Operating Systems

An operating system is a piece of software that manages all resources of a computer system i.e, both hardware and software, & provide an environment in which user can execute his/her program in a convenient & efficient manner & acting as Resource manager.

- What if there is no OS?
  (a) Bulky and complex app.
  (b) Resources exploitation by ⊥ App
  (c) No Memory protection.
- What is an OS made up of → collection of system software.
- Application software performs specific task for the user.
- System software operates & control the computer system & provides a platform to run application software.

Surbhi Shekhar 20BCS114

* Operating System functions :-

- Access the computer hardware.
- Interface b/w the user & the computer hardware.
- Resource management (i.e, memory, file, device, proces).
- Hides the underlying complexity of hardware.
- facilitates execution of application program by providing isolation & protection.

Throughput :- No. of task perform per unit time.
(linux)

Goals of OS:-

(i) Maximum CPU utilization

(ii) Process starvation not occur
↳ (Aisa na ho ki kisi aur job ko mauka na mile). Ek he time lele.

(iii) High Priority Execution

Types of OS

Surbhi Shekhar

(i) Single process OS → Execute one job at a time. (oldest)
e.g- MS DOS

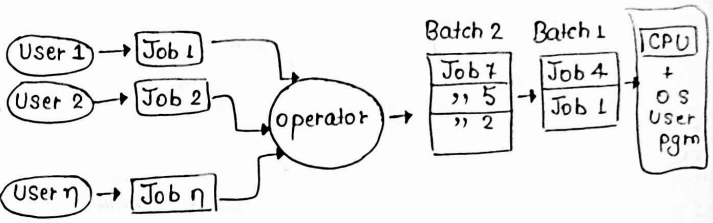
(ii) Batch processing OS → (like punch card - ATm)

- Operator works → sorts the jobs.

(a) 1st ly, User prepare his job using punch cards.
(b) Then, submit the job to computer Operator.
(c) Operator collects ~~submit~~ jobs from diff. user and sort the jobs into batches with similar needs.
[e.g:- ATLAS]

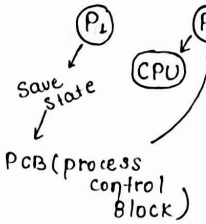
(d) Then, Operator submits the batches to the processor one by one.
(e) All the jobs of one batch are executed together.

- Priorities cannot be set, If job comes with higher priority.
- May lead to starvation (A batch take more time to complete)
- CPU may become in case of I/O operations.



(iii) Multiprogramming OS → Single CPU

- Context Switching for processes



Surbhi Shekhar

(Phele ek process complete krega until I/o operation na aaye)
[e.g:- THE]

- Increase CPU utilization by keeping multiple Jobs (code and data) in the memory so that the CPU always one to execute in case some job gets busy with I/O
  - Single CPU
  - Context switching process.
  - Switch happens when current process goes to wait state.
  - CPU idle time reduced.

(iv) Multitasking → logical extension of multiprogramming. (Premetive - bich mai chod Jana)
(All goals)
- single CPU
- Able to run more than one task.
- Context switching & time sharing used.
- Increased responsivness.
- CPU idle time is further reduced.

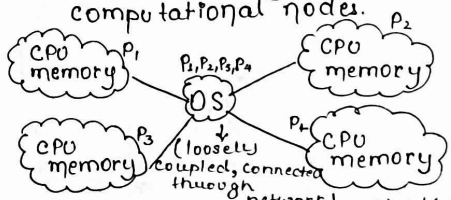
(Ek time cha h process ka utne time hua to thik nahi to next) (eg- CTSS)

(v) Multiprocessing OS → more than 1 CPU in a single computer.
- Increase reliability, 1 CPU fails othe can work.
- Bettter throughput

e.g:- [windows]

- Lesser process starvation (if 1 CPU is working on some process, other can be executed on other CPU.
- Context switching & Time sharing is also apply.

(vi) Distributed OS → (loosely coupled)
- OS manages many bunches of resources, >=1 CPUs, >=1 memory, >=1 I/OPUs etc.
- Loosely connected autonomous, inter connected computer nodes.
- collection of independent, networked, communicating & physically separate computational nodes.



[eg:- AIX]

(vii) RT OS [Real Time OS] → Hard (delay nahi hoga time imp.)
Soft (lit bit delay)
(delay nahi hoga time matters)
- Error free, computations withintight-time boundaries.
- Air Traffic control system, ROBOTs etc
- Must be fast .. (jb kam mile tab kaye)

Used - ACT, Industrial application

Sorbhi Shekhar

Process

When we double click exe file that convert into process.

Process → Program under execution.
Whenever a program comes in RAM called process.

Program :- A pgm is executable files contains a certain set of instructions written to complete the job
- It is compiled code,
- Store in S Disk

THREAD (minute work that execute independently).
- Light weight process.
- Independent execute.
- Single ~~stream~~ sequence stream with process.

Multi tasking	Multi threading
(i) More than 1 process Context ~~concept~~	(i) More than one threads Context. ~~concept~~
(ii) Isolation & Memory protection concept.	(ii) No isolation & memory protection concept.
(iii) Processes are ~~schedule~~ schedule	(iii) Threads are schedule
(iv) No. of CPU 1	(iv) More than 1 (Better)

Thread Scheduling :-
Threads are scheduled for execution based on their priority.
Even though threads are executing within runtime, all threads are assigned processor time slices by OS.

Difference b/w

Thread context switching	Process context switching
(i) Thread contex switching	(i) Process context sw..
(ii) Fast switching	(ii) Slow
(iii) CPU's cache state is preserved.	(iii) CPU's cache state is flushed.
(iv) Doesn't includes switching of memory address space.	(iv) Include switching of memory address space.

THREAD
E.g:- Multiple tabs in browser, text editor (when you are typing in an editor, spell-checking, formatting to text and saving the text are don concurrently by multiple threads).

Sujitkumar Thakur

Components of Operating System

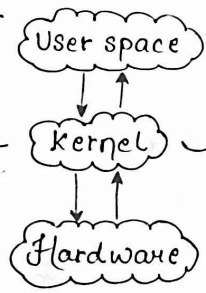
(i) User space

- App are run there
  - → No hardware access
  - → provide convient environment to user and apps
- (a) GUI (Graphical User interface)
- (b) CLI (command line interface) - Terminal. window - (power shell)
- for linux, we can (ex) Use (MKdir) command to create new folder.

→ User space interact with Kernel.

(ii) Kernel

- Access to underlying hardware.
  - → Heart of OS
  - → directly interacts with hardware.
  - → It is the core component
  - → Very fast part of OS to load on start-up.



main component of OS. & also OS made up of this 2 component.

OS

Archi....

functions of Kernel :-

(i) Process Management
- (process creation, termination)
- Process & thread schedule
- Kernal also do context switching & time switching.
- It synchronize the process.
- And also commonicate b/w the process.

(ii) Memory Management :-
- Allocating and deallocating memory as per need.
- Free space management.
- Keeping track of which part of memory are currently being used by which process.

(iii) File Management :-
- Create / delete files
- Create / delete directories (directory mang.)
- Mapping files into Sec. storage.
- Backup support onto a stable storage media.

(iv) I/o Management :-
- Manage & control all I/o devices.

continue... next page

(a) Buffering (data copy b/w 2 devices), Caching & spooling).

(i) Spooling
- within differing speed 2 jobs.
- E.g. print spooling & mail spooling.

(ii) Buffering
- within one job
- E.g. Youtube video buffering

(iii) Caching
- memory & web caching.

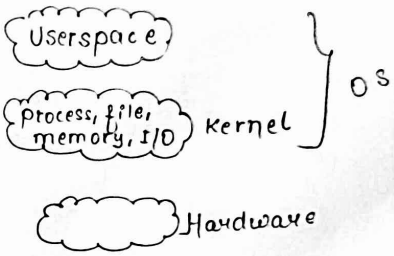
Types of kernel:-

(i) Monolithic kernel:-

(a) All functions are in kernel itself
disad:- (b) Bulky in size
(c) Memory req. to run in high
(d) Less reliable, one module crashes --> (whole kernel is down)
(e) High performance as communication is fast.
(less user mode, kernel mode overheads)

E.g -> Linux, Unix, MS-DOS

(Adv)
- fast communication b/w different component of kernel.
- Less Reliable.



(ii) Micro Kernel:-

Dis-Adv.
- performance less
- Overhead occur

Advantage:-
- Less bulky
- More reliable
- Stable

(a) only major functions are in kernel
  (i) Memory manag.--
  (ii) Process "
(b) File & I/O management are in User space.
(c) smaller in size
(d) More reliable
(e) More stable
(f) peformance is slow
(g) Overhead switching b/w user & kernel mode.

e.g:- L4 linux, MINIX, Symbian OS.

(iii) Hybrid Kernel:-

(a) Advantage of both worlds (file manag. in user & rest in Kernal space).
(b) Combined approach
(c) Speed increases
(d) Stability increase
(e) IPC also happens but less overheads.

(iv) Nano/Exo Kernels:-

Questions:-

Q.) How will communication happen b/w user mode and Kernel mode?

Ans.) Inter process communication (IPC)

∴ Done by Shared memory and message passing.

System Calls

How do apps interact with Kernel? → using system calls.

User mode to Kernel mode:-
(i) Execute process
(ii) Software interrupt

Example:- Creating a folder name (movies)

E.g. Mkdir indirectly calls kernel and asked the file mgmt. module to create new directory.
- Mkdir is just a wrapper of actual system calls.
- Mkdir interacts with kernel using system calls.

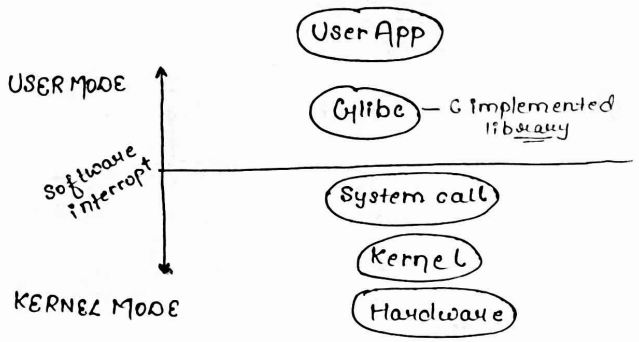
E.g Creating a process.
- User executes a process (US)
- Uset system call (US)
- Exec System call to create a process.
- Return to US.

* Transitions from US to KS done by software interrupt
* System calls are implemented in C.

A system call is a mechanism which a user program can request a service from the kernel for which it does not have the permission to perform.

User programs typically don't have permission to perform operations like accessing I/O devices & communicating other programs.

- System calls are the only way through which a process can go into kernel mode from user mode.



* Types of System Calls

(1) Process control
- (a) end, abort
- (b) load, execute
- (c) create process, terminate process
- (d) get process attributes, set process att...
- (e) wait for time
- (f) wait event, signal event
- (g) allocate and free memory.

(ii) file Managment
- (a) Create file, delete file
- (b) open, close
- (c) read, write, reposition.
- (d) get file attri..., set file attributes

(iii) Device Managment
- (a) req device, release device
- (b) read, write, reposition
- (c) get & set device attributes
- (d) logically attach or detach devices.

(iv) Information maintenance
- (a) get time or date, set time or date
- (b) get system data, set system data
- (c) get process, file, or device attributes
- (d) set process, file, or device attributes

(v) Communication Management
- (a) create, delete communication connection
- (b) Send, receive messages
- (c) transfer status information
- (d) Attach or detach remove devices.

Examples of Windows & Unix System calls

Category	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseFile()	close()
	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()/ chown()
	SetSecurityDescriptorGroup()	
Device Manag....	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Manag....	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()

How Operating System Boots up?

* What happens when you turn on your computer?

(i) Power on

(ii) CPU loads BIOS & UEFI (Basic input-output system)

CPU is the main component for turning on the computer

Fork :- (system call, used to create a child process)

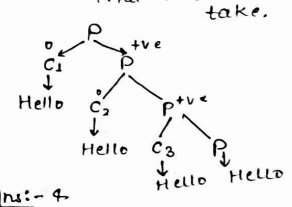
(Parent ka clone hoga)

fork() → 0 (child)
→ +1 (parent)
→ -1 (child not creat for some reason). 0 that condition doesn't take.

Total = 2^n

Child = $2^n - 1$.

Q.)
```
#include<stdio.h>
#include<unistd.h>
int main()
{
  if(fork() && fork())
    fork();
  Printf("Hello");
  return 0;
}
```



Ans:- 4

Process

(i) System-call involved
(ii) OS treat diff. process differently
(iii) Context switching slow
(iv) Blocking a process will not block another
(v) Independent.
(vi) Diff. process have diff. copies of data, files, code.

Threads

(i) System call not involved.
(ii) All user-level threads treated as single task for OS.
(iii) fast
(iv) Blocking a thread will block entire process
(v) Interdependent.
(vi) Thread share same copy of code & data.

User-level Thread	Kernel-level Thread
(i) User level thread managed by User level library.	(i) Kernel level thread managed by OS.
(ii) Typically fast	(ii) Slower than.
(iii) Context switching faster	(iii) Slower.
(iv) User level threads perform blocking operation then entire process blocked.	(iv) If one kernel level blocked, No affect on others.

Scheduling Algorithms

Pre-Emptive

- → SRTF (Shortest Remaining time first)
- → LRTF (Largest Remaining time first)
- → Round Robin
- → Priority based.
  - ↓
  - (Most in Preemptive)

Non-Pre-Emptive

- → FCFS (first come first serve)
- → SJF (Shortest Job first)
- → LJF (longest Job first)
- → HRRN (Highest Response Ratio Next)
- → Multilevel Queue.
- → Priority based

* Chalte hue process ko rok kai dusre process ko CPU mai daldete hai to woh Pre-Emptive hota.....!!! (fixed time)

Ready ⇄ Running (priority of higher)

* Ek process ko pura execute kaune kai baad he dusre process ko CPU mai run krege.....!!! (Non-pre-emptive)

CPU-Scheduling

Arrival time → Time at which process enter to Ready State.

Burst time → Time req. by process to get execute on CPU. (duration → how much time taken)

Completion time → Time which process complete its execution.

Turn-Around time → {Completion time − Arrival time}
→ {12 − 1} = 1 hour = 60 min.

Waiting time → {Turn Around time − Burst time}
• [60 min − 15 min (Ram mai laga)]
= 45 min.

Response time → [{The time at which a process get CPU first time} − {Arrival Time}]

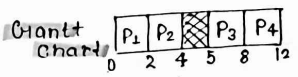
First Come First Serve (FCFS)

(Com.-Arr) (TAT-Burst)

Process No.	Arrival time	Burst time	Completion time	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

In FCFS WT & RT is same because of Non-preemptive

Gantt Chart:



Criteria = "Arrival Time"
Mode :- "Non-Preemptive"

(8 − 6) (Left gantt P_4 value − Arrival Time of P_4)

Average Turn-Around Time = $\frac{2+3+3+6\ (TAT)}{4\ (Total)}$

$= \frac{14}{4}$

Averge Waiting Time = $\frac{0+1+0+2}{4} = 3/4$

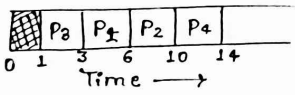
Shortest-Job first (SJF)

Process No.	Arrival Time	Burst Time	Completion time	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4	4	4	14	10	6	6

criteria = "Burst Time"
Mode - Non-Premptive

both same in Non-Preemptive

Gantt Chart



* Alsa process jiska burst time kam hoga hum us process ko select karenge...!!!
(Sabse phele use CPU pe run karwenge)

- Averge TAT = $\frac{25}{4} = 6.25$
- Average WT = $\frac{12}{4} = 3$

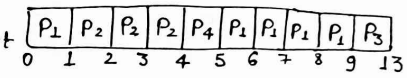
Shortest Remaining Time first (SRTF):-

→ Means (SJF) Shortest Job first with Preemptive)

Process No.	Arrival time	Burst time	Completion time	TAT	WT	RT
P_1	0	~~8~~ ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~	9	9	4	0
P_2	1	~~3~~ ~~2~~ ~~1~~ ~~0~~	4	3	0	0
P_3	2	4	13	11	7	7
P_4	4	~~1~~	5	1	0	0

Criteria = "Burst time"
Mode = "Premptive"
RT = (CPU first time - Arrival time)

Gantt chart



Avg. TAT = $\frac{24}{4} = 6$, Avg. RT = $\frac{7}{4} = 1.75$

Avg. WT = $\frac{11}{4} = 2.75$

Round-Robin (RR)

Criteria = Time Quantum (Burst time)
Mode = Preemptive

* Ek process hai manlo jo ki bus 5 sec tak execute hoga if 5 se jada hua to next ko dega bich mai rok kr aur uska turn last mai bad mai aayega, if 5 se kam hai manlo 3 sec to run hoga the next pe jaiyega...!!!

• Sequence of process in Ready Queue (R/Q).

Process	Arrival time	Burst time	Completion time	TAT	WT	RT
P_1	0	5 ~~3~~ 1	12	12	7	0
P_2	1	4 ~~2~~ 0	11	10	6	1
P_3	2	2 ~~1~~ 0	6	4	2	2
P_4	4	1 ~~0~~	9	5	4	4

Let time = 2

Ready Queue: P_1 P_2 P_3 P_1 P_4 P_2 P_1 (0)

Avg. TAT = $31/4$

Avg. WT = $10/4$

Running Queue: P_1 (1) P_2 (2) P_3 (3) P_1 (4) P_4 (5) P_2 (6) P_1

0 2 4 6 8 9 11 12

Total context switching = 6

Priority Scheduling

(Preemptive)

Criteria = Priority

* Jis process ki jada priority hogi uss process ko hum phele allot karenge

Priority	Process	Arrival	Burst	Completion	TAT	WT
10	P_1	0	5 ~~4~~ ~~3~~	12	12	7
20	P_2	1	4 ~~3~~	8	7	4
30	P_3	2	2 ~~1~~ 0	4	2	0
40	P_4	4	1 ~~0~~	5	1	0

Cond:- (Higher the no. higher the priority)

Gantt chart: P_1 P_2 P_3 P_3 P_4 P_2 P_1

0 1 2 3 4 5 8 12

Avg. TAT = $\frac{12+7+2+1}{4} = \frac{22}{2} = 11/2 = 5.5$

Avg. WT = $\frac{7+4+0+0}{4} = 11/4 = 2.5$

Question:-

Mode:- Preemptive

Criteria:- Priority based

find CT of P_1, P_2, P_3, P_4

Process	AT	Priority	CPU	I/O	CPU
P_1	0	2	1 ~~0~~	5	3 ~~2~~ 1
P_2	2	3	3 ~~2~~ 1	3	1
P_3	3	1	2 ~~1~~ 0	2 ~~1~~	1
P_4	3	4	2	4	1

lowest the number higher the Priority

Gantt chart: P_1 P_2 P_3 P_3 P_2 P_2 P_4 P_3 P_1 P_2 P_4 P_2

0 1 2 3 4 5 6 7 8 9 10 11 13 15

P_3 I/O; P_4 I/O; P_1 I/O; P_3 I/O; P_2 P_4 I/O

P_4 (17 18) → P_4 I/O

$4/18$ =

usage = $14/18$ =

Multi-level Queue Scheduling

Highest Priority: [System process] → RR

Medium [Interactive process] → SJF

lowest [Batch process] → FCFS

→ CPU

Agar higher priority wale jaisa process aaye to dusre ko mauka ni milega (starvation hoga)!!

Multilevel feedback Queue Scheduling

to solve the starvation of (lowest priority process)

Process Synchronization

Process

Cooperative processes
→ Ek ki execution dusre pe affect daalti hai.
because they shared
because they have something common.
(common may be variable, memory, Code, Resources).

Independent
→ No affect on others.

So, that's why Process synchronization used to solve the problem of cooperative processes.

→ int shared = 5 6 4

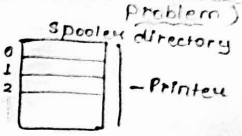
P1
int x = shared; (x=5)
x++; (x=6)
Sleep(1);
shared = x;

P2
int y = shared; (y=5)
y--; (y=4)
Sleep(1);
shared = y;

that condition called **Race condition**

Process Synchronization (Printer Spooler Problem)

```
1. Load Ri, m[in]
2. Store SO[Ri], "P-N"
3. INCR Ri
4. Store m[in], Ri
```



Critical Section → It is a part of pgm where shared resources accessed by various processes. (kind of Cooperative)
→ It is also a placed where shared variables, Resources are placed.

Synchronization Mechanism

4 conditions ko fulfill Karni chaiye.

Primary (that is mandatory):
(i) Mutual Exclusion
(ii) Progress

Secondary:
(iii) Boondeal wait
(iv) No assumption related to Hardware (H/W) speed.

(i) **Mutual Exclusion** → Agar ek sath 2 process hai agar ek critical section mai aagya to dusre ko ni aane dega.

(ii) **Progress** → Ek critical section hai mano kuch common code pada hai jo P_1 & P_2 access krna chahein hai execute krne ko.
Assume critical section is empty & P_1 is intrested to come but P_2 jane he ni deta (Because of P_2 mai kuch aisa code likha hai jo use and one aane se mana kar raha hai) to ye progress nahi hai

(iii) **Bounded wait :—**

Mano P_1 hai critical section to obvious P_2 bahar hoga. P_1 ek bar andar se bahar execute hue, 2 bar 3 bar.. but aisa na ho bi 2 bar ho. agar mano 4 bar P_1 to mauka mila to next P_2 to bhi mile.. !!!

(iv) **No assumption related to H/W speed :-**

Mano ye soln 32 bit system pe chalega 64 bit pe nahi then, this is not a condition.

Lock Variables in OS

Critical section soln using "Lock"

```
do{
    acquire lock
        CS
    release lock
}
```

1. while(lock==1);
2. LOCK = 1

ENTRY CLOSE

3. Critical soln.

4. LOCK=0 exit code

- Execute in user mode.
- Multiprocess soln
- No mutual exclusion guarantee.

Probe:- Is step 1 & 2 kai bad preemption hosta hai.

Test & Set instructions in OS

→ **Critical Section soln :-**

```
while(test_and_set(&lock));
CS
lock = false;
```

```
boolean test_and_set(boolean *target)
{ boolean r = *target;
    return r
}
```

(iii) **Bounded Wait :-**

Mano P_1 hai critical section to obvious P_2 bahar hoga. P_1 ek bar andar se bahar execute hue, 2 bar, 3 bar.. but aisa na hoti 2 bar ho, agar mano 4 bar P_1 ko mauka mila to next P_2 ko bhi mile.. !!!

(iv) **No assumption related to H/W speed :-**

Mano ye soln 32 bit system pe chalega 64 bit pe nahi then, this is not a condition.

Lock Variable in CS

Critical Section soln using "Lock"

```
do{
    acquire lock
    CS
    release lock
}
```

1. while(LOCK==1); 2. LOCK = 1 — ENTRY CLOSE
3. Critical soln.
4. LOCK = 0 — exit code

- Execute in user mode.
- Multiprocess soln
- No mutual exclusion guarantee.

Probe:- Is step 1 & 2 ka beech preemption hosta hai.

Test & Set instructions in CS

→ Critical Section soln :-

```
while(test_and_set(&lock));
CS
lock = false;
```

```
boolean test_and_set(boolean *target)
{
    boolean r = *target;
    return r
}
```

Turn variable (Strict Alteration Method) :-

- Two process solution
- Run in User Mode (means kernel ki need nahi)

	Process "P_0"	Process "P_1"
Entry code →	while(turn != 0)	while(turn != 1)
	CS	CS
Exit code →	turn = 1;	turn = 0;

→ mutual exclusion satisfied
→ Progress also satisfied
→ Bounded waiting also satisfied
→ Hardware independent.

Semaphores

It is a method or tool which prevent "race condition". (one of the method)

→ It is an integer variable which used in mutual exclusive manner by various concurrent cooperative process in order to achieve "synchronization"

Semaphores
- $(-\infty$ to $\infty)$ Counting
- Binary (0,1)

P(), Down, wait ← Entry code
CS
V(), Up, signal, Post, Relase ← Exit code

Entry code:-

```
Down(Semaphore S)        {(down = P(), wait)
{
 S.Value = S value - 1;
  if (S.Value < 0)
   {
    Put process(PCB) in
          Suspended list Sleep();
   }
  else
    return;
}
```

Exit Code :-

```
Up(Semaphore S)        {(Up() = V(), signal..)
{
S.value = S.value + 1;
  If(S.value ≤ 0)
  {
   Select process from suspendend list
       wake up();
   }
}
```

Question:- S=17, 5P, 3V, 1P P=down V=up

17 - 5 + 3 - 1

= 14

Binary Semaphore (0,1)

Down, P, wait

UP, V, ~~Spec~~ Signal.

Entry code:-

```
Down(Semaphore S)
{
  if(S.value == 1)
   {
    S.value = 0;
   }
  else
  {
   Block process & place in suspend list,
          Sleep();
  }
}
```

Exit code:-

```
UP(Semaphore S)
{
  if(suspend list is empty)
  [{
    S.value = 1;
  }
  else{
    Select a proces from suspend list
        & wake up();
  }}
```

(successful operation)

→ If S is is 1 and peform Down that become 0 but when S value is 0 it doesn't become 2 come is blocked i.e, unsecssful operation.

Solution of Producer- Consumer problem

Counting Semaphore → full = 0 = No. of filled slots
→ Empty = N → no. of Empty slots

Binary Semaphores S = 1;

Produce_item(itemp;)
1) down(Empty);
2) down(S);

critical section → Buffer [IN] = itemp;
IN = (IN+1) mod n;

3) UP(S);
4) UP(full);

IN = 3/4

→ Empty = 5
full = 3

N = 8

0	a
1	b
2	c
3	d
4	
5	
6	
7	

Consumer

1) down(full);
2) down(S);
Item c = Buffer(out);
out = (out+1) mod n;
3) UP(S);
4) UP(empty);

out = 0

Let case-1:-

Producer: Empty = 5/4 → UP(full)
full = 3/4 →
Let, S = 1/0 → UP(S)

IN = (3+1) mod 8
= 4 mod 8 = 4

consumer

Now, down full, full = 4,
down(S), S = 0
out = 0 (out+1) mod n
= (0+1) mod 8
= 1
UP(S), S = 1
UP(empty), Empty = 5

Case-2:-

Empty = 5, Full = 3 / IN = 3, OUT = 0 / S = 1

Producer →

Empty = 5
down(empty), Empty = 4
down(S) — here, producer preempt

& consumer come.
then, down(full), full = 3/2
down(S), S = 1/0
out = (0+1) mod 8
= (1)

If here consumer preempt to producer the value S or O doesn't change & that get blocked.

UP(S), S = 0/1
UP(empty), Empty = 4/5

then, come to producer:-
down(S) = 1/0
IN = (3+1) mod 8
= 4
UP(S) = 0/1
UP(full) = 2/3

Readers-Writer Problem:— (Solve)

* Same data pe read write hoga then → problem
R - W → Problem
W - R → Problem
W - W → Problem
R - R → No problem.

```
int rc = 0          (rc = read count)
Semaphore mutex = 1;
Semaphore db = 1;

void Reader(void)
{
  while(true)
  {
  down(mutex);
  rc = rc+1;
  if (rc == 1) then down(db)
      up(mutex)
      [CB]
      down(mutex)
  rc = rc-1
  if(rc == 0) then up(db);
  up(mutex)
  Process_data
  {
  }
```

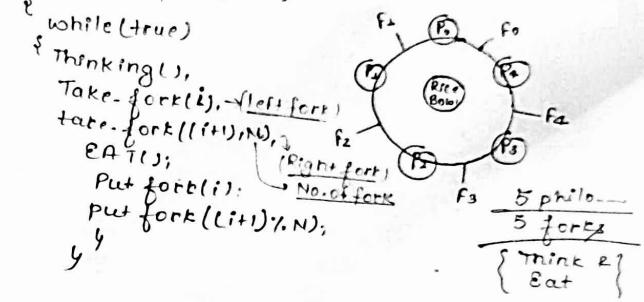
db = 1
mutex = 1
rc = 0

→ 0
→ 0+1 = 1
→ True db = 0
→ Ø 1
[CB R1]

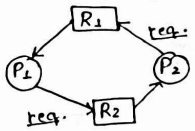
```
void writer(void)
{
  while(true)
  {
  down(db)
  {CB}
  {
  up(db);
  }
}
```

Dining Philosophers Problem.

```
Void philospher(void)
{
  while(true)
  {
  Thinking();
  Take_fork(i);  (left fork)
  take_fork((i+1)%N);  (Right fork)
  EAT();
  Put fork(i);
  Put fork((i+1)%N);
  }
```



DeadLock:



$P_1 \leftarrow R_1$ (but execute when gets R_2) but R_2 already taken P_2)

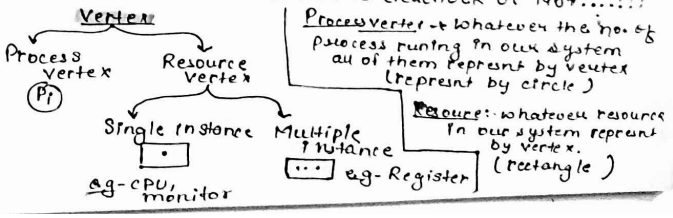
$P_2 \leftarrow R_2$ (execute when gets R_1 but R_1 already taken by P_1)

Conditions of Deadlock :- (i) Mutual Exclusion (ii) No preemption (iii) Hold & wait (iv) Circular wait.

Necessary condition

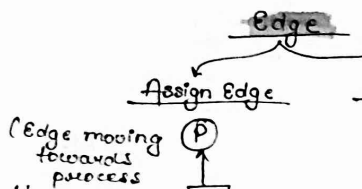
Resource Allocation Graph (RAG)

Method to know that this is deadlock or Not....!!!

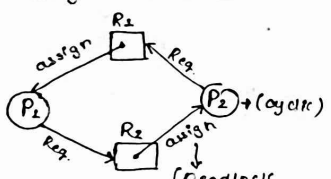


Single Instance:- resources which can be only one like CPU, Monitor.

Multi Instance:- which have multiple resources like register.



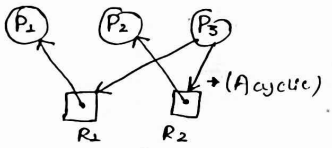
Single Instance



	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

Availability = (0,0)

Multi-Instance



(No deadlock) become no demand of $P_1 + P_2$

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	1	1

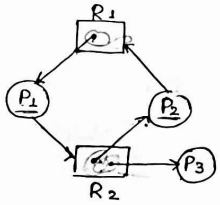
Avail = (0,0) (1,0) (1,1)

→ If RAG has circular wait (cycle)
=> Always deadlock but that's only when there is a single instance Resource.

→ If RAG has No cycle then (No Deadlock) but only in case of single instance.

Multi-Instance Resource Allocation

→ More than 1 instance / Register.



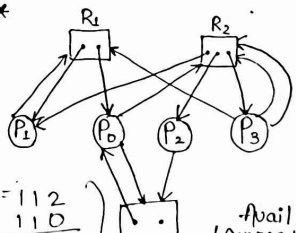
	Allocation		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Availability = $\begin{matrix}0 & 0\\ 0 & 1\end{matrix}$

We can fulfill P_3 (0,0)
then (0,1)
P_2 (0,1)
P_1 1,0
1 1

(No deadlock circular)

*



	Allocation			Request		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	1	2	0

P_2 = 1 1 2
1 1 0
2 2 2
0 1 0
2 3 2

Avail:- (R_1, R_2, R_3) (current) (0,0,1)
P_2 0 1 0
0 1 1
then P_0 0 1 1
1 0 1
1 1 2

(No Deadlock)

* Various Methods to handle Deadlocks :-

(i) Deadlock ignorance (Ostrich method)
(ii) Deadlock prevention
(iii) Deadlock Avoidance (Banker's Algo)
(iv) Deadlock Detection & Recovery

→ Don't want to affect the performance (speed).

→ 4 important condition :-
(i) Mutual Exclusion (ii) Hold & wait
(iii) No-preemption (iv) Circular wait

Deadlock prevention means any of the condition of deadlock not follow then that is Deadlock prevention.
(Any of condition become false)

→ We are trying to give the resources we will check that whether it is safe or not (Safe-Unsafe situation) – Banker's Algo

→ We are trying to detect (by Resource Allocation graph) whether it is deadlock or not & when we detect it we are trying to recover from it

Banker's Algorithm

(also used for deadlock detection)

(Deadlock avoidance)

In the algo we have to give the information to the system beforhand (which process are coming, which process will req. for which resources, how much resources with them req, for how long will they do it) so that system perform safely.

(know the future)

[Max need - Allocation = Remaining Need]

Process	Allocation A B C	Max Need A B C	Available A B C	Remaining Need A B C
P_1	0 1 0	7 5 3	3 3 2	7 4 2
P_2	2 0 0	3 2 2	5 3 2	1 2 2
P_3	3 0 2	9 0 2	7 4 3	6 0 0
P_4	2 1 1	4 2 2	7 4 5	2 1 1
P_5	0 0 2	5 3 3	7 5 5	5 3 1
	7 2 5		10 5 7	

[Total, A = 10, B = 5, C = 7]

{Safe = Deadlock not occur
Unsafe = Occur Deadlock}

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

Process	Allocation	Max	Avail	(max - Alloc) Remaining
P_0	1 0 1	4 3 1	3 3 0	3 3 0
P_1	1 1 2	2 1 4	4 3 1	1 0 2
P_2	1 0 3	1 3 3	5 3 4	0 3 0
P_3	2 0 0	5 4 1	6 4 6	3 4 1
			8 4 6	

($P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$)

(Safe state = No deadlock)

PPT

Booting → start OS when user turn of computer.

Boot seq → initial set of operation that computer perform when power turn on.

Boot loader → typically load OS for computer.

Traps:- Abnormal condition detected by CPU.
- dividing by 0
→ Accessing memory that doesn't exist.

Interrupt → higher priority disturb previous process.
hardware → circuit (And & Or)
Software → Access code that doesn't exist.

Storage Structure

Main-Memory → only large storage media that CPU access directly.

Secondary-memory → extension of main memory that provide large nonvolatile storage.

Caching → (Ram used as cache) hold recently access data.

Context switch → CPU switch to another process.

Process Termination → It execute last statements of OS to delete it.

Independent → connot affect by other process (execution
~~Dependent~~ Cooperating → affect by another process)

IPC → process to communicate & to synchronize their action

Synchronization → way by which process share same memory are managed in an OS.

Buffering → are where main memory used to Store & hold data temporarily.

Dispatch latency → time takes for dispatcher to stop one process & start another running.

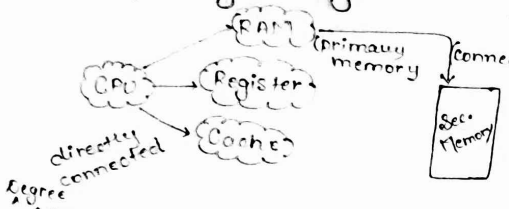
Thuroughput → process that complete their execution per time unit.

Hard real time computing → req. to complete task within a gurantee amount of time.

- Avoid internal fragmentation → by dynamic allocation.
  internal (Static)
  Dynamic (External)
- SRTF (premeetive) / SJF (Non-Preemptive)
- Optimal algorithm →
  (replace page that not use for long period of time)
  Practically not possible → don't know the future.
- LRU - (page replace not use for long time) - cache.

Scheduler → Konsa next process chaiye
D → CPU ka control us process ko dena

Memory Management (method of managing) primary memory



Multi-programming → Multiple process ko sec. memory se RAM kai andar (Main Memory) lana so, that CPU can execute them (efficiency ↑es.)
Utilization also increases.
→ If CPU become at Rest (Ideal) the efficiency ↓es.

Example:- (RAM = 4MB, Process = 4MB, 'K' = I/O operation)
$\frac{4MB}{4MB} = 1$, CPU Utilization = $(1-K)$ K amount of time is (70%)
then, {CPU Utilization = 30%}

Example:- RAM = 8MB, Process = 4MB
No. of processes = $\frac{8MB}{4MB} = 2$
if 1 process take 'K' amount of time then
2 process totatal time = (K^2) - I/O operation
then,
CPU utilization = $(1-K^2)$ (if K = 70%) then,
" = $1-(0.7)^2$
= $1 - 0.49$
= 0.51
= 51%

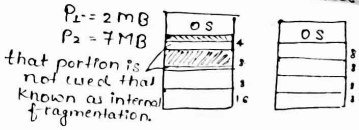
Example:- RAM = 16MB, Process = 4MB, 'k' is the amount of time. ($K^4 = 70\%$)
No. of process = $\frac{16MB}{4MB} = 4MB$
CPU = $(1-K^4)$
= $(1 - 0.2401)$
= 93% ~~= 94% (approx)~~
= 0.76 = 76%

Memory Management Techniques

(Process can store on diff. diff. location of memory.)
(Non-continous allocation of)

- Contiguous (continous address mai store karte)
  - fixed Partition (Static)
  - Variable Partition (Dynamic)
- Non-Contiguous
  - Paging
    - Multilevel paging
  - Inverted Paging
  - Segmented Paging
    - Segmentation

• Fixed Partition → (Static Partition)
- No. of partitions are fixed
- Size of each partitions may or may not same.
(that is know as internal fragmentation)



Advantage :-
→ Easy to implement.

Problem :-
- Limit is fixed in fixed i/p process size.
- Degree of multi programming is also limited.
- Internal fragm.
- External frag. (whenever internal frag. is here External is also)

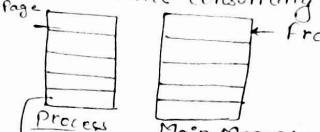
(Dynamic Partition) Vaviable partitioning :-
whenever the processes coming into the RAM only then we are allocating the space to the processes

Advantages:-
i) No internal fragmentation
ii) No limitation on no. of Process (i.e, degree of multi-programming)
iii) No limitation on process size.

Disadvantages:-
i) External fragmentation
(A process cannot span b/co 2 different location) (due to contiguous allocation)
↓
can solve it by compaction (but undesirable)
↓
used to remove external fragmentation
(ii) Allocation / Reallocation is little bit complex.

Methods of Allocation in Memory
(i) First-fit :- Allocate 1st hole that is big enough.
(ii) Next-fit :- Same as 1st fit but start search always from last allocated hole.
(iii) Best-fit :- Allocate smallest hole that is big enough (slow) ↓ (left space min...)
(iv) Worst-fit : Allocate the largest hole. (slow) ↓ (left space max...)

Non-Contiguous :- Memory Allocation.
Time consuming / costly



Page Size = Frame Size

Every process divided into pages.

Paging :- Process ko page mai divide krenge aur page size = main memory frame to process ka page properly fit hojana frame kai andar. phir page ko memory allocate krdenge.
(external fragmentation ko remove krne ko). or faster access to data.

Q.) Process size = 4B, Page Size = 2B.
Process/No. of pages = 4B/2B = 2

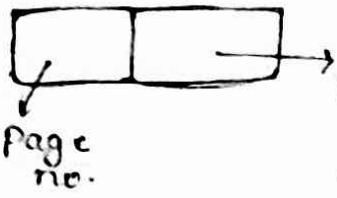
Q.) main memory size = 16B
frame size = 2B
No. of frames = 16B/2B = 8 frames

Mapping → To address CPU generate kari jata hai (use) that is not the absolute address
Memory management unit → CPU jo address generate kari rahi hai wo actual mai absolute address mai use convert kar rahi hai...!!
Memomury management unit (MMU) use page table.

Page Table:- contain the frame numbers where that the page actually present in main memory

- CPU always work on logical address (generate)

made of 2 things



Question:-

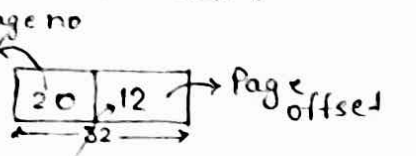
Logical Address space = 4 GB
Physical " " = 64 MB
Page Size = 4 KB = frame size

} find, No. of pages, No. of frames, No. of entries in page table, size of page table?

Solution:- Memory is byte addressable:-

LA = $2^{?}$ 4 GB

= $2^2 \times 2^{30} = 2^{32}$



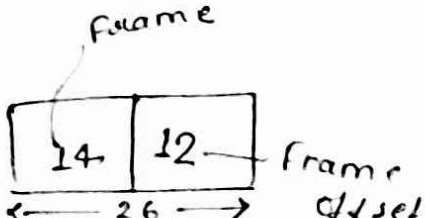
Page size = 4 KB

= $2^2 \times 2^{10} = 2^{12}$ → 12 bit in page offset

→ No. of pages = 2^{20}

→ No. of frames:-

Physical address → 64 MB
→ $2^6 \times 2^{20}$ B
→ 2^{26}



→ No. of frames:- 2^{14}

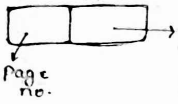
→ No. of entries in page table = No. of pages in process

So, Page no. = 2^{10}

No. of entries in page table = 2^{10} (1 million)

Page Table:- contain the frame numbers where that the page actually present in main memory

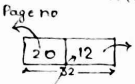
- CPU always work on logical address (generate) made up of 2 things



Question:-
Logical Address space = 4 GB
Physical " " = 64 MB
Page Size = 4 KB = frame size
Find, No. of pages, No. of frames, No. of entries in page table, size of page table?

Solution:- Memory is byte addressable:-

LA = 2^{32} 4 GB
= $2^2 \times 2^{30} = 2^{32}$

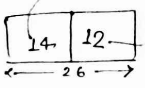


Page size = 4 KB
= $2^2 \times 2^{10} = 2^{12}$ — 12 bit in page offset

→ No. of pages = 2^{20}

→ No. of frames:-

Physical address → 64 MB
→ $2^6 \times 2^{20}$ B
→ 2^{26}



→ No. of frames:- 2^{14}

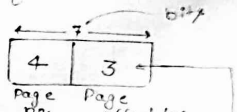
→ No. of entries in page table = No. of pages in process.

So, Page no. = 2^{10}
No. of entries in page table = 2^{10} (1 million)

Size of page table: No. of value × size
→ No. of pages × bits in pages (frame)
→ $2^{20} \times 14$ bits

Question:- System which has LA = 7 bits, PA = 6 bits, Page-Size = 8 words then calculate no. of pages & no. of frames.

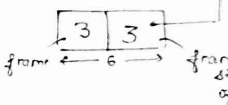
→ Logical address:- 7 bits
page size = 8 words
= 2^3



No. of pages = 2^4
= 16

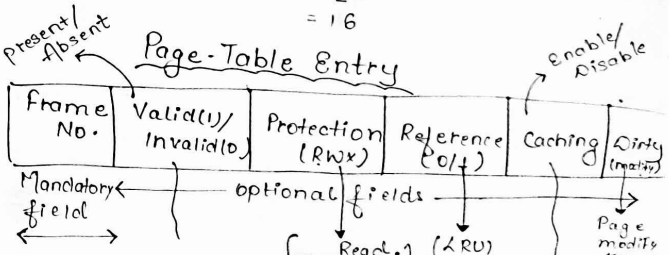
Physical address:- 6 bits

No. of frames = 2^3
= 8



No. of entries in process = no. of pages in process
= 2^4
= 16

Page-Table Entry



Two-level Paging (Multilevel Paging)

Example:-

Physical address space = 256MB $2^8 \times 2^{20} = 2^{28}$ | Page no. 16 | offset 12 |

Logical address space = 4GB = $2^2 \times 2^{30} = 2^{32}$ | 20 | 12 |

Frame size = 4KB = $2^2 \times 10 = 12$

Page table entry = 2B

PT size = $2^{20} \times 2B$ = 2MB

Advantage:- Time kam lega searching mai

~~Inverted PA~~

(i) Each process has its own page table. } Disadv.

(ii) Page table be in M/m

Inverted paging

Ek global page table banana. Pure OS mai bus ek hi page table maintain kreng.

Frame no.	Page no.	Process ID
0	P_0	P_1
1	P_1	P_2
2	P_2	P_1
3	P_1	P_3
4	P_3	P_2
5	P_2	P_3

* normally phele page no. aata phir frame but in Inverted paging phele frame no. aayega phir page no.

(entry = no. of frames of MM)

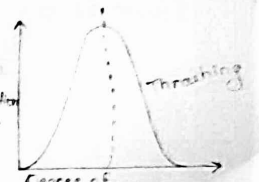
Disadv:-

Searching time ↑ es

→ Each process has its own page table.
→ PT will be in MM.

Thrashing:-

To much fault • page hit also ↑es



and and time handle se main memory (page) lane mai lage

& CPU utilization ↓es

Remove of thrashing:-

(i) ↑es main memory (less possible)

(ii) long term scheduler (jada se jada process to ready state mai lana..)

Segmentation

~~is a process in which we divide~~ is a method in which process will be divided into the parts & then put them in MM.

Segmentation & Paging

↳ Segmentation is a method in which process will be divided into the parts & then put them in MM.

↳ Paging is which process can be divided into fixed sized pages & then put them in MM.

• bina jane page ko ki usmai kya likha hai use divide kre main memory mai dal degi.

~~Problems~~:- Problem:-

(i) Page fault

Segmentation :- Also he divide ni krega.
Segments mai krega like ek main(), ek add(), Subs, Stack, sqrt (convert that in segments)

→ Segments size may or may not same.
(various size)

[logical address convert → Physical address
It helpto convert | Memory Management Unit]

→ Paging size same but Segmentation size different.

Overlay

is a method in which large size process can be put into the main memory.
Main memory se size bada hai phir bhi put kar skte by overlay.

• Used in Embedded
↓
fixed functionality pe work krta. (main memory fixed)

→ division must be independent.

Assembler:- used to convert source code into object code.

Question 2 Assembler
Pass 1 :- 80KB , pass 2 : 90KB
Symbol table : 30KB
Common routine = 20KB
Overlay driver size = 10KB

80KB (both pass)
90KB
30KB
20KB
10KB
230KB

Pass 1
80KB
30KB
20KB
10KB
140KB

Pass 2
90KB
30KB
20KB
10KB
150KB

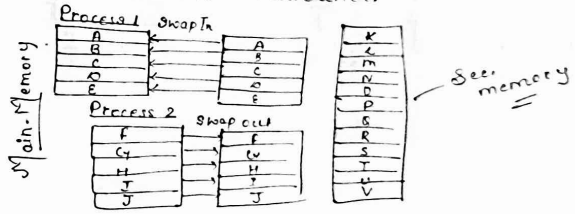
Ans:- 150KB

choose maximum

= p (page fault service time) + (1-p) (main memory access time)
↓ (msec) ↓ most in ns.

Demand Paging

Demand the page from main memory.
Process reside in sec. memory & pages loaded only on demand not in advance.



Page Replacement → (to see page fault)

(i) FIFO (ii) Optimal Page replacement
(iii) Least Recently Used (LRU).

(i) FIFO- Reference - 7,0,1,2,0,3,0,4,2,3,0,3,1,2,0

f3			1	1	1	1	0	0	4	4	4	3	3	2	2
f2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	*	*	*	*	Hit	*	*	*	*	*	*	Hit	*	Hit	

Page Hit = 8
Page fault = 12

Hit Ratio = $\frac{\text{No. of hits}}{\text{Reference no.}} = \frac{3}{15} \times 100 = 20\%$

Miss = $\frac{12}{15} \times 100 = 80\%$

* No. of frame jada hone se fault page jada & hit kam hoga → Belady's

(ii) Optimal Page Replacement

Replace the page which is not used in longest dimension of time in future.

Ref:- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f_4			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_3		1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
f_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	7	7	7	7
	*	*	*	*	Hit	*	Hit	*	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit

Hit = 12 $\frac{12}{20} \times 100 = 60\%$

fault = 8 $\frac{8}{20} \times 100 = 40\%$

(iii) Least Recently Used (LRU)

Replace the least recently use page in past.

• Ref :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f_4			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_3		1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
f_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	7	7	7	7
	*	*	*	*	Hit	*	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	*	Hit	Hit	Hit

Hit = 12
fault = 8

(speed less than FIFO)

• EMAT = Hit (TLB + MM) + Miss (TLB + MM + MM)

(iv) Mostly used Used (recently) (replace the most recently used page in past).

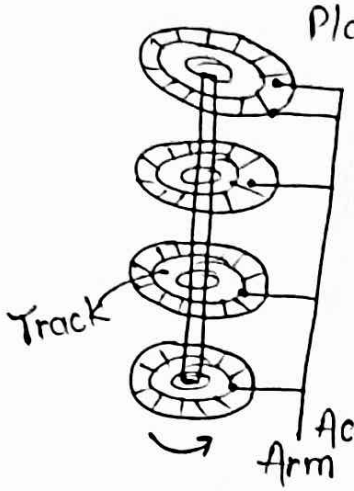
Ref:- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f_4			2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	0	0
f_3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f_2	0	0	0	0	3	0	4	4	4	4	4	4	4	4	4	4	4	4	4
f_1	7	7	7	7	7	7	7	7	7	7	2	2	2	2	2	2	2	2	2
	*	*	*	*	*	*	*				✓	✓	✓	✓					

Hit = 8
fault = 12

• EMAT = Hit (TLB + MM) + Miss (TLB + MM + MM)
↓
PT (page table)

Disk Architecture



Platters → surface → Track → sectors
2 (upper/lower)

• Platter = 8, Track = 256, sector = 512
surface (2) upper-lower, Data = 512 KB

Disk Size = platter × surface × Track × sector × Data
= $8 \times 2 \times 256 \times 512$ KB
= $2^3 \times 2^1 \times 2^8 \times 2^9 \times (2^9 \times 2^{10}\ B)$
= 2^{40} B
= 1 TB

No. of bits req. to represent Disk size = 40

→ **Disk Access Time :-**

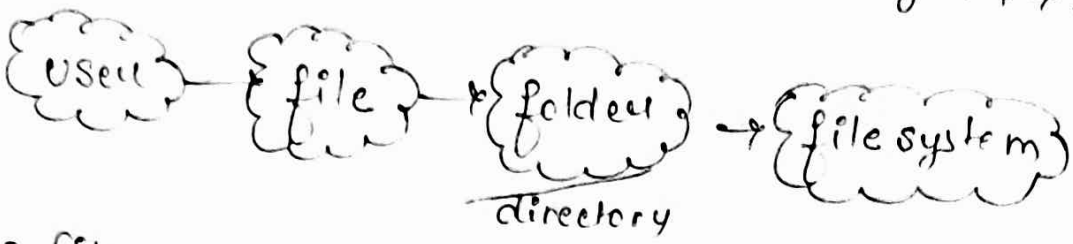
(i) Seek Time → time taken by R/W head to reach desired track.
(ii) Rotation time :- Time taken for 1 full rotation (360°)
(iii) Rotational latency :- Time taken to reach desired sector (half of rotation time)
(iv) Transfer Time = Data to be transfer / Transfer rate ; Transfer rate : (Data Rate)

→ (No. of heads × (surface) × capacity of 1 track × No. of Rotations in one second)

Disk Access Time = ST + RT + TT + CT + ST (Qtime)
(count time)

(mange file) **File-System** (It is software)
↳ manage how data will be stored & how data will be fetch)

* Harddisk provide permanent storage. → file system



• **file** → a collection of data or collection of information.

Operations on files

1) Creating
2) Reading
3) writing
4) Deleting — delete along with attributes
5) Truncating → Attributes not delete
6) Repositioning

file attributes

1) Name
2) Extension (type).
3) Identifier
4) location
5) size
6) Modified data, create data
7) Protection / Permission
8) Encryption, compression

File Allocation Method

- Contiguous Allocation
- Non-contiguous Allocation
  - → linked list Allo...
  - → Indexed Alloca...

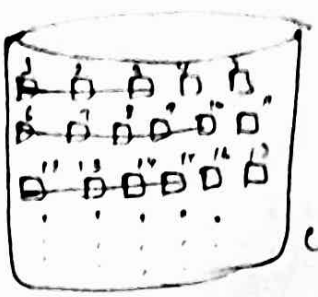
Purpose

(i) Efficient Utlization Disk (ii) Access faster. (iii) performace ↑

Contiguous Allocation → files block ko continous way mai rakh rhe hai. (storage mai)
e.g - student sit at school in cont. manner

Directory → use to store all the informations.

file	start	length
A	0	3
B	6	5
C	14	4



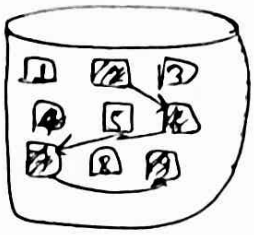
Adv
(I) Easy to implement
(II) Excellent Read performance. (Sequential or direct both me hai.. ☺)

Disadv.-
(I) Disk will become fragmented
(II) Difficult to grow file.

linked-list Allocation

→ comes under Non-contiguous Allocation.

Adv:.
(I) No External frag.-
(II) file size can increase.

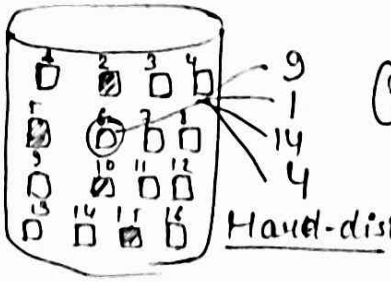


Disadv.-
(I) Large seek time
(II) Random Access / Direct Access Difficult.
(III) Overhead of pointers.

Indexed-Allocation

Directory

File	Index blk
A	6



Adv
(I) Support direct access
(II) No External frag.-

Q → Har file ka alag index hoga

Disadv -
(I) pointer overhead
(II) Multilevel Index.