal (mutex);
}.
```

Dining Philosopher problem:

- K philosphes seated around a circular table.
- there is a chopstick b/w each philosopher.
- A philosphes may eat if he can pick up the 2 chopsticks adjacent to him.
- One chopstick may be picked up by any one of its adjacent followes but not both.



philosopher = i

```
pick (chopstick[i])
pick (chopstick[(i+1) mod k)
// eat
release (chopstick [i])
release (chopstick[(i+1] mod k)
```

Soln using Semaphore :

Binary Semaphore array chopstick [5] = {1, 1, 1, 1, 1}

```
{wait (chopstick[i])
wait ( chopstick[(i+1) % 5])
// eat
signal (chopstick [i])
signal (chopstick (i+1) % 5)
}.
```

```
wait (chopstick [(i+1] %k)
wait (chopstick (i))
signal (chopstick [(i+1] %k)
signal (chopstick (i])
```

this soln may lead Deadlock.

→ Some way of the way to avoid deadlock as follows –

(i) there should be at most (k-1) philosphers on the table.

(ii) A philosophes should only be allowed to pick their chopstick if both are available at the same time.

(iii) One philosopher should pick the ~~left~~ right chopstick first and then ~~right~~ left chopstick next; while all others will pick the right on first then left one

Deadlock :

Operations on Resources : 3 (can be H/w or S/w)
1. Request : Process requested for a resource to OS when OS allocate the resource to process then process can use it.
2. Use : When use is completed then process release the resource.
3. Release :

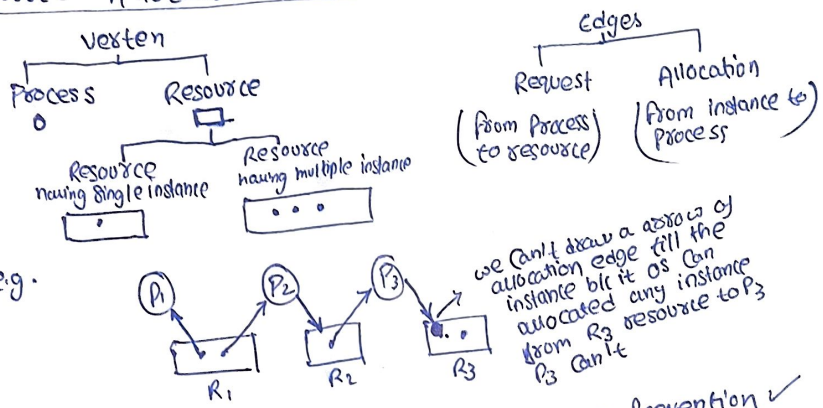
Defn : If two or more process are waiting for such an event which is never going to occure.

- Neccessary condⁿ for Deadlock :

Deadlock can occure only when all following condⁿ are satisfied : (All four should satisfied together then Deadlock will occure.)

1. Mutual Exclusion : If one process using a resource then other process can not use it.
2. Hold & wait : Each deadlocked process should hold atleast one resource & should wait for atleast one resource
3. No-preemption : No any forcefull preemption of resource.
4. Circular wait :

- Resource Allocation Graph : Directed Graph.



- Recovery from Deadlock :

1. Make sure Deadlock never occure. (Deadlock Prevention ✓ Deadlock Avoidence)
2. Allow deadlock, detect & recover
3. OS pretend that there is no any deadlock.

Deadlock prevention : Prevent any of four ~~neccary~~ neccessary condition to occure.
- ME
- H&W
- NoP
- Circular w.

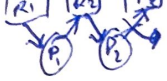
Practically Impossible

(i) How we will avoide or prevevent to ocure mutual exclusion.
- multiple resources available for simultaneous execution of the process
- make all processes independent

(ii) Prevent from Hold & wait.
- we make an arrangement such that process will either hold or wait but not together
1. If all resources are available then acquire or else just wait for all (Process may suffer from starvation)
2. If a process is trying to acquire a resource which is not available while holding some resource then process will release the allocated resources.
3. If a process holds resources but not using them there will be poor utilization of Resource.

(iii) No-preemption : 

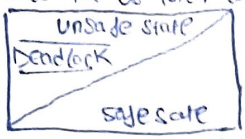
P_1 request for R_1 & R_1 is held by P_2. P_2 is also in wait for other process OS may preempt R_1 from P_2 and will give it to P_1



(iv) Circular wait :
1. All the resources have been give sequence no (unique) like this R_1, R_2, R_3
2. Any process while holding a resource R_i can request for R_j only when $j > i$
3. If a process is in " " " R_i & wants another resource R_i when $j < i$ then process will have to release R_i will have to acquire R_i first

Deadlock Avoidance:

- In deadlock avoidence the OS tries to keep system in safe state.



- In deadlock Avoidence the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system.
- It will be implemented using "Banker's Algorithm".

Banker's algo: The bankers algo is a resource allocation & deadlock avoidance algo that tests for safety.

Eg:

Process	Allocation	Max	Available	Need (Max − Allocation)
P_1	1	3	1 ~~5~~ 10	2
P_2	5	8	12	3
P_3	3	4		1
P_4	2	7		5
System ~~has~~ total	11			

= 12 resources

1. Execute P_3
2. P_1
3. P_2
4. P_4

All processes can execute hence ⇓ Safe State.

Sequence - Safe Sequence $< P_3\ P_1\ P_2\ P_4 >$

Can have multiple safe sequence.

For 4L for 4 year.

Plan	Allocated
C_1 4L	1L, 2L, 3L, 4L
C_2 5L	5L
C_3 6L	6L

✓ P_3 = 3 + 1 = 4 (Available) → Finish ④ → Pay back available 4.

✓ P_1 = ~~1~~ + 2 → available ⇒ Finish ⑤ → payback + 2 available.

P_2 = 5 + 3 → available → Finish ⑧ → Payback + 2 available = 10

P_4 = 2 + 5 ⑦ → Paybaybback + 5 ⇒ 12 available

②.

Process	Allocation A B C	Max A B C	Available A B C	Need (Max − Allocation)
P_0	0 1 0	7 5 3	~~3~~ 3 2	7 4 3
P_1	2 0 0	3 2 2	~~5 3 2~~	1 2 2
P_2	3 0 2	9 0 2	7 4 3	6 0 0
P_3	2 1 1	2 2 2	~~7 5 3~~	0 1 1
P_4	0 0 2	4 3 3	~~10 5 5~~	4 3 1

P_0 → Can't fullfill

P_1 → Can "

P_2 → Can't

P_3 → Can

P_4 → Can't

① Either can start from P_1 or P_3

Safe sequence:

$< P_1\ P_3\ P_0\ P_2\ P_4 >$

	Available A B C
After P_1 →	3 3 2 → need
	1 2 2 → available
	3 2 2 → return by P_1
	5 3 2 → Available After P_1
After P_3 →	2 1 1
	7 4 3

210

P_0

P_2

P_4

Banker's Algo:

(i) Allocation : Matrix of size $n \times m$.

(ii) Max : " " $n \times m$

(iii) Need : " " $n \times m$

(iv) Available : Array of size ~~m~~ n

(v) Finish : " " " n

Finish [0]

[n]

1. Let work and finish be vectors of length 'm' and n respectively
   Initialize : work = Available.
   Finish = false ; for (i = 1, 2, 3, 4 ... n) .
2. Find i such that both.
   (a) Finish [i] = false.
   (b) Need [i] <= work.
   If no such i exists goto step (4)
3. work = work + Allocation [i].
   Finish [i] = true.
   Goto step (2)
4. If Finish [i] = true for all
   then system is in safe state

1. What will happen if Process P_1 requests one additional instance of resource type A and two instances of resource type C?

   Request P_1 = <1, 0, 2> — check Request is valid or not, after that check Resource is Available or not.

   - Validity → ($Request_i \leq Need_i$) eg. P_1 Can't demand resources more than it's current need
     P Current need → <1 2 2> > demand <1 0 2> Valid → ✓
   - Available → $Request_i \leq Available$ Available → ✓
   - Allocate & check safety :
     as virtually do → & check safety

Process	Allocation	Max	Need	Available
P_1	3 0 2		0 2 0	2 3 0
	~~2 0 0~~ 3 2 2		~~1 2 2~~	~~3 3 2~~

Allocate <1 0 2> to process P_1,
1. $Allocation_i = Allocation_i + Request_i$
2. $Need_i = Need_i - Request_i$
3. $Available = Available - Request_i$

→ Yes Safe.

Yes, request can be granted.

2. What will happen if Process P_0 request one additional instance of resource type A & two " " " " C

<1 0 2>

request will not be granted

Available <1 0 2>

3 3 2
↓ ↓ ↓
2 3 0

Q. Consider a system with 4 processes A, B, C and D. All 3 processes required 6 resource each to execute the ~~net~~ min & max no. of resource of the system should have such that deadlock can never occure & may occure respectively.

	min	max	Allocate	Available
✓	P_1	6	5	1
	P_2	6	5	
	P_3	6	5	
	P_4	6	5	

5 + 1

Deadlock never occurs at → (2) happen

max

	Max	Allocated
P_1	6	5
P_2	6	5
P_3	6	5
P_4	6	5

20 — deadlock occure.

Q. Consider a system with 3 processes that share 4 instance of the same resource type. Each process can request a max of K instances. Resource instance can be requested and released only one at a time the largest value of K that will always avoid deadlock is -

Soln

Process	max	Allocation	available
P_1	K = 2	1	1
P_2	K	1	
P_3	K	1	

Q. Consider a system with n process with single Resource R. Each process requires K instances to execute what is the max. instance of R to cause a deadlock.

Process	Max	allocation	Available.
P_1	K	K-1	(1) to deadlock free.
P_2	K	K-1	
⋮	⋮	⋮	
P_n	K	K-1	
		n(K-1)	n(K-1) + 1

Recovery from Deadlock :

1. Make sure that ~~allocation~~ deadlock never occurs. Prevent

(written)

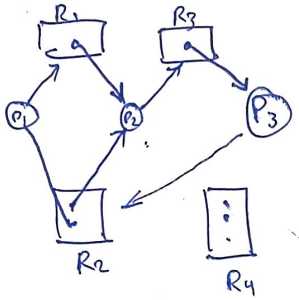
* Deadlock Detection :

Deadlock detection is done using

1. When all resources have single instance. → wait for graph
2. when resource have multiple instance. → detection algo.

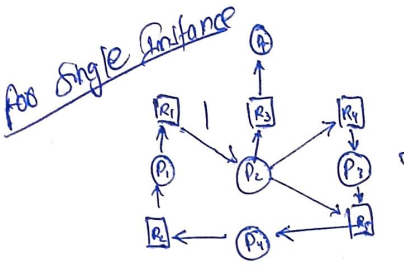
Wait - for - Graph : It is created from resource allocation graph.

↳ vertices ⇒ only process
edge ⇒ from P_i to P_j
(Process P_i is waiting for a resource which is held by P_j)

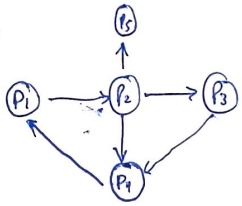


If there is cycle in graph
↓
Deadlock

For Single Instance

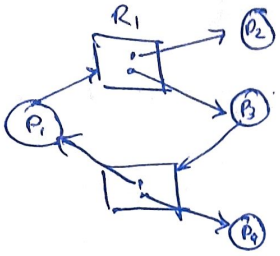


Resource - allocation graph

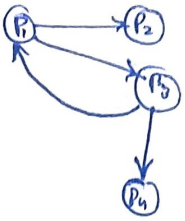


Wait - for - Graph

If a resource category contains more than one instance, then the presence of cycle in the resource allocation graph indicate the possibility of a deadlock, but doesn't guarantee one



⇒



Cycle : Yes
deadlock = No

* resource $\xrightarrow{\text{has}}$ multiple instance. $\xrightarrow[\text{is done by using}]{\text{detection}}$ specific algo

Detection -algo usage-

1. Do Deadlock detection after every resource allocation
2. Do Deadlock detection only when there is some clue.

Recovery from Deadlock:

there are three basic approaches to recovery from deadlock.

1. Inform the system operator and allow him/her to take manual intervention
2. Terminate one or more processes involved in the deadlock.
3. Preempt resource

Process Termination:

1. Terminate all the process involved in the deadlock
2. " process one by one until the deadlock is broken.

Which processes to terminating next

1.
2.
3.
4.
5.
6.

Resource Preemption.

Important issue to be address when preempting resource to retrieve deadlock

1. selecting a victim.
2. Rollback
3. Starvation.

System Call : Programmatic way in which a Computer program requests a service from the kernel

- How System Call work.

Request of Process from user space → Interrupt request for kernel → Control Transfer to Kernel → Kernel checks if operation can be done → Kernel performs Operation → Kernel return the O/P to User process

- Service Provide by SC
1. Process Creation & management.
2. main memory management
3. File access Directory and file system management
4. Device handling (I/O)
5. Protection
6. Networking.

- Process Control :
  1. Create process (window : NtCreateProcess in the window NT native API)
  2. Terminate
  3. Load execute.
  4. get/ set process attributes.
  5. wait for time wait event signal event
  6. allocate and free memory
- File management
  1. Create file delete file.
  2. open Close.
  3. read write reposition.
  4. get / set file attribute
- Device management
  1. request device release device
  2. read write reposition
  3. get / set device attribute
  4. Logically attach or detach device
- Information maintenance.
  1. get / set total system information (including time date Computername)
  2. get / set process file or device metadata (including opener (creation) author, time date)
- Protection
  1. get / set file permission

fork () → If process execute the fork () n time < no. of process = 2^n, no of child process = $2^n - 1$

- Fork System Call is used for creating a new process which is called child process.
  (which runs concurrently with the process that makes the fork() call (Parent Process).

- Parameter & Return Value.
  ↓ no ↓ Integer.

- -ve value : Creation of child process was unsuccessfull
- zero : Returned to the newly created child process
- +ve value : Return to parent . the call contains process ID of newly created child process

Memory Management :

It is module of OS

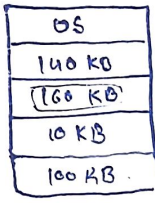
* Function of MM module:
  1. Memory allocation
  2. Memory deallocation
  3. Memory protection.
* Goals of MM module :
  1. Max. utilization of space. (min memory fragmentation)
  2. Ability to run larger programs with limited space. (Using virtual memory concept)
* Memory management Techniques :

Contiguous	Non-Contiguous.
1. Entire process should be stored in MM on consecutive locations	Process can be stored on non-consecutive locations.
2. Two types of Technique. • Fixed partition. • Variable partition	also have two type of techniques: • Paging • segmentation.

* Contiguous Memory Management :

1. Fixed partition contiguous MMT :

- The MM is devided into fixed no. of partitions and each partition can be used to accomodate max. one process. (may be of diff^{n} size)



- new process arrive with size = 50 KB. this process should be allocated in which partition that decission is taken by a policy called Partition allocation policy.
- Process has limited memory address to access after that Process will not able to access any addresses that's how protection is implemented

Partition Allocation policy - which process is allocated in which partition that decission is taken under this policy.

Partition allocation policy
1. First fit
2. Best fit
3. worst fit
4. Next fit

Max degree of Multiprogramming (no. of process) is limited by no. of partition.

no. of partition = max no of process allocated at a time

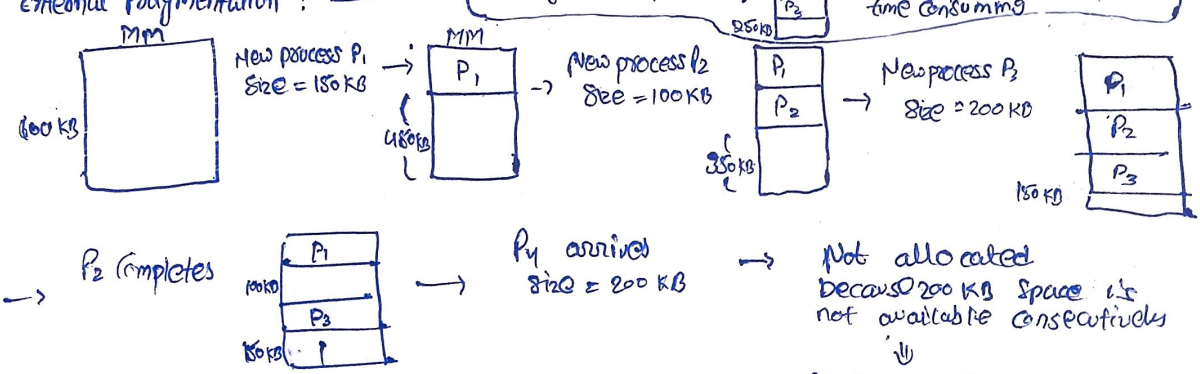
* whichever policy is used ⇒ there is internal fragmentation. (Like 50KB Process allocated in 100KB fixed partition) there is internal fragmentation (wastage space) is 50KB.

Internal fragmentation : If extra space is allocated to process more than required space. Hence wastage of that extra space is known as internal fragmentation.

2. Variable partition contiguous MMT :

- MM is not devided into partition initially
- when a new process arrives a new partition is created of same size as process size. and a process is allocated into that partition.
- Hence no any internal fragmentation
- External fragmentation :

Sol^{n}: Compaction : Collect entire process into one side of MM so that other side of MM can have entire free space. after Compaction : P1, P3 (250KB) — costly process time consuming



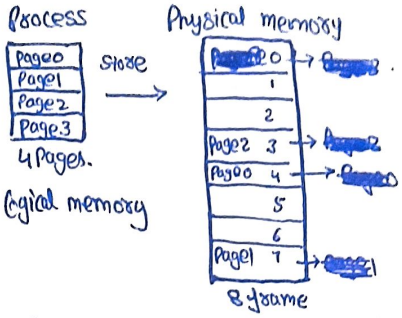
Defn: If enough space is available to store a process but not in consecutive hence wastage of space is known as external frag.

Non-contiguous MMT: Process is scattered in memory not allocated at one area

Two Techniques : ① Paging : scattered in same size of memory areas.
② segmentation : scattered in variable size of memory areas

Paging : Process is divided in equal size of Pages.
Physical memory is divided in same equal size of frames.
Pages are scattered in frames.

e.g.



Note: OS will maintain a particular table which is list of which process is stored in which frame of the Physical memory is known as Page table

Page table is also stored in MM in form of Pages (in single or multiple frame)

0	4
1	7
2	3
3	0

Frame no.

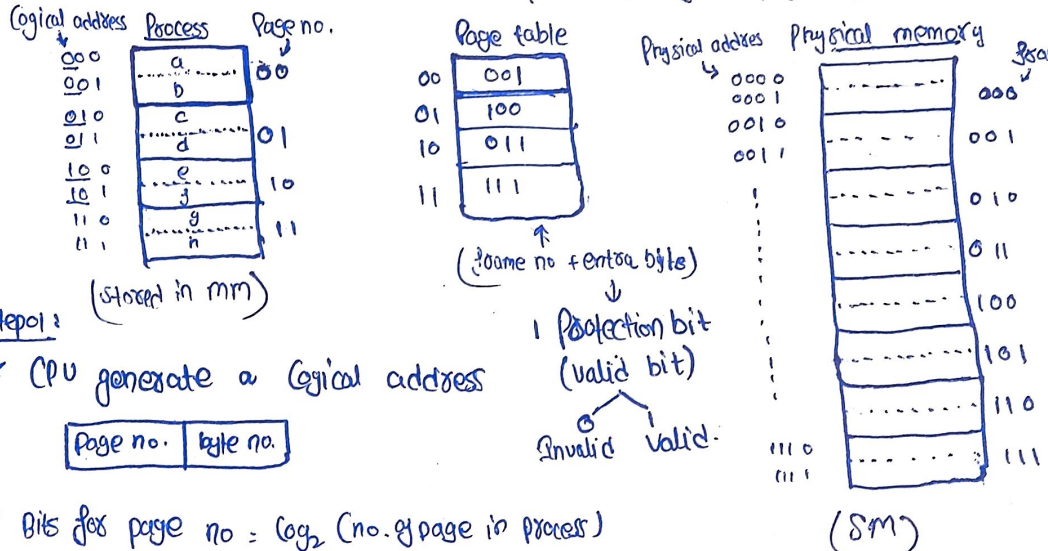
Note: frame no. we keep into page table.

- Process will have a view of process & its pages
- Page table is used to map a process pages to physical frame.
- No of entry in Page table = No of Pages in Process
- each entry of Page table contains frame no. + extra bit
- Page table size = No of entries in PT x 1 entry size
  No. of Pages in Process x 1 entry size
  Page " " " " x (Frame no + extra bit size)

★ whenever CPU request a specific content.
⇓
1. find out content present in which page.
2. search PT & get the frame no. in which the page stores.
3. Go to that frame no. Get the content frame that frame

★ 2 times physical memory is accessed — one for page table.
one for content

Logical to physical address Conversion (How CPU fetch page from the MM:



1 dirty / modified bit
0 not modi
1 modify
M=0 means while replacement a page is replaced from MM directly without coping it back to secondary memory it saves time

Step01 :

★ CPU generate a Logical address

Page no.	byte no.

- Bits for page no = $\log_2$ (no. of page in process)
  Bits of byte no = $\log_2$ (Page size)
  Logical address space = Collect of all logical addresses (Process size).

Step02 CPU generate Logical address = 101 ⇒ Page no. = 10 & byte no = 1
⇓
Page table.
⇓
frame → 011

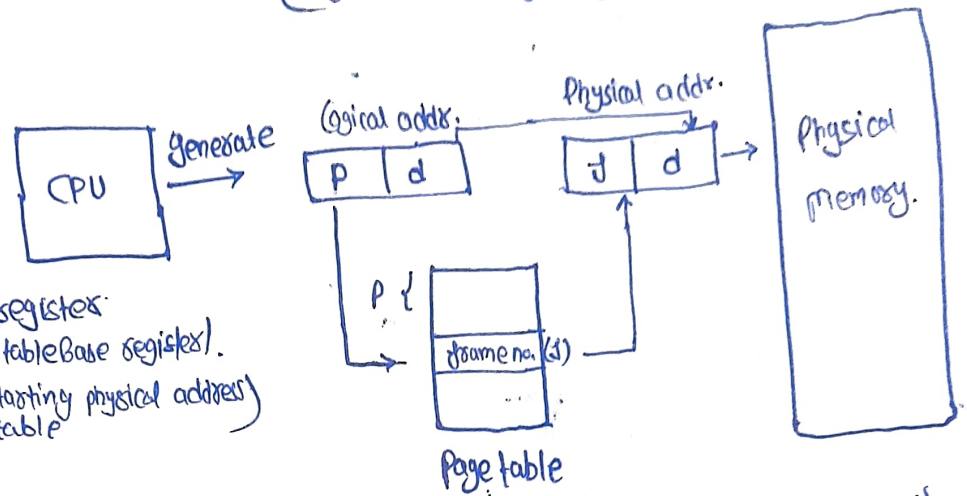
011	1

→ Physical address.

Step 03 Physical address | frame no | byte no |

Bits for frame no = $\log_2$ (no. of frame in physical memory)
Bit for byte no = $\log_2$ (page size). (line no. / displacement / page offset)
Physical address space = collection of all physical addresses (physical memory size)

Diagram:

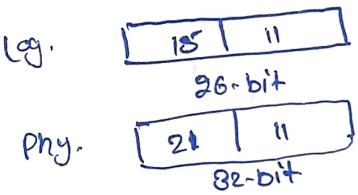


these is a register PTBR (Page table Base register). (which hold starting physical address of Page table

? then How CPU know the location of Page Table ← (It is also stored in physical memory)

Q. Logical address = 26-bit.
Physical address = 32-bit.
Page size = 2KB = 2^{11} bit.
One Page table entry size = 4 bytes

1. Bit in Page offset : 11
2. No. of Pages in Process: 2^{15}
3. Bits for page No: 15-bit
4. Number of frames in physical memory: 2^{21}
5. Bits for frame no : 21-bit
6. Page table size : Page no * each page size,
7. = $2^{15} \times 4B = 2^{17}$ byte.
8. = 128KB.



Time Required in Paging:

- Effective memory access time = (Page table access time + content access time) from mm.
  By default P.T is stored in MM
  = 2tmm (tmm = main memory access time)

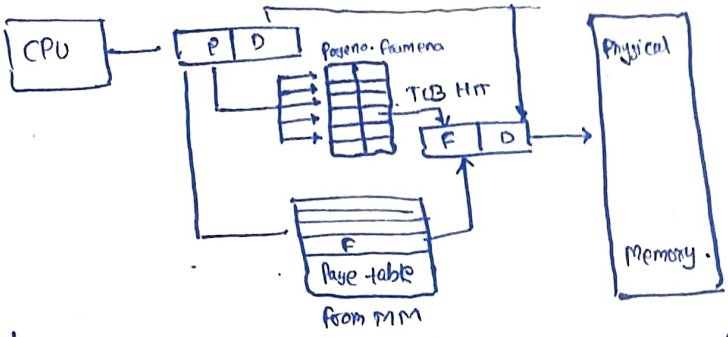
Special Case : If PT is very small is stored in Registers
= tmm (Register access time is negligible)

Eff. tmm = $t_{PT} + t_C$
= 2tmm
= tmm Special case

Performance Improvement:

TLB (translation Lookside Buffer) is used to improve the performance of Paging.

TLB : Translation Lookside Buffer is a memory hardware that is used to reduce the time taken to access a user memory location.
It store few frequently accessed page table entries so that can get physical addr. without accessing the mm in less time.



TLB Hit → Required PT entry is present in entry

TLB Miss → Required PT " " not " in entry

Effect access time with TLB.

$T_{effective} = H * (t_{TLB} + t_{mm}) + (1-H) * (t_{TLB} + 2t_{mm})$

t_{TLB} → translation t_{mm} → content

TLB → to ensure its a miss $2t_{mm}$ < translation content

Simplify:

$T_{eff} = t_{TLB} + t_{mm} + (1-H)\, t_{mm}$

Q. $t_{TLB} = 40ns$.

TLB reduce effective memory access time from 400ns to 280ns? what is hit ratio. = ?

Sol^ t_{eff} without TLB = 400 = 2 * t_{mm}. ⇒ $t_{mm} = 200ns$

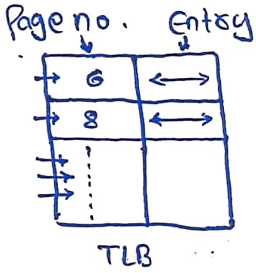
t_{eff} with TLB = 40 + 200*(1-H) 200

$\frac{280-240}{200} = (1-H) \Rightarrow \boxed{H = 0.8}$

TLB Mapping: TLB is searched using page no. of logical address. And there is a pattern to bring PT entry into TLB that pattern is known as mapping.

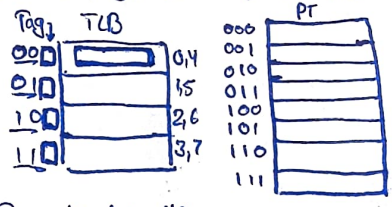
Types: (i) Direct mapping
(ii) set Associative mapping
(iii) fully Associative mapping.

Fully Associative mapping: using Content addressable memory (doesn't have address)



Direct Mapping:

TLB can store = 4 entries
PT can store = 8 entries



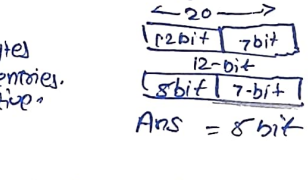
Tag will identify the entry is 0 or 4

Tag	TLB entry no.

* Bits for TLB entry no = $\log_2$ (no. of entries in TLB) -bit
* Total tag-directory size = No. of entries in TLB * tag-bits.

Q. CAS = 128 MB. = 2^{27}
Page size = 4 KB = 2^{12}
1 Page table entry = 4 Bytes.
TLB size = 1K bytes.

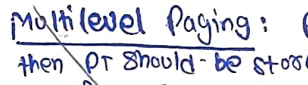
1. PT size
2. Tag directory size



① PT size = $2^{15} \times 4 = 2^{17}$ B

$\frac{1KB}{4B} = \frac{2^{10}}{2^2}$

② 15 = 128 KB.



no. of entries = $\frac{2^{10} * 2^7}{2^2} = 2^8$

Tag directory size = $2^8 * 7$ bit

Problem with direct mapping:

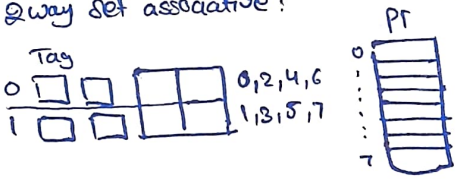
i) CPU generates request page no. like 1,5, 1,5,1,5
TLB miss is more
So eliminate this problem we derive set-associative mapping

Set-Associative Mapping:

Page no

Tag.	TLB set no.

2 way set associative:



* Bits in TLB set no = $\log_2$ (no. of set in TLB)
* no. of set in TLB = $\frac{\text{No. of entries in TLB}}{K}$

where - K → associative for K-way associative.

Q. LA = 20-bits
Page size = 256 bytes
TLB size = 512 entries.
4-way associative.
Tag bits = ?

Ans = 8 bit

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

Multilevel Paging: PT size > Page size, then PT should be stored on multiple pages.

e.g. Process size = 32 B. no of Pages = $\frac{32}{4} = 8$.
Page size = 4B.
Page table entry size = 1B.
Page table size = 8 * 1B = 8B
no. of pages required for PT = $\frac{8B}{4B} = 2$

Let understand PT in memory:

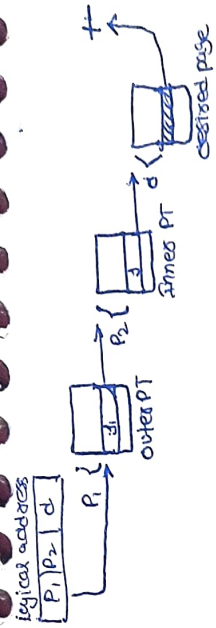
Multilevel Paging:

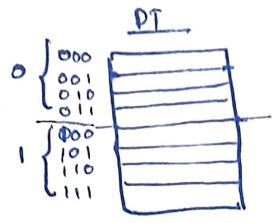
PT size > 1 Page size then PT should be stored on multiple pages in MM

Let take an example to understand that.

Process size = 32B
Page size = 4B.
Page table entry size = 1B.
Page table size = 8 * 1B = 8B.
No. of page = $\frac{32}{4}$ = 8 pages

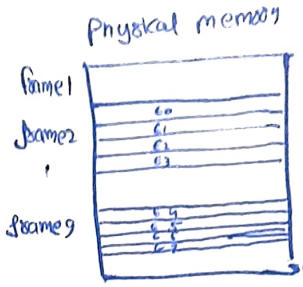
Here page table size (8B) > 1 Page size (4B)
So, No. of pages required to store PT in MM = 8/4 = 2

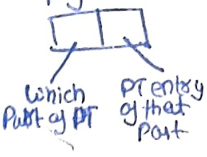




Outer Page Table

0	frame no. 2 / frame no. 4
1	





no of PT entries in one pg = $\frac{4B}{1B}$ = 4

Ques: No. of pages = 2^{15}
Page size = 2KB.
PT entry = 4B.
PT size = $2^{15} \times 4$ = 128KB.
no. of Pages for PT = $\frac{128}{2}$ = 64 = 2 (Outer PT)
no of PT entries in one Page = $\frac{2KB}{4B} = 2^9$ (Inner PT)

6	9	11

Q. No. of Pages = 2^{10}
Page size = 8KB
entry size = 4B
no. of entries in one Page = $\frac{2KB}{4B} = 2^9$

LA

20	11

P1	P2	D
11 bit	9	11

this outer PT also exceeding the limit → we can not search in a single PT with 11 bit then again have done more-and-more partition

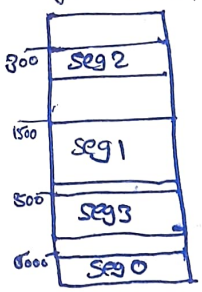
2	9	9	11

Segmentation :

- Divide Process in logically related partitions (segments)
- Segments are scattered in physical memory.
- Segments can be of variable sizes

Segment Table

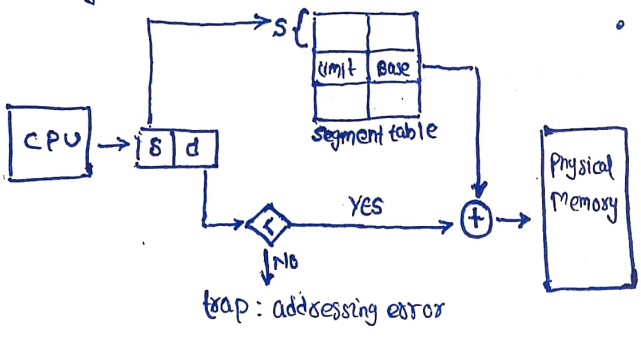
	Base	Limit
0	6000	600
1	1500	1000
2	300	300
3	500	480



Base: Starting address of segment in MM
Limit: Size of Segment

Note

- Size of Segment can vary, so along with base, keep limit information also.
- Limit define max number of words within the segment. (Provide Protection memory protection)
- Physical address = Base + d (d < limit)

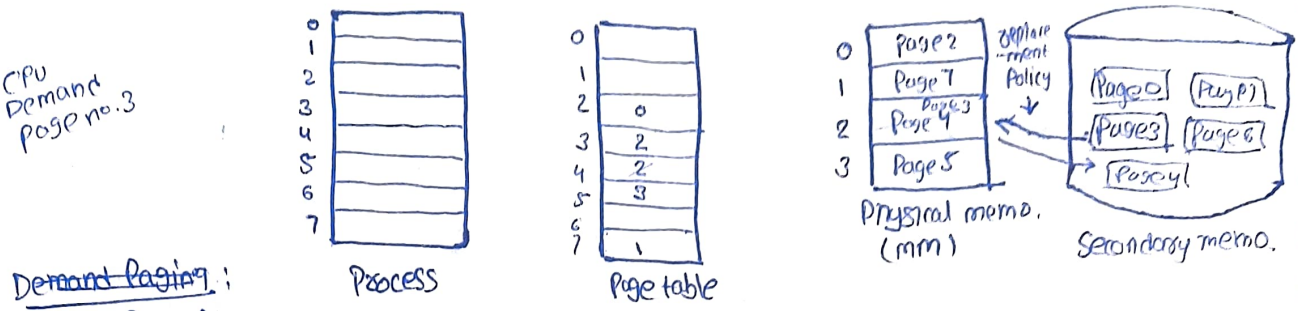


Q. Find Physical address for following request?

S	d	Physical address (Base+d)
0	430 ✓	6000 + 430 = 6430
1	962 ✓	1500 + 962 = 2462
2	321 ✗	300 + (321 > 300 (limit)) → addressing error
3	441 ✓	5000 + 441 = 5441

Q. Max. segment size = 2KB = 2^{11}B ⇒ d=11-bits
Number of segment in process = 2^{10} ⇒ S= 10 bits
Logical address = 21... bit | S=10 | d=11 |

Virtual Memory
- Feature of OS
- Enable to run large process with smaller available memory



Demand Paging:

Note Page fault: If CPU demand a page which is not present in MM then it is known as page fault.

Page fault service — In case of page fault, OS bring faulted page from secondary memory to MM by replacing a page (if needed) & (update the ~~text~~ page tables)
↳ DMA might required for the data transfer ~~between~~ Hard Disk & MM

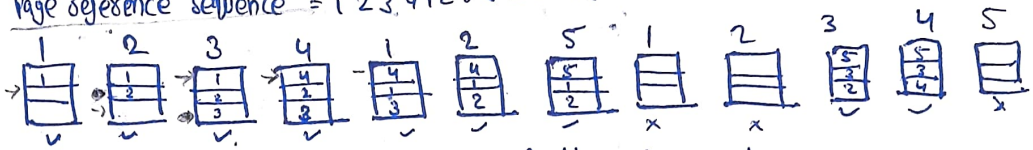
Demand Paging: Bring pages in memory when CPU demand
Types. ① Pure Demand Paging.
↳ Initially All the pages are kept in the secondary memory
② Demand Paging.
few pages are kept in MM & few are in secondary memory

Effective access time. Virtual memory = (1-p) * Effective memory access time + p * page fault service.
p = page fault rate.

Page Replacement Policy: FIFO, optimal, LRU, LFU (frequently), MFU, LIFO, Second chance

- FIFO (first in first out): Replace that page which is brought to MM first.

Assume, No. of frames = 3 (All empty initially)
Page reference sequence = 1 2 3 4 1 2 5 1 2 3 4 5



no. of page fault = 9. Page fault rate = 9/12.

Belady's Anomaly: For some page reference sequence, increasing no. of frames increase no of page fault also.
Only FIFO suffer from this.

Advantages: ① Simple & easy to implement
② Low overhead
Disadvantages: ① Poor Performance
② Suffer from Belady's Anomly
③ Doesn't consider the frequency of use or last used time simply replace the oldest page.

- Optimal Policy: Replace a page which is not going to be refered in near future.
It provide minimum no. of page fault.
Advantages: Easy to Implement, Simple data structure are used, Highly efficient
Disadvantage: Require future knowledge of the program, Time consuming
Due to future requirement hence practically not possible to implement

LRU (Least Recent used): Replace the page which CPU has not used since longest time.
Advantage: efficient, Doesn't suffer from Belady's
Dis: Complex Implementation, Expensive, Require H/w support

Counting Algorithm: Counting Algo look at the no. of occurances of a particular page and use this as the criterion for replacement.
- LFU (Least frequently used) . MFU (most frequently used)

- LFU: Replace page which has been used mini. no of times.
- MFU: " " " " " " max. " " "
- LIFO: (Last in First out).

- Second chance replacement: · It is a variant of FIFO
  · But gives a second chance to those pages to which CPU used again
  · Each page will have an associated reference bit.
  · Replace page using FIFO but with R=0 — bring a new page in MM — page got used
  · When a page second chance then reset R=0

Two types of request
- Page no. or Page reference seq.
- Logical add or byte address → get the page no & then form page reference seq.

Frame Allocation: ① Equal Allocation ② Propotional Allocation. e.g. 6 frame, 2 process, P_1=8KB, P_2=11KB

$P_1 \to 3$ frame · $P_2 \to 3$ frame.

$P_1 \to \frac{8}{8+11} = \frac{2}{3} \to \frac{2}{3} \times 6 = 4$ frame similarly, $P_2 = 2$ frame

How many frames do we allocate per process?
- In single user, single tasking system its simple - all the frame belongs to the user's process.
- 2 Questions: What is minimum no. of frames that a process needs.
  Is page replacement is Global or Local.

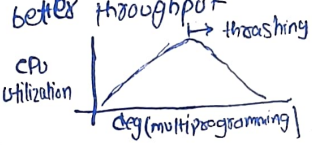
minimum no. of frame: Every process have enough pages to complete an instruction.

~~Fram Allo~~

Local vs Global Allocation:
- Local replacement requires that the page being replaced be in frame belonging to the same process.
- The no. of frames belonging to the process will not changed.
- This allow processes to control their own page fault rate.

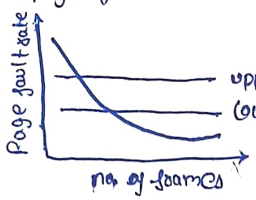
Global (Better)
- the process can replace a page from a set that include all the frames allocated to user process.
- High-priority process can increase their allocation at the expense of lower-priority process.
- Global allocation make for more effecient use of frames & their better throughput

Thrashing:
- high-level paging activity.
- Degree of multiprogramming → no. of processes.
- where CPU utilization start decreasing → thrashing.



How to handle thrashing:
- Locality model
- ① working set model → define particular set size = size of w. set > locality. " " " " < locality (Problem to store)
- ② page fault frequency

CPU spents too much time for page fault service as compared to the process execution (productive work)



(Increase no. of frames)
(decrease no. of frames)

Problem with Virtual memory: Page table is too large and so many page table entries are invalid
Solving this problem: using inverted Page table.

Types of Page table:
① Hirarchical PT (multilevel page table) ✓
② Inverted PT
③ Hashed PT

Note: valid entries in PT → no. of pages are in MM. / max no of entries = No of frame in MM.

② Inverted PT:
- Smaller PT size as compared to normal PT
- to identify page no. of which process store pid along with page no.

Each entry in the inverted PT contains.
1. Page no:
2. Process id:
3. Control bit
4. Chained pointer:

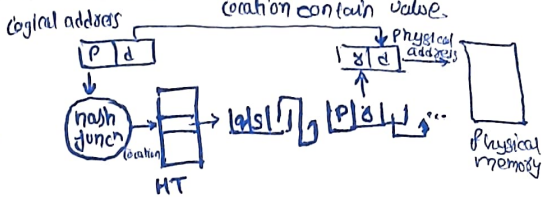
CPU generate: | Pid | PI | di | as logical address

Hashed PT : In this virtual page, the no. is hashed into a page Table. this PT mainly contain a chain of element hashing to the same element

Hashing : key : values.

location = H(key) H → Hash function. location contain value.

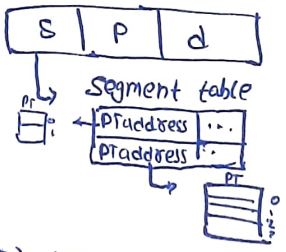
Each element mainly contain
- Virtual page no.
- The value of the mapped page frame.
- A pointer to the next element in the LL

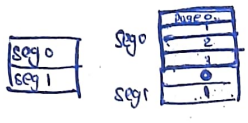


Segmented Paging :-
- Pure Segmentation is not very popular and not being used in many of the operating system.
- Segmentation can be Combined with Paging to get ~~most~~ the best features out of the both technique.

- Divide each segment into multiple pages.
- Keep one page table for each segment.
- Segment table entry points to the page table of the segment
- MM has frames of size equal to page size and each page can be kept on any frame.

- L.A. :

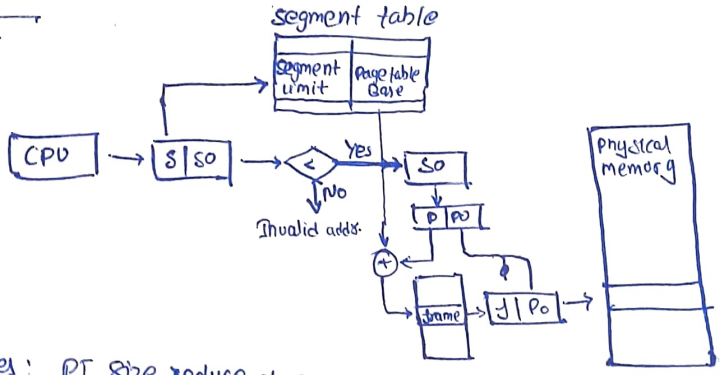




S → segment no.
P → Page no of the segment
d → offset or displacement or Line number

- No. of bits in S → depend on no. of segment
  " " " " P → " " " of pages in segment
  " " " " d → " " Page Table

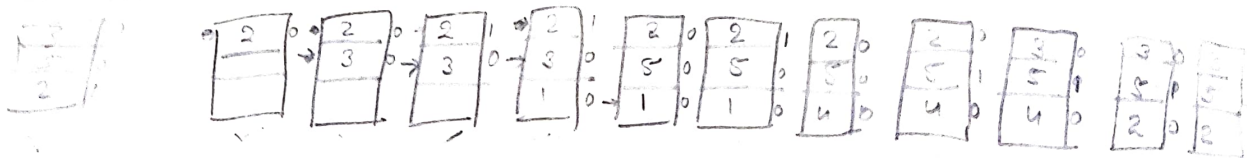
- H/W :



S → Segment table.
So → Segment offset
P → Page Number
Po → Page offset
f → Frame Number

- **Advantages :** PT size reduce as pages are present only for data of segments, hence reducing the memory requirement
  Given programmers view along with the advantage of paging.
  Reduce external fragmentation in comparison with segmentation.
  Since the entire segment need not be swapped out the swapping out into virtual memory becomes easier.
- **Disadvantages :** Internal fragmentation still exists in page
  Extra Hardware required
  Translation becomes more sequential increasing the memory access time
  External fragmentation occurs because of varying size of PT and of ST in today's system.

Second chance. 2 3 2 1 5 2 4 5 3 2 5 2 — Page reference sequence



File System :

A file is a named collection of related information that is recorded on secondary memory / storage.

File Attribute : ① Name ② Extension, ③ Size ④ Data ⑤ Author.
⑥ Create, modified, Accessed ⑦ Attributes - Read only, hidden
⑧ Default program ⑨ security details

File Directory : Collection of files. (Folder)

File System : • module of OS which manages, control & organizes file and related structure.
• It is a part of OS which give organized view of files.
• Provides us beautiful view of ugly H/w (secondary memory)

Types of file system : 1. FAT 32 (window OS) (File allocation table) 2. NTFS (New tech FS) 3. HFS+ (mac OS) 4. Ext2/3/4 (Linux) 5. Swap (used for swap-in & swap-out)

Secondary storage → HDD → magnetic disk
Single disk → optical disk

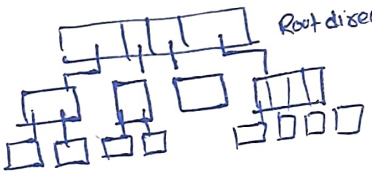
File Directory structure :

1. single level Directory →  Single directory
All file should be unique. → are not classified on the basis of user.
Multiple user can't keep their file in single directory.

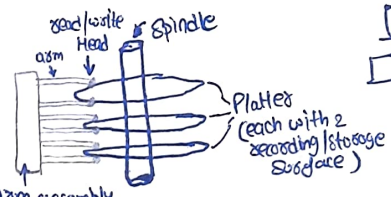
2. Two level Directory →

user1	user2	user3
dir1	dir2	dir3

3. Tree Structure Directory →



Magnetic Disk :



(3G-track Entire Disk)

Top view of MD :

- Each surface is divided into concentric track
- Each track further divided into sector
- Each sector has constant storage capacity

(smallest unit of MD) which can be read or write at one.

- Sector is a addressable unit of disk
- No. of sectors in disk (ns) = 2 × no. of platters × no of track per surface × no. of sector per track
- add. size = $(\log_2(ns))$-bits per disk
- Cluster : Collection of consecutive sector

Sector Capacity of Disk : ns × 1 sector capacity

Disk Access Time :

Disk Access time :

~~read/write head to move to the correct track on the magnec~~ the time taken for

Cylinder : Collection of track of same radius from all surface.
Content in disk stored cylinder wise so that seek time can be saved.

Disk Access Time :

Seek + Rotational Latency + 1 sector transfer time + additional delay.

① Seek time : the time taken for read/write head to move to the correct track on the magnetic disk is called seek time.

② Rotation delay / Latency : the time required by the read/write head to rotate to the requested sector from the current position.

③ sector transfer time : Time taken to transfer data from the disk or read/write 1 sector

Note: avg rotational latency = 1/2 disk rotation time / 2.
1 sector transfer time = 1 disk rotation time / no of sector per track.

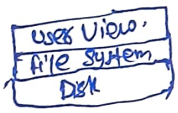
Disk Partitioning :

- Low-level (physical) : It is creating the tracks & sector done by manufactures.
- High level (logical) : creating file drive, file, role done by user.

Logical formatting —
- Primary Partition : use to store as 1 users file both.
- Extended partition : use to store only users files

Disk Block :

It is a logical representation of smallest unit of disk.
1 Block ⇒ 1 to 2 disk sector (based on whichever OS used)



Note: A file is smaller than a block should occupy entire blocks
Hence there will be internal fragementation.

File System : Free Space Management :

1. Free List : Linked list of all free blocks.

2. Bitmap Method :

n-blocks in disk, n-bit info ⇒ if bit { 0: block is free, 1: occupied }

No searching in free list but in bitmap we search for bit zero
free list is faster in allocating free block
free list one is variable whereas bitmap is constant.
these blocks are free.

File Allocation method :
1. Contiguous method
2. Linked Allocation
3. Indexed Allocation.

Performance :
1. Fragementation : Internal, external
2. Increase in file size : Inflexible.
3. Type of access : Sequential, random/direct

1. Contiguous method : File Allocation table

File name	Start block no.	no. of blocks
abc.doc	4	3

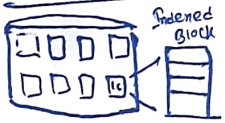
2. Linked Allocation :

File Allocation table.

File Name	Start block no.	Last block no.
abc.doc	4	7

Performance
1. Fragmentation : Internal
2. Increase in file size : Flexible
3. Type of access : sequential

3. Indexed Allocation



File Allocation table

File Name	Index Block
abc.doc	16

Performance.
1. Fragmentation : Internal
2. Increase in file size : Flexible
3. Type of access : sequential

Multilevel indexing

Master Boot Record: A MBR is a special type of boot sector at the very begning of Partitioned computer mass storage devices.
- Contains the info. regarding how & where os is located in the hard disk So that it can be booted in the RAM.

Unix I-node : The inode (indexed node) is a data structure in Unix style file system that the describes a file system object such as a file or a directory
Each node store the attributes & disk block location of the object's data
The no. of Inode limit the total no of file / directories that can be stored in system.

Disk scheduling : Done by OS to Schedule I/O requests arriving for the disk.
Using this we will try to save seek time.
Disk Scheduling algorithm : ① FCFS ② SSTF ③ Scan ④ C-Scan ⑤ Look ⑥ C Look