OPERATING SYSTEM

(2)

PAGE NO.
DATE

Raghul Bachhiche

Basic Concepts of Operating System :-

CPU

GPR's	R_1	ALU	→ Arithmetic Logic Unit
	R_2		
	R_3		
	R_n		
Program Counter	PC		
Stack Pointer	SP	CU	→ Control Unit
	Accumulator		

(i) OS is a kind of manager who manages all processes under the system.
(ii) It is interface b/w user & system.

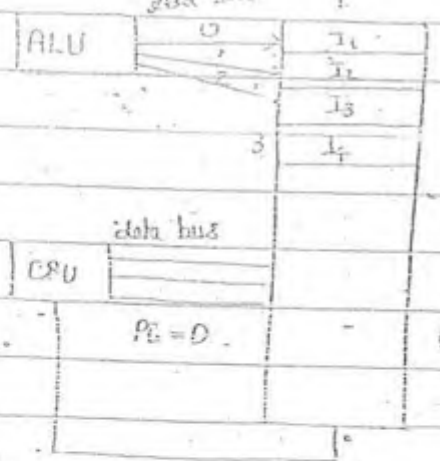
Program → (i) Appln Program
→ (ii) System Program

(iii) Every CPU has set of instruction.

Instruction :-

MOV
ADD

Eg :-



Steps :-
(i) Fetch the instructions
(ii) Decode the instructions
(iii) Read operands
(iv) Execute instructions
(v) Result stight back.

Instruction cycle

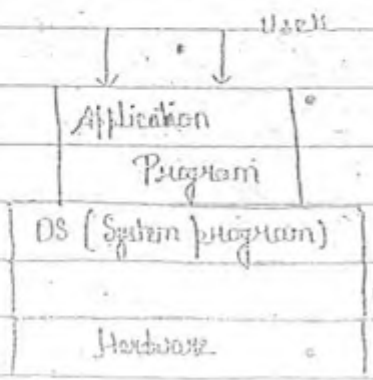
* Default Reg is accumulator.
* When fetch instruction complete the PC increased by one.
* PC always point next instruction
* CPU checks interrupts after the completion of an instruction cycle and before the start of next instruction.

PAGE NO.
DATE :

Program

Application Program (Don't directly interact with the hardware)

System Program (Directly interact with the hardware)



* OS is System Program
* Interpreter, Compiler are also System program
* OS uses device driver who actually easier the hardware device driver is also a system program.
* OS is an interface b/w application program (user) and hardware.
* OS provides an environment to the appln program so that appln program can create, efficiently and effectively.
* OS implements the function which is required to manage the appln program and which is required to interact hardware.

$f_1()$
$f_2()$
$f_3()$ - Kernel.
:
$f(n)$ -

Interf Appln & O.S

PAGE NO.
DATE :

* Kernel Implement critical function under OS.
* kernel is a set of function which implement critical activities of OS.
* Notepad is an appln program.

Implements the → | f1() | f2() |

Headline of function

↓

The code displays the character in monitor.

```
main()
{
  int j=2;
  printf("%d", i);
  fun(i,2)
}
int fun(int x, int y)
{
  return x+y;
}
```

* System Call or API (Application Program Interface)

When appln prog. wants to use services of OS then ... call

* System Call is interface between appln program & ...
* ~~Functions are called system call in unix, linux~~
* In Windows VC++ language is used to use sys...
* In linux C language is used to call system call
* ...

Interface between Application program & O.S

with UNIX or LINUX

System calls

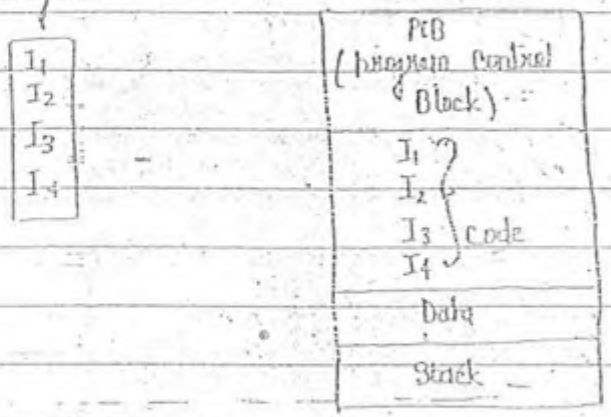
or

Application program interface → we ...

PAGE NO.
DATE :

Process :—

* Program is a set of instruction and process is a combⁿ of PCB (program control block)



* PCB is data structure created by OS when a program goes from execution.
* Program under execution is called process.

PCB

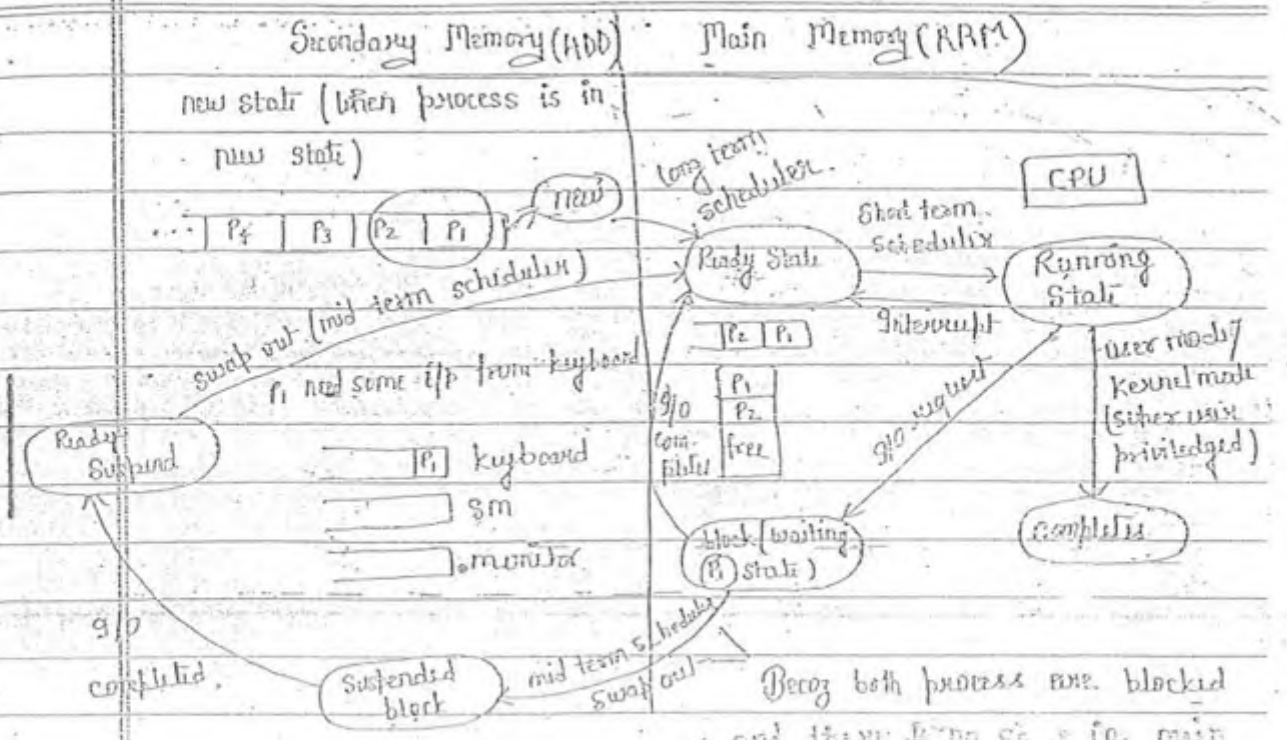
Process Identification
Registers Values
Control Information

Process Identification :— when program wants to execute OS gives an unique id to program, which is called.

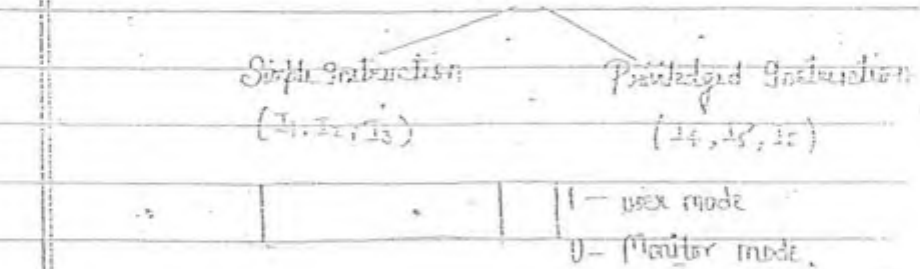
Process Id (PID) which is non negative number.

Control Information :— priority of program
memory related info
file related info
Signal related info
State of process.

PAGE NO.
DATE :



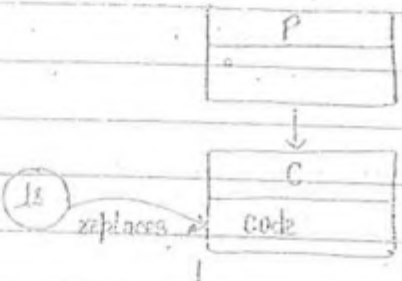
* Long term scheduler decides that which process should go to the main memory from secondary memory.
* Short term scheduler decides which process or program will go for execution.
* It maintains a queue for every device
* The freq of long term is less (10 times/sec) means it more time long and less than short term (50 times sec) short term scheduler freq is very high (100 times/second)
* Instruction can is divided into two categories —



PAGE NO.
DATE :

* 9 can be negative.
* ls command :- list of directories and files.

```
main()
{
  int i = 0
  if (fork() == 0)
  {
  execl ("/bin/ls", ls, NULL);
  printf ("Inside Child");
  }
  else { wait (NULL);
  printf ("Inside parent");
  }
```



→ show list of directory & files.

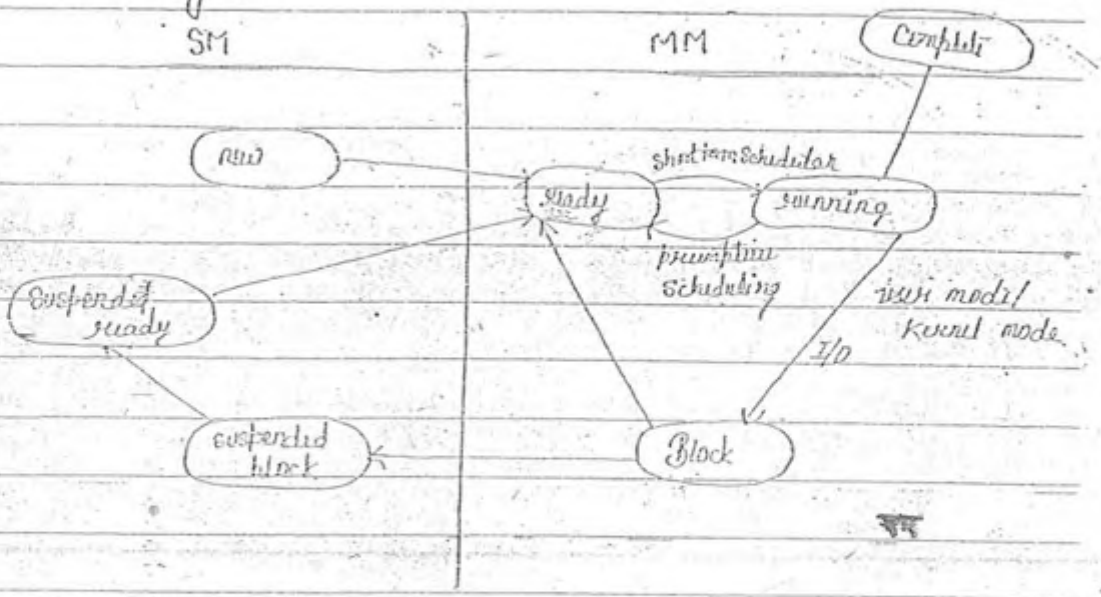
Inside parent

* wait(NULL) :- Parent wait till all child have completed.
* One process of program can access address of another process but thread can
* CPU, main - Physical resource
  time - Logical resource

28th June 08

PAGE NO.
DATE :

Context Switching :—



Process P_1 :—

Addresses	Instructions	
0	I_1	MOV R_1, #5 // $R_1 \leftarrow 5$
1	I_2	MOV R_2, #10 // $R_2 \leftarrow 10$
2	I_3	ADD R_1, R_2 // $R_1 \leftarrow R_1 + R_2$
3	I_4	MOV R_4, R_1 // $R_4 \leftarrow R_1$

Process P_2 :— I_5

I_6

I_7

Stp 1 :— PC = 0

$R_1 = 5$

Stp 2 :— PC = 1

$R_2 = 10$

Stp 3 :— PC = 2

$R_1 = 15$

Interrupt occurs so P_1 will go back to readyqueue now process is selected by short term scheduler for next execution. The complete value of R_1 will be change so when P_1 will wrong value go to the R_4. So when a process is it value are saved inside PCB.

PAGE NO.
DATE :

* This is done by dispatcher module of OS.
* Each process has its own PCB.
* When P interrupted

PCB (P)

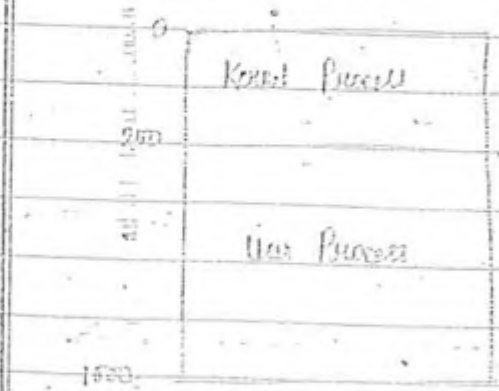
R_1=11
R_2= PC=3
R_3=

These values are called context

* Which when a process is block its context is saved and when program resume values are reloaded.
* This process is duty of the dispatcher.
* This process is known as context switching.

Types of Operating System :—

1. Batch Operating System
2. Multiprogram
3. Time Sharing
4. Multiprocessor (Mainframes)
5. Distributed OS
6. Clustered OS
7. Real Time OS
   - ↳ Hard Real time OS
   - ↳ Soft Real time OS



PAGE NO.
DATE :

Address space for kernel → 0 - 2vo

" " " user → 2o1 - 1ooo

★ If any program try to access the space of kernel it is immediately killed by OS.

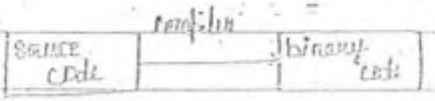
★ Kernel can use address space of user.

★ Compiler is platform dependent. Platform is comb^n of os & cpu, i/o devices.

★ Firstly program was in secondary memory when user want to execute it loader loads it into main memory.

★ .exe files are platform independent

★ Compiler :-



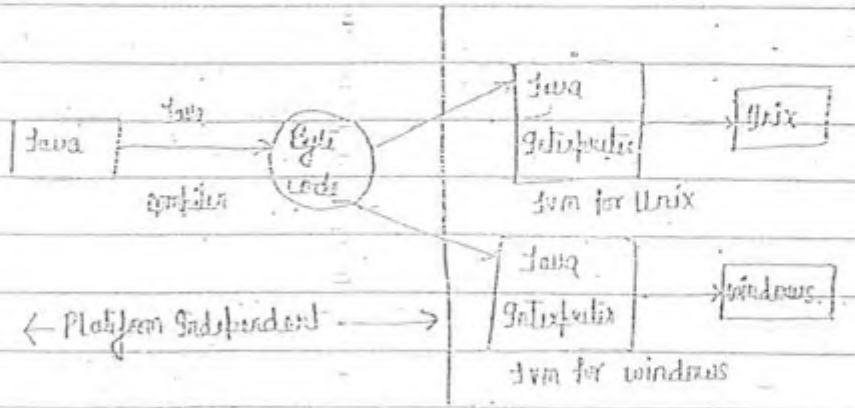
★ Interpreter :-

★ C is compiled language.

★ Java has both compiler & interpreter.

★ Compiler is time efficient/fast.

★ Platform Independency of Java.



PAGE NO.
DATE :

★ Javac – compiler

Java – interpreter

★ JRE support files which are necessary for interpreting program.

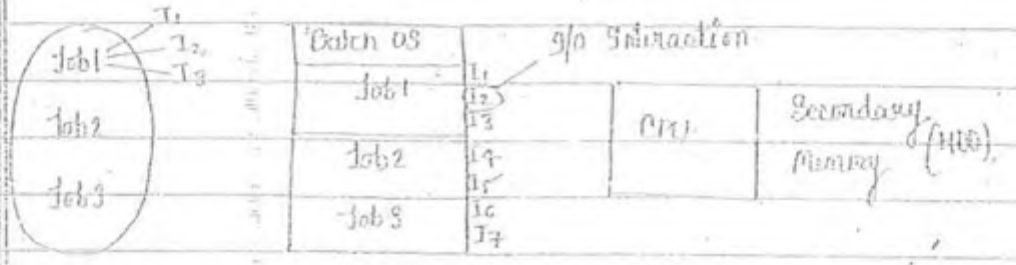
Goal of OS :–

(i) Resource Utilization (In Unix)

(ii) Convinence to user (In Windows)

(iii) provide environment to user process so that they can execute efficiently and effectively.

1. Batch Operating System :–



★ Same type of jobs are combined in one unit known as batch.

★ Value inside it is known as control.

★ Responsibility of BOS is change control from one job to another job.

★ It is simple but it have limited facility.

★ Problem :– (i) Low CPU utilization.

(ii) CPU goes to idle state if interaction need of interaction.

2. Multiprogram OS :–

Important Characteristics :–

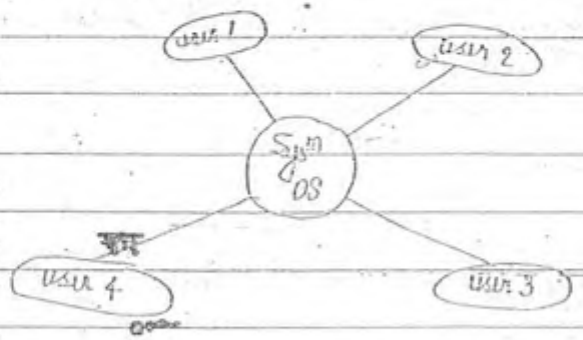
(i) More than one process can be loaded in main memory (concurrent execution).

PAGE NO.
DATE :

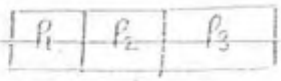
(iii) If any process requested for I/O activity, block the process and switch the control to another process.

* Problem :- It doesn't support multiuser capability.

3. Time Sharing OS :-

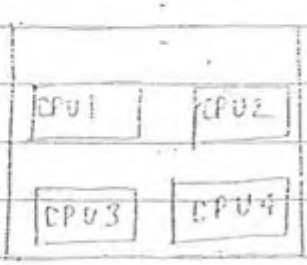


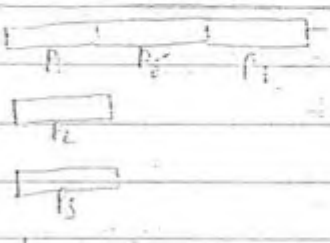
Kernel	
P_1	user 1
P_2	t = 5ms
P_3	
P_4	user 2
P_5	t = 5 ms
P_6	user 3
P_7	t = 5 ms.

* It has capability of multiprogram OS
* It assign time quantum
* Execution is done in round robin fashion
* DOS support partial multiprogramming. It doesn't support full multiprogramming.
* 

Concurrent execution
* Time sharing is multitasking but at a time one program is executed.

4. Multiprocessor :-



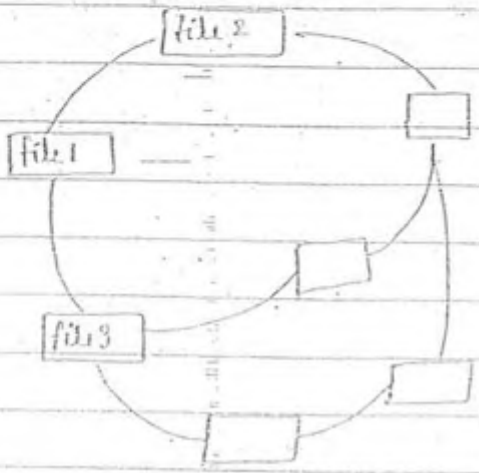


Parallel Processing.

PAGE NO.
DATE :

★ It is tightly coupled system.
★ Single processor system support concurrent execution but multiprocessor system support both parallel and concurrent execution.
★ Unix is multiuser OS.

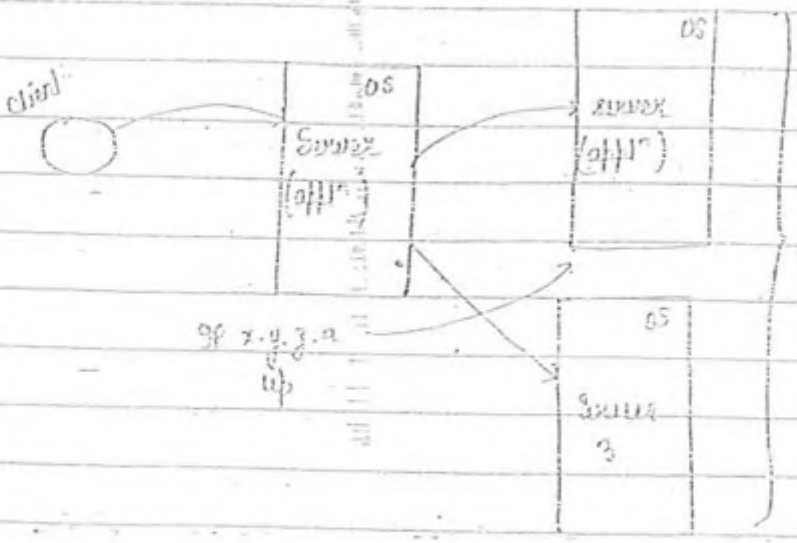
5. Distributed Operating System :-



★ P3 can run on sys^m 3 and the result is sent back to the sys^m.

6. Clustered Operating System :-

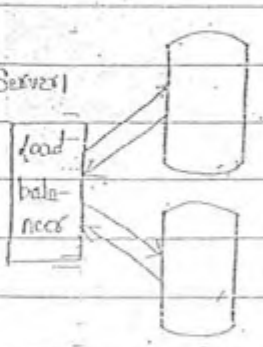
1st Clustered OS :-



PAGE NO.
DATE :

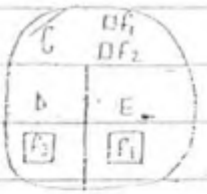
* If server 1 is down then Server 2 will be up immidiately and this is the responsibility of OS. OS will regularly check that server 1 is down or not. IP address and applications will move to the server 2.

2nd Cluster OS :—

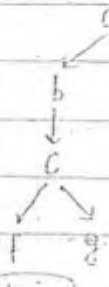


* load balancer is combⁿ of h/w and s/w
* If any request comes load balancer collect the information about the load of each system then the request is sent to the server which has less load.
* If server is down then hdd of it can be shared so SAN is used.
(SAN - Server Area Network)

Mounting :—



* Abstraction means hiding technical details from user.
* OS create DS for accessing file and that DS is called file structure.
* FAT - 16/32, NTFS file system - Windows.
ext - 2/ext - 3 → Linux.
* Floppy ⇒ this file can be accessed only when the file structure of floppy is mounted as file system of OS.



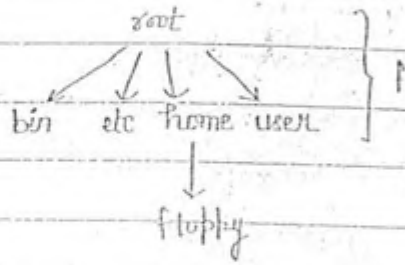
file

PAGE NO.
DATE :

Command :—

mount ⟷ floppy

⟷
floppy

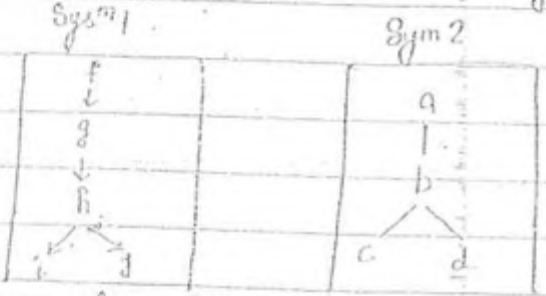


path — /home/a/b/c/f/file1.

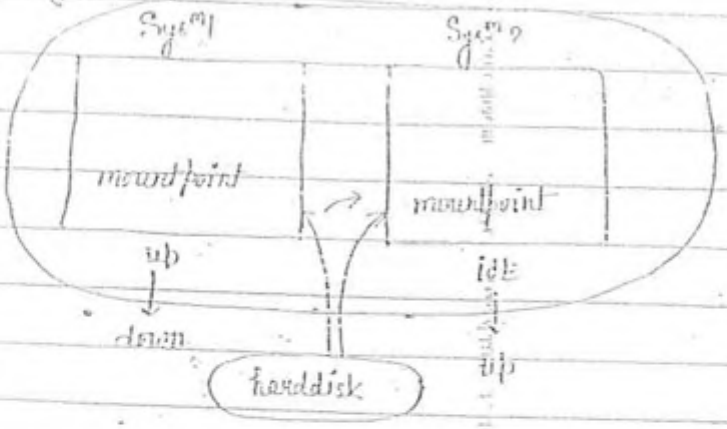
Before removing the device we have to unmount.

unmount ⟷ floppy

* Can mount other sys^m HDD to another system.



* Storage Area Network :—



* Plug and play is process of connecting two devices whether, windows automatically mount the devices.

PAGE NO.
DATE :

4. Real Time OS :- There are two type of OS :-

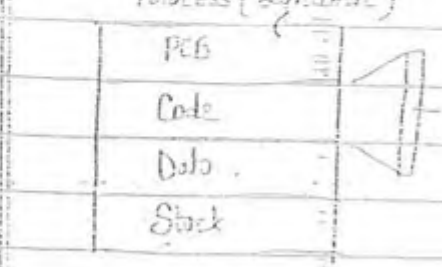
↳ Hard RTO

↳ Soft RTO

* RTOS are specific OS which are implemented depending upon goal.
* Designing and implementation is hard in hard RTOS and designing and implementation is easy in easy in soft RTOS.
* Hard real time OS :- each and every activity is bounded with delay. Kernel is bounded with delay. Appln program is bounded with delay.
* goal of RTOS is time utilization.
* If a process has time 5 sec then if it is completes with time it is success, if it takes more time than it will fail.
* Generally secondary memory is not available in hard real time OS because it takes more time (in ms) to copy in main memory.
* These OS are installed in Rom.
* use in Rocket launching.
* Soft Real Time Operating System :- It is like priority scheduling.
* If priority is higher than the process is critical and it is exited before lower priority processes.
* If any lower priority processes is running and any higher priority process is blocked and sent to ready queue the higher priority OS execute first.
* Embedded OS is a type of soft RTOS.

4th July 09

Threads :- light weight process.



PAGE NO.
DATE :

$P_1, P_2, \ldots P_n$

Process

Code		
Data		
PC register	PC register	PC register
Stack	Stack	Stack

In a process P the code is :-

```
main()
{
int a=5
int b=6
fun1();
Fun2();
}
fun1(){
}
fun2(){
}
```

All codes are executing independently

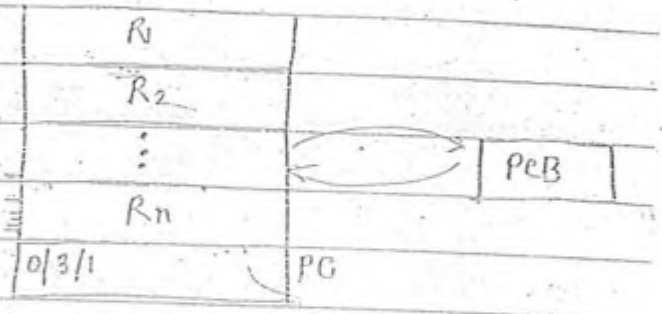
Thread of execution = sequence of execution

After executing main function partially control goes to the fun1, after executing fun1 partially control goes to the fun2 Then this type of execution is called multithreading.

A Program can be compiled without main function but it can't execute w/o main function bcoz execution start from main function.

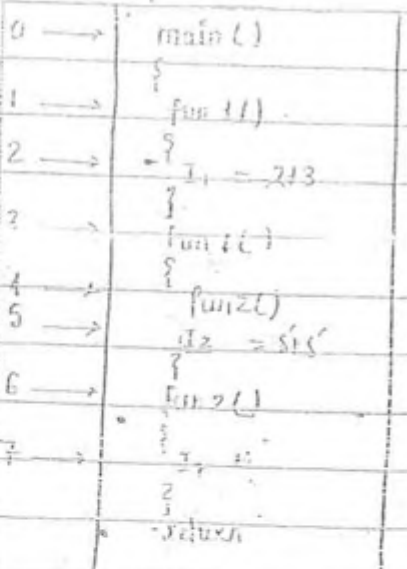
PAGE NO.
DATE :

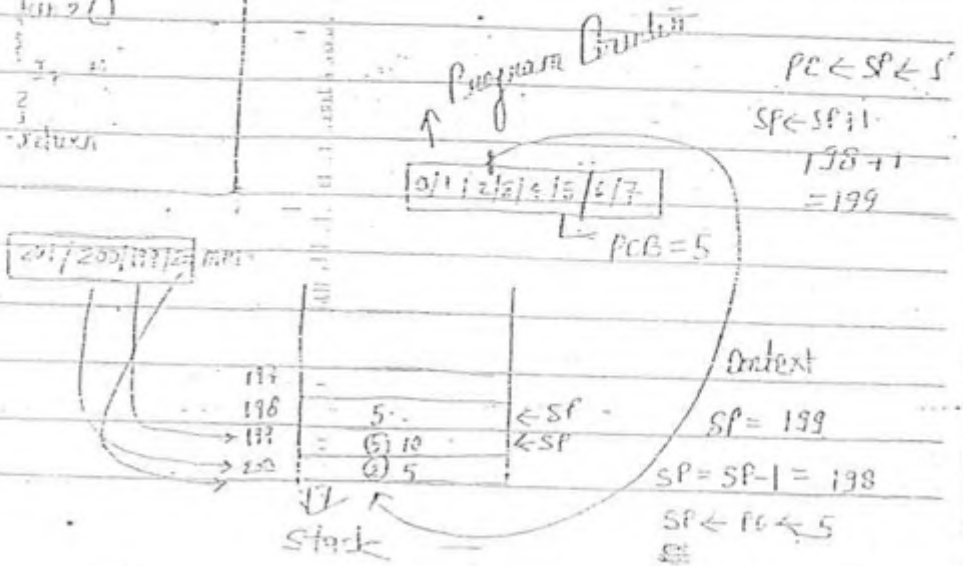
```
main ()
{
0 → I1
1 → I2
   ?
   fun1()
   {
3 → I3
4 → I4
   ?
   fun2(){
5 → I5
6 → I6
7 → I7
   ?
}
```



* At the time of context switching we need to store the i/p, o/p so each thread have PCB.
* Stack is never shared between the thread.

Stack :-





PAGE NO.
DATE :

* Before control transfer from main fun1 CPU will perform some action.

SP ← SP−1	⇒ subroutine call
(SP) ← PC	execution.

Return instruction :—

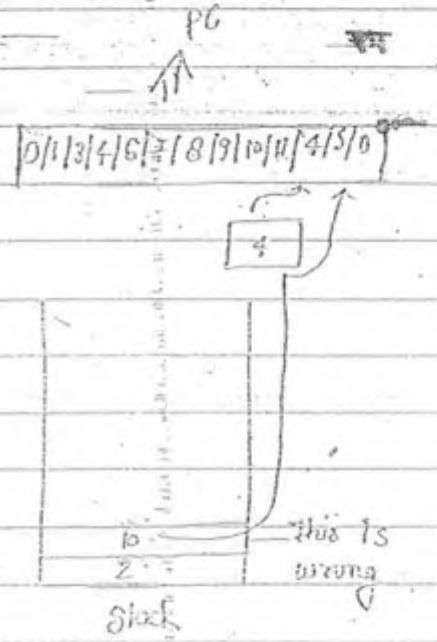
PC ← SP

SP ← SP+1

* exit(0) means about the program.

Why each process has its own stack :—

```
main()
{
 fun1()
 I1
{
fun1()
{
 I2 ——→ context switching
 I3
{
fun2()
{
 I4
 I5
 fun3()
 I6
{
```



When Context switching Occure :—

(1) Interrupt (data xfer from HDD, press key)

(2) I/O request

(3) Program Completed.

* Addresses allocated to a process are shared between the threads.

* PC are different because are need to save context.

PAGE NO.
DATE :

* Code will different be same but each thread has different portion.

Common in threads :-

(i) Code

(ii) Data

(iii) Resource (file/ i/o device / n/w device / memory / alarm / signals)

* Every thread has its own id, pc, registers value and stack.

* Creation of thread is easy as compared process, management of thread is also easy as compared process. So it is called light-weight process.

Benefits of Threads :-

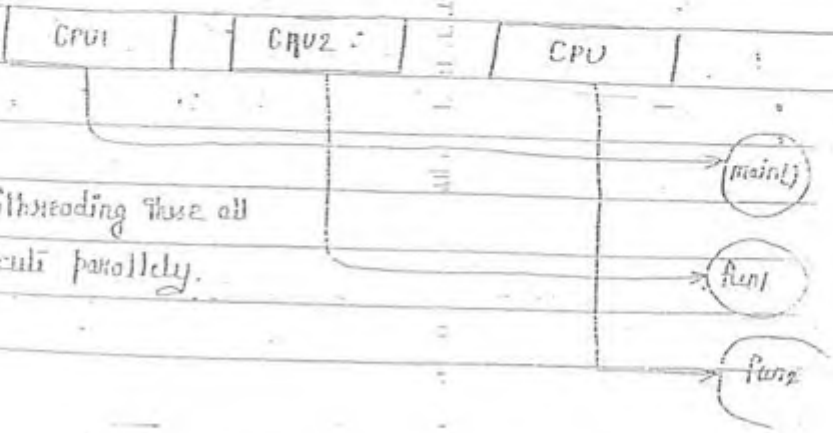
1. Responsiveness
2. Economy
3. Support Multiprocessor Architecture
4. Resource Sharing.

* In the case of thread creation OS has to create very less and small data structure. It also need not to allot memory and resources.

* Responsiveness :- In multithreading response time is needed and it is very less.

* First concept of thread is given by Microsoft and Windows is based on threads.

Support Multiprocessor Architecture :-



In multithreading these all can execute parallely.

PAGE NO.
DATE :

Thread Management :—

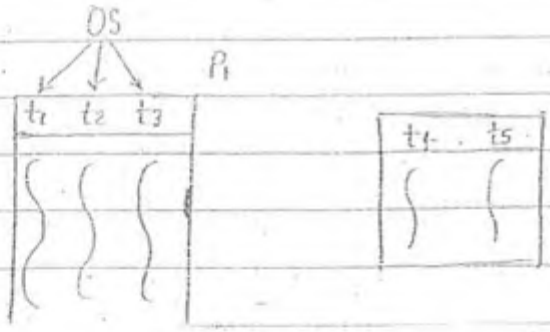
(i) Creation of Thread
(ii) Deletion of Thread
(iii) Scheduling of Thread
(iv) Context save and restore of Thread.

Thread Management

Kernel — User level Display
(If all these responsibility are done by user level Threads)

* Kernel level thread creation requires more time and user level thread creation requires less time.

Drawback of user level thread :—



If Thread are created by user level library then kernel doesn't know about that Thread. If process P_1 get resources $R_1 R_2 R_3$ that Thread will share that resources.

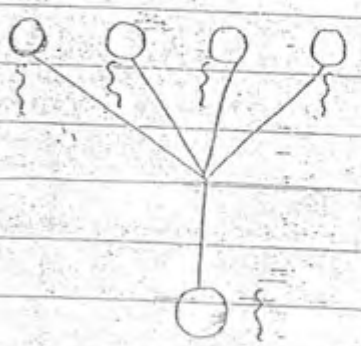
* Owner of resources are processes not Thread.
* When a Thread execute an i/o instruction that control go back to the kernel, kernel doesn't know anything about the process so it will block whole process.

PAGE NO.
DATE :

System :-

(i) user level thread
(ii) Kernel level thread
} both type of thread are possible in system.

user level thread -



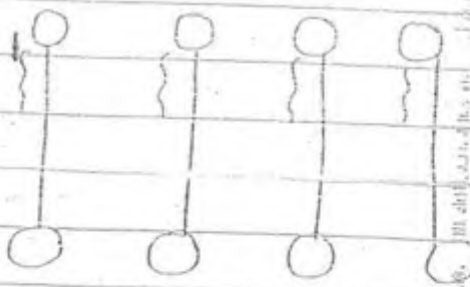
Kernel level thread -

When any thread (user level) wants to use as it connect to kernel.

Mapping :- There are three type of mapping b/w kernel and user :-

(i) one to one mapping
(ii) many to one mapping
(iii) many to many mapping

1. one to one mapping :-

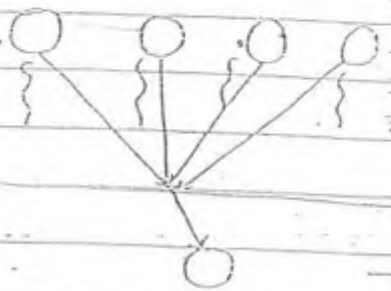


→ user level thread

→ kernel level thread

If OS has the ability to create two thread only then after it OS will not create any more thread.

2. Many to one mapping :-

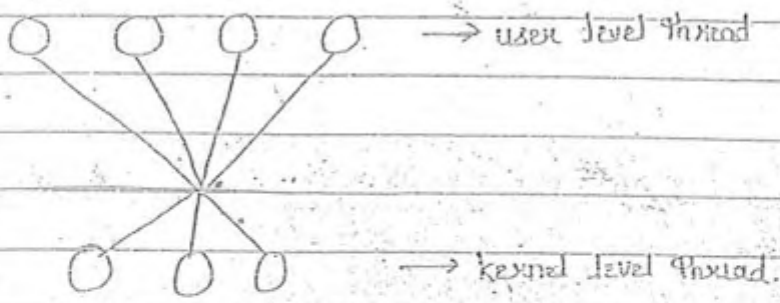


→ user level thread

→ kernel level thread

PAGE NO.
DATE :

3. many to many mapping :-



Example of Thread :-

```
Class TestClass extends Thread
{
  public void run()
  {
    int a = 2;
    int b = 3;
    int c = a+b;
  }
  System.out.println(c);
  }
  class mainclass
  {
```

→ public static void main (Strings args[]) { }

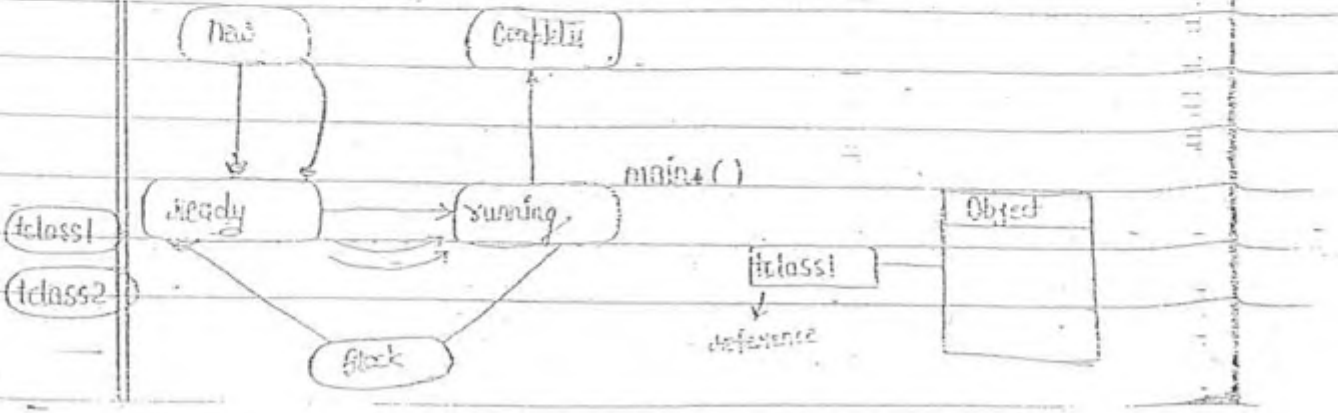
→ TestClass tclass1 = new TestClass();

→ TestClass tclass2 = new TestClass();

tclass1.start();

tclass2.start();

}



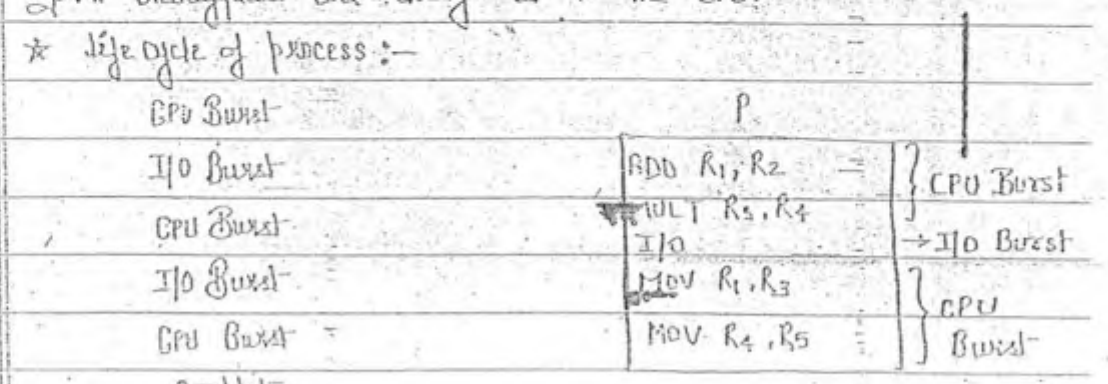
PAGE NO.
DATE :

* which thread will exute next is decided by scheduler.

Process Scheduling :- (short term scheduler)

* Responsibility of short term scheduler is to choose process from readyqueue and assign it to the CPU.

* life cycle of process :-



* There may be two CPU burst but there may not be two or more i/o burst.

Dispatcher :-

(i) Save the context
(ii) Reload the context
(iii) Transfer the control to chosen process.

Types of Scheduling :-

↳ Preemptive Scheduling
↳ Non-preemptive Scheduling

Processes under ready queue - P_2, P_3,
Process under running - P_1.

When short-term scheduler choose process :-

(i) executing process completes
(ii) executing process does i/o
(iii) If a process comes to ready queue who have higher priority than executing process.
(i) block to ready.

PAGE NO.
DATE :

(ii) suspend to ready

(iii) new to ready

(iv) interrupt

* Two i/o request can be done.

5th July 09

PAGE NO.
DATE :

Characteristics of Scheduling Algorithm :-

1. CPU Utilization :-

$$\frac{\text{Total busy time of CPU}}{\text{total time}} \times 100 \Rightarrow \quad \%$$

2. Throughput :-

$$\frac{\text{Total no. of completed process}}{\text{total time}}$$

⇒ no. of completed process per unit of time.

3. Turn Around Time :-

process complition time — process submission time

4. Waiting Time :-

Total time spent by a process in readyqueue

5. Response Time :-

first response — process submission time

Scheduling Algorithms :- There are some scheduling algorithm -

(i) First Come First Serve
(ii) Shortest job first
(iii)
(iv)

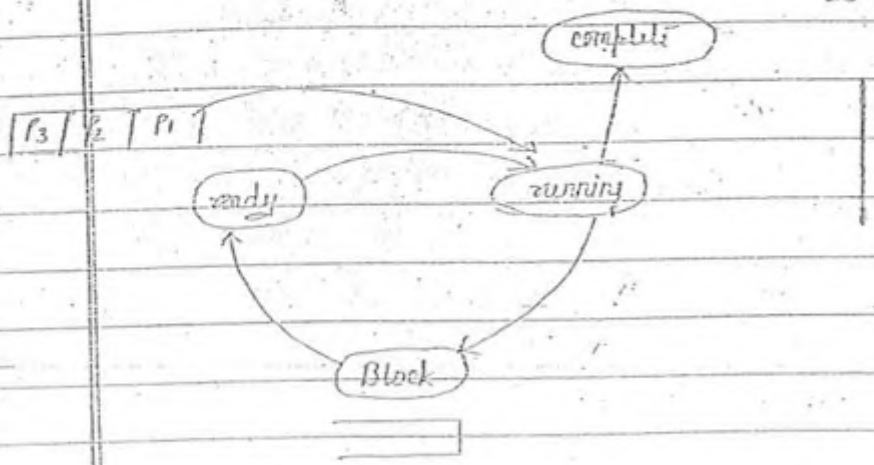
First Come First Serve Scheduling :-

* Process who comes first is served first
* It is always non preemptive.

PAGE NO.
DATE :

Eg :-

Process	Arrival Time	CPU Burst
P_1	0	5
P_2	1	1
P_3	2	2



Gantt Chart :- t=5

1. Turn Around Time :-

Turn Around time for $P_1 = 5 - 0 = 5$

Turn Around time for $P_2 = 6 - 1 = 5$

Turn Around time for $P_3 = 8 - 2 = 6$

Avg. turn around time $= \frac{5+5+6}{3} = 16/3$

2. Waiting Time :-

Waiting time for $P_1 = 0$

Waiting time for $P_2 = 4$

waiting time for $P_3 = 4$

Avg. waiting time $= \frac{0+4+4}{3} = 8/3$

2. Shortest Job First Scheduling :- two type of SJF

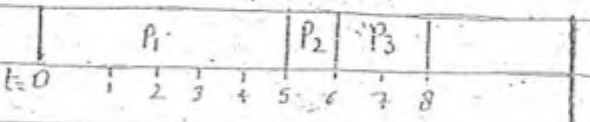
1. non preemptive version
2. preemptive version (Shortest remaining time first)

PAGE NO:
DATE :

Eg :-

Process	Arrival time	Burst time
P_1	0	5
P_2	1	1
P_3	2	2

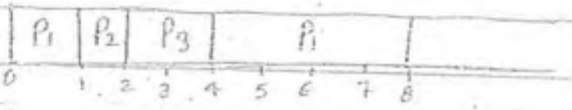
1. Non preemptive :-



★ SJF is practically implemented on long term scheduling.

★ Long term scheduler schedules the process on the basis of size of process.

2. Preemptive :- (Shortest Remaining Time First) :-



1. Avg. turn around time :-

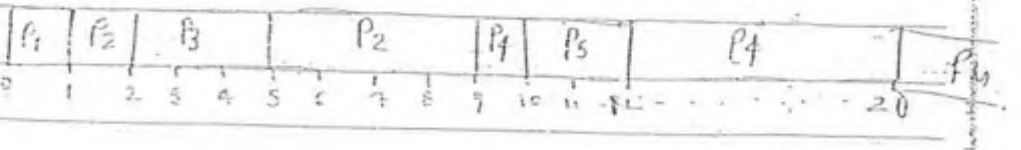
$$\frac{8+1+2}{3} = 11/3$$

2. Avg waiting time :-

$$\frac{3+0+0}{3} = 1$$

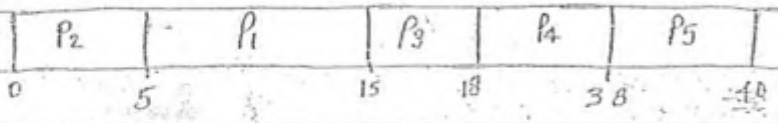
Eg :-

Process	Arrival Time	CPU Burst
P_1	0	10 → 8
P_2	1	5 → 4
P_3	2	3 → 2
P_4	5	20
P_5	10	2



PAGE NO.
DATE :

Ans:- Use FCFS to find average turn around and avg waiting time (He should be busk with SJF)



Average turn around time =

$$\frac{15+5+18+33+30}{5}$$

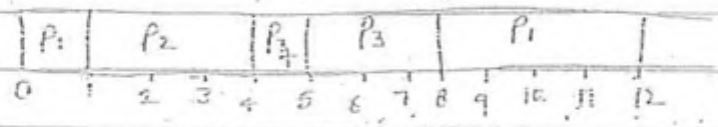
Average waiting time =

$$\frac{5+0+13+13+28}{5}$$

June/July 2004 (CS)

Process	Arrival time	burst time
P_1	0	5
P_2	1	3
P_3	2	3
P_4	4	1

Avg turn around time with preemptive shortest remaining processing time first algo.



Avg turn around time :-

$$\frac{12+3+6+1}{4} = \frac{22}{4}$$

PAGE NO.
DATE :

Priority Scheduling Algorithm :- There are two type of priority schedu-
ling :-

1. non preemptive Scheduling
2. preemptive Scheduling

* priority is assigned by OS while creating a process, it is an non negative integer value. Eg :-

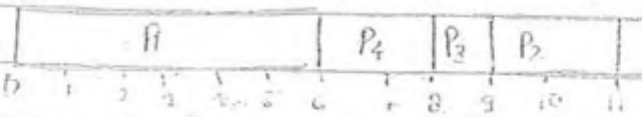
P_1	P_2	P_3	P_4
1	2	3	10

* Higher priority process can preempt the lower priority process.

Eg :-

Process	Arrival time	CPU Burst time	Priority
P_1	0	6	5 (lowest)
P_2	1	2	4
P_3	3	1	3
P_4	3	2	1 (highest)

1. non preemptive :-



Average turn around time :-

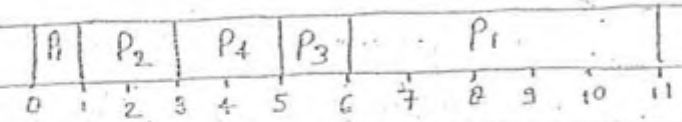
$$\frac{6+10+6+5}{4} = \frac{27}{4}$$

Average waiting time :-

$$\frac{0+8+5+5}{4} = 4$$

PAGE NO.
DATE :

2. Preemptive :—



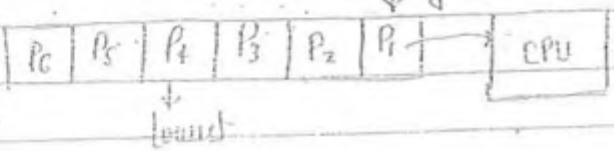
Average Waiting time :—

$$\frac{5+0+2+0}{4} = 7/4$$

Average turn around time :—

$$\frac{11+2+3+2}{4} = 18/4$$

★ Priority algorithm has a problem of starvation.



Average waiting time = $\frac{5+0+2+0}{4} = 7/4$

Average turn around time :—

$$\frac{11+2+3+2}{4} = 18/4$$

★ Priority algorithm has a problem of starvation. If higher priority process are arriving in the system then lower priority process will not get chance.

★ Starvation can be solved using aging technique.

★ Aging is a technique in which OS periodically increase the priority of process.

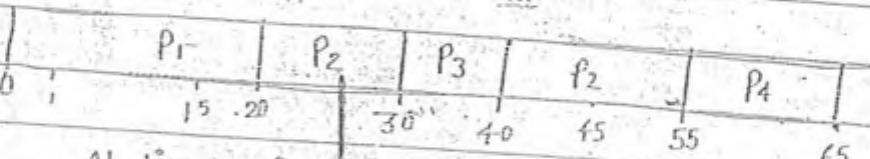
Que Calculate (AT) SRTF

Process	Execution time	Arrival time
P_1	20	0
P_2	25	15

PAGE NO.
DATE :

Process	Execution time	Arrival time
P_3	10	30
P_4	15	45

Waiting time for P_2 ?



At time 15- $P_1 \rightarrow 5$

$P_2 \rightarrow 25$

At time 20- $P_1 \rightarrow 0$

$P_2 \rightarrow 25$

At time 30- $P_2 \rightarrow 15$

$P_3 \rightarrow 10$

At time 40- $P_3 \rightarrow 0$

$P_2 \rightarrow 15$

At time 45- $P_2 \rightarrow 10$

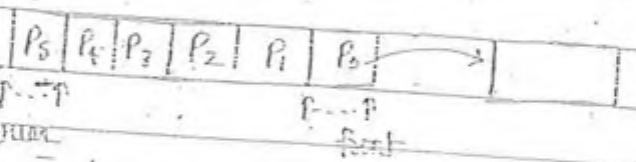
$P_4 \rightarrow 15$

Waiting time for P_2 =

$5 + 0 = (15)$ Ans

4. Round Robin Scheduling :—

* There is no non-preemptive version of round robin.



* Every process for fixed time quanta (t)
* OS assigns a fixed time quanta (q) to each process for execution.

PAGE NO.
DATE :

* If a process completes within given time quanta then immediately next process is scheduled by short term scheduler for execution.
* If process needs more time than given time quanta then after executing given time quanta, then it will again go back to the ready queue.
* If time quanta is high Round Robin becomes FCFS.
* If quanta is small more complex switching.

Eg :— RR algo; time quanta — 2.

Process	CPU burst	I/O burst	CPU burst	Arrival time
P_1	3	1	1	0
P_2	1	2	2	1
P_3	5	3	2	2
P_4	7	1	3	3

P_1	P_2	P_1	P_3	P_4	P_1	P_2	P_3	P_4	P_3	P_4	P_3	P_4
0	2	3	4	6	8	9	11	13	15	16	19	21	22

↳ goes at 4 and will come at 4+1 = 5.
↳ It will come again in ready queue at 3+2=5

* Till time 2 two processes are waiting at readyqueue P_1 and P_3. This tie can be break by SJF.

P_4	P_3	P_4	P_3	P_2	P_1	P_4	P_3	P_1	P_2	P_1

Average waiting time = $\frac{5+7+12+11}{4} = 35/4$

Average turn around time = $\frac{9+10+19+21}{4} = 59/4$

PAGE NO.
DATE :

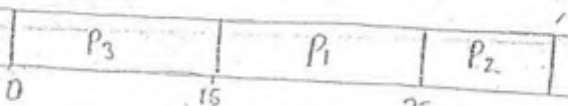
Que 2007 (IT) What is turn around time for P_2:

1- preemptive

2- non preemptive

Process	Priority	CPU time	Arrival time
P_1	10 (highest)	20	00:00:05
P_2	9	10	00:00:03
P_3	8 (lowest)	15	00:00:00

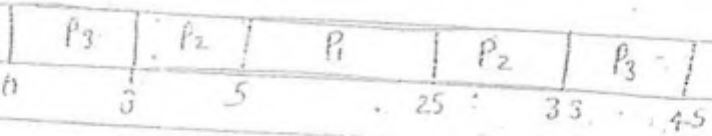
non preemptive :-



Turn around time for P_2 =

$44 - 03 = 41$

preemptive :-



At 3 - $P_2 - 9$
$P_3 - 8$

At 5 - $P_2 - 9$
$P_3 - 8$
$P_1 - 10$

At 25 - $P_2 - 3$
$P_3 - 8$

Turn around time for P_2 =

$33 - 03 = 30$

PAGE NO.
DATE :

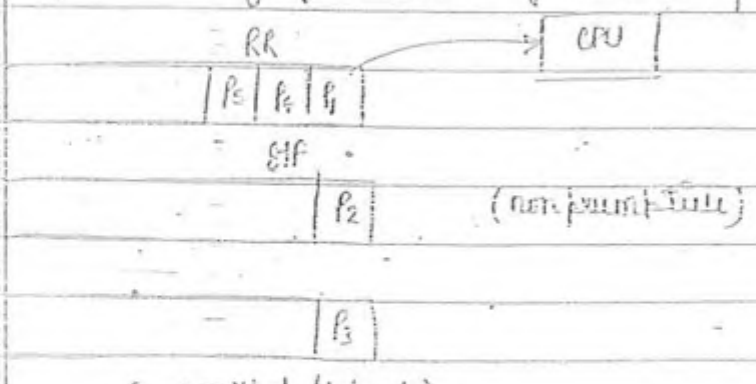
5. Multilevel Queue Scheduling.

6. Multilevel feedback queue Scheduling

* There are multiple queues
* Each has its own scheduling algorithm
* Every queue has a priority.

	Priority	
Queue 1 (RR)		
	1 (highest)	60%
Queue 2 (SJF)		
	2	30%
Queue 3 (FCFS)		
	3 (lowest)	10%

* If a process want to execute OS insert the process in any of the available queue based on nature of process.
* Nature of process decided by -
  (i) size of process
  (ii) Priority of process
  (iii) Type of process (user or kernel)
  (iv) Queue of process.
* Priority :- on the basis of priority OS decide which process will go for execution from which queue.



PAGE NO.
DATE :

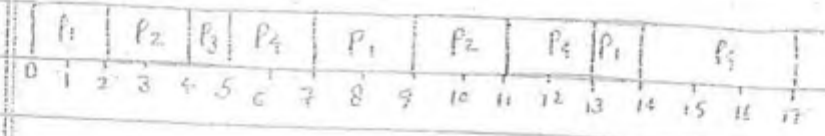
* When a queue is completed then process of queue are scheduling

Multilevel Queue	Multilevel feedback Queue
Process cannot move from one queue to another	Process can move between the queues.

Eg:-

Process	Arrival time	CPU burst	priority	time quanta = 2ms
P_1	0	5	1 (highest)	
P_2	1	4	2	
P_3	2	1	3	
P_4	3	7	4 (lowest)	

Initially all processes arrive in first (RR) queue, after they get the (q=2 ms) move to the 2nd queue (SJF). If any process in SJF has more than 3ms wait time it is moved to RR Queue



P1 | P4 | P2 | P1 | P4 | P5 | P2 | P1 — (priority)
(5) (3) (17) (11) (4) (7)

P4 | P1 | P4 | P2 | P1 — SJF (non preemptive)

(Jan-Feb 2023)

A uniprocessor system has two processes, P1 & P2. Both processes have arrived at 10 ms CPU burst and 90 ms I/O burst. Both the processes are arrived at same time. I/O burst of both process can be performed parallely. (quanta = 5)

- (1) FCFS
- (2) SJF

PAGE NO.
DATE :

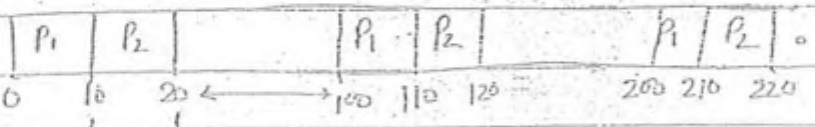
③ Static priority

④ Round Robin

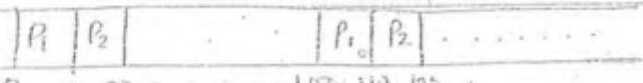
In which case CPU utilization is least

Process	Arrival time	CPU	I/O	CPU	I/O	...
P_1	0	10 ms	90 ms	10	90	
P_2	0	10 ms	90 ms	10	90	

1. FCFS :-



2. SJF :-



3. Static Priority, (P_1 has higher priority) :-

4. Round Robin :-

Ans. Round Robin.

11th July 09

PAGE NO.
DATE :

Process Synchronization :-

① global Shared Variable :- This variable is accessible by multiple process. $P_1, P_2, \ldots P_n$.

② Shared Resources :- Eg. printer, memory, CPU, file, i/p device.

③ Shared Code :- Code accessible by multiple processes.
Eg. kernel code.

(Memory)
global ch | a b |
P_1 char global ch ; P_2
(global shared)

Memory
a
localch

```
main()
{
char localch1;
localch1 = getch();
global ch = localch1
printf("%c", globalch);
}
```

Switching →

```
main()
{
char localch2;
localch = getch();
→ global.ch = localch2;
printf("%c", globalch);
}
```

Memory
b | localch

critical section.

* Process can switch at any point.

o/p ⓑ wrong

Concurrent execution of process may lead to wrong o/p.

* Critical section is a part of code, where we update the value of global shared variable. The section of the code which are not critical are called remaining.

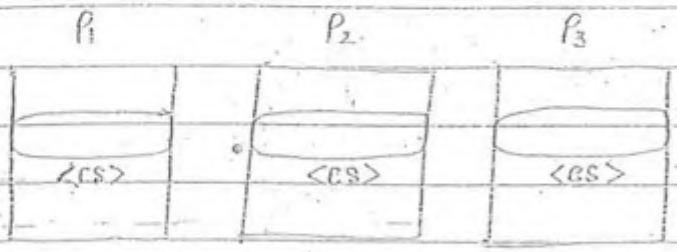
* CS is part of code where code access (updating) the global shared variable or shared resources.

PAGE NO.
DATE :

* If one process is inside its own critical section then no other processes are allowed to go inside its own CS.

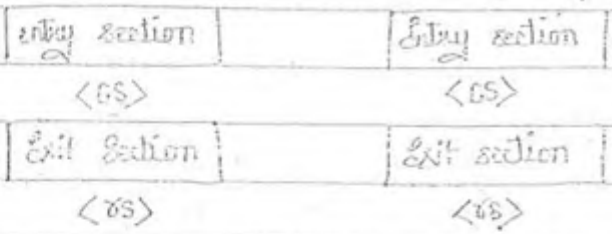
or

At a time only one processes are allowed to go inside the critical section.



⇒ code who checks that whether any other process is in critical section or not.

Design the solution for critical section problem :-



Sol^n of Critical Section Problem :- divided into two parts :-

1. Sw Approach :- (A) Sol^n of two processes
   - (i) Dekker's Algo
   - (ii) Peterson Algo
   - (B) n no. of processes
   - (i) Bakery's Algo
2. Hw Approach :-
   - (i) Test & Set
   - (ii) Exchange & swap

PAGE NO.
DATE :

3. Operating System and Compiler Suppot Soln :-

(i) Semaphores :

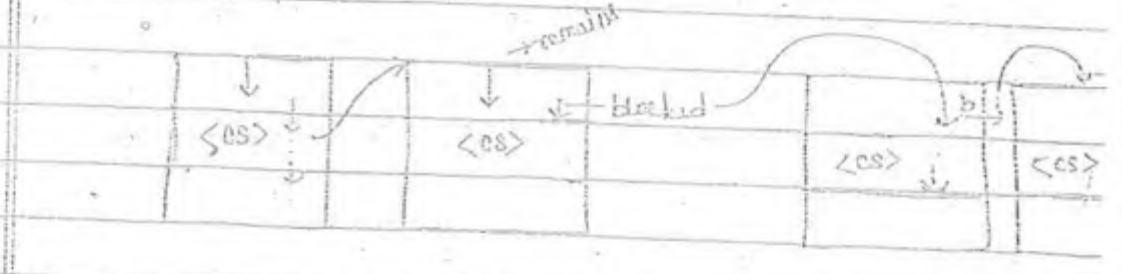
(ii) Monitor.

Characteristics of Critical Section Solution :-

(i) Mutual Exclusion
(ii) Progress
(iii) Bounded Waiting
(iv) Starvation } Soln should be free from this too
(v) Deadlock

1. Mutual Exclusion

2. Progress :- P_1 P_2 ..... P_n



which process go inside critical section if more than one process want to go there own critical section.

This decision is based on processes want to go inside critical section and this decision can be delayed to infinit time.

2. Bounded Waiting :- There should be an bound on max no. of process allowed to go inside the critical section after a process requested for CS.

* Deadlock means all processes are blocked but in starvation only one process is not getting resources.

* If bounded waiting is not then after some processes the stand process will get chance for execution.

PAGE NO.
DATE :

Solutions :-

1. Dekker's Algo 1st :-

P1 int turn = 0; P2

(shared global variable)

P1	P2
do	do
{	{
Entry section	
while (turn == 1) ②	while (turn == 0)
{	{
do nothing	do nothing
}	}
spin lock / busy waiting	
<CS>	<CS>
turn = 1	turn = 0 ①
<RS>	<RS>
} while(1)	} while(1)

* If process P1 want to exect twice and P2 dont want to exect then this is not supported by this algo.

2. Dekker's Second Algo :-

int turn[2];

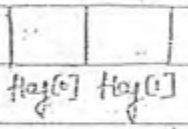
flag[0] flag[1]

P1	P2
do {	do {
flag[0] = 1	flag[1] = 1
while (flag[1])	while (flag[0])
{	{
do nothing	do nothing
}	}
<CS>	<CS>
flag[1] = 0	flag[1] = 0
<RS>	<RS>
while(1)	} while(1)

PAGE NO.
DATE :

★ This solution solve the problem of deadlock.

3. Dekker's third Algorithm :-

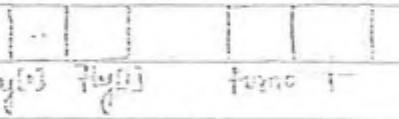


P_1	P_2
do	do
{	{
while(flag[1])	while(flag[0])
{	{
do nothing	do nothing
}	}
flag[0] = 1	flag[1] = 1;
<CS>	<CS>
flag[0] = 0	flag[1] = 0
<RS>	<RS>
} while(1)	} while(1)

So both processes are in critical section, This algo violate the property of mutual exclusion. So this also not a right algo.

Peterson's Algo :-

```
int turn = 0;
int flag[2];
```



P_1	P_2
do{	do{
flag[0]=1;	flag[1]=1
turn = 1;	turn = 0;

PAGE NO.
DATE :

```
while (flag[1] && turn==1)        while (flag[0] && turn==0)
{                                 {
do-nothing                        do nothing
}                                 }

<CS>                              <CS>

flag[0]=0;                        flag[1]=0;

<RS>                              <RS>

} while (1);                      } while(1);
```

* This scenario is absolutely correct, it don't have deadlock also. It also has bounded waiting.

2. Hardware Support :-

(i) test & set

(ii) Exchange & swap

1. Test & Set :- Test & set, & swap are special instructions supported by hardware.

```
int TestAndSet (int & lock)
{
  int temp;
  temp = lock
  lock = 1;
  return temp;
}
```

Atomic Instruction

* An instruction cycle means when logic is implemented in an instruction cycle then means the full code execute in a single instruction cycle.

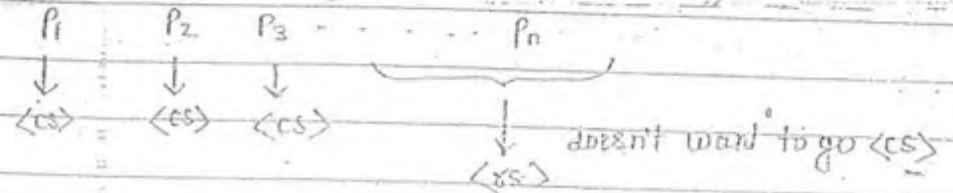
```
P1                P2                P3          Pn
do                do
{                 {
while (TestAndSet (turn));   while (TestAndSet (turn));
{                 {
do nothing        do nothing
}                 }
```

PAGE NO.
DATE :

```
<CS>                    <CS>
turn = 0                turn = 0
} while(1)              } while(1)
```

In pass by reference another object is not created, So the memory location with name turn is used by int & bolt. int & bolt is alliance of turn because both are name of same memory loc.

* no bounded waiting : Starvation is possible.
* Progress



If decision for going <CS> is based on only the processes who want to go in critical section, then this is progress.

2. Exchange & Swap :-

```
Atomic {
    void swap (int &a, int &b)
    {
        int temp;
        temp = a;
        a = b;
        b = temp;
    }
}
```

bolt

temp

* This solution provide the mutual exclusion.

Solution is given below :-

PAGE NO.
DATE :

P_1	P_2
int k=1	int k=1
while (k==1)	while (k==1)
{	{
swap (lock, k)	swap (lock, k)
}	}
<CS>	<CS>
lock = 0	lock = 0
<RS>	<RS>

12th July 09

PAGE NO.
DATE :

Semaphore :- A global shared variable is used for process synchronization. There are two type of semaphore variable implementation.

(i) Int type
(ii) Structure type
  - (a) Counting Semaphore
  - (b) Binary Semaphore

* OS gives two function to access the value of semaphore variable.

(i) wait (Semaphore S) - P(S)
(ii) Signal (Semaphore S) - V(S)

P(S) / V(S) > Semaphore Variable

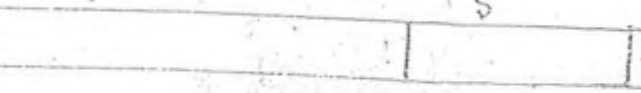
Code for wait():-

```
int s=1;
wait (int s)
{
  while (s<=0)
  {
    do nothing
  }
  s--;
}
```

Code for signal():-

PAGE NO.
DATE :

```
signal(int s)
{
  stt;
}
```



P_1	P_2	P_3
do {	do {	do {
wait(s)	wait(s)	wait(s)
↓	↓	↓
<CS>	<CS>	<CS>
signal(s)	signal(s)	signal(s)
<RS>	<RS>	<RS>
} while(1);	} while(1);	} while(1)

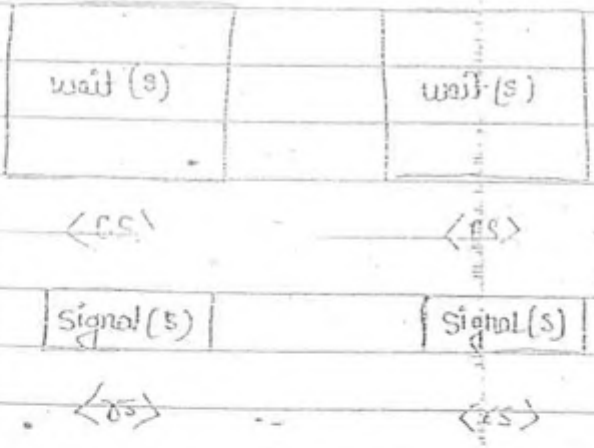
* Sol^n of busy waiting & spin lock is structure implementation

Structure Semaphore :-

```
{
int count;
process *l;
} s;
S.count = 1;
wait (Semaphore S)
{
  S.count--;
if (s.count<0)
```

PAGE NO.
DATE :

{
block the process & insert into
block queue of semaphore
}
Signal (Semaphore S)
{
S.count ++;
if (S.count ≤ 0)
{
pick one process from block queue of
semaphore and insert to the ready
queue of the sys^m
}



* Blocked process when start again then it will from critical section.
* no busy waiting.
* Fifo is progress
* A field is used to link the process.

PAGE NO.
DATE :

* If scheduling is FIFO in block queue then bonded waiting is there and if we use any other policy then there is no bounded waiting.

Implementation of Critical Section :-

P_1	P_2	P_1	P_2
P(Sx)	P(Sx)	P(Sx)	P(Sy)
P(Sy)	P(Sy)	P(Sy)	P(Sx)
<cs>	<cs>	<cs>	<cs>
V(Sx)	V(Sx)	V(Sx)	V(Sy)
V(Sy)	V(Sy)	V(Sy)	V(Sx)

will go to deadlock state

P(Sx) → wait

P(Sy) → Signal.

PAGE NO.
DATE :

Que 80
gate 2003, CS.

binary semaphore S, T

P	Q
while (1)	while (1)
{	{
W —	U —
—	—
—	—
print 0	print 1
print 0	print 1
X —	Z —
—	{
}	

O/p should be 00 11 00 11 00 11 00

Then which of following is true :—

(a) P(S) P(T)
V(S) V(T)
S=1, T=1 ⇒ wrong

(b) P(S) P(T)
V(T) V(S)
T=0, S=1 ⇒ right

PAGE NO.
DATE :

Binary Semaphore :—

```
Struct BinarySemaphore
{
  Boolean value;
  process *L;
} bs;
bs.value = true(1);
wait(BinarySemaphore bs)
{
  if (bs.value == true)
  {
    bs.value = false
  }
  else
  {
    block the process
    and insert into
    block queue of
    Semaphore
  }

Signal(BinarySemaphore bs)
{
  if (queue is empty)
  {
    bs.value = true;
  }
  else
  {
    give one process from
```

PAGE NO.
DATE :

block queue of
osemaphore and
insert into ready-
queue of the sys^m.

bs.value = true / false

{

P_1	P_2	P_3
do	do	
{	{	
wait(bs)	while(bs)	
<cs>	<cs>	
Signal(bs)	Signal(bs)	
<rs>	<rs>	
} while(1)	} while(1)	

☆ Busy semaphore has its own blocking queue.

☆ This soln provide mutual exclusion but dont deal with bounding limit.

PAGE NO.
DATE :

Qn 1:- Implementation of counting semaphore using binary semaphore.
[Two binary semaphore and one integer type variable.]

Qn 2:- Suppose there are 10 no. of processes and they are executing concurrently. Code for P_1 and P_9 are similar

wait (S)
—
—
—
Signal (S)

Code for P_{10} —
Signal (S)
—
—
—
—
wait (S)

Where S is semaphore variable & its initial value = 7
then find out how many processes are allowed to go inside the critical section simultaneously.

Ans (4)

PAGE NO.
DATE :

Bakery's Algorithm :-

```
Boolean choose[n] - Initial value = false
int number[n] - Initial value = false

P0 - (1)

choose[0] = true;
number[0] = 1 + max(number[0], number[1], ..., number[n-1]);
choose[0] = false;
for (j=0; j<N; j++)
{
  while (choose[j]);
  while (number[j] != 0 && number[j] < number[0];
}

P1 - (2)

choose[1] = true;
number[1] = 1 + max [n[0], n[1], ... n[n-1]);
choose[1] = false;
for (j=0; j<N; j++)
{
  while (choose[j]);
  while (number[j] != 0 &&
    number[j] < number[1]
}
```

PAGE NO.
DATE :

P_0

F	F	F	F	F

0	0	0	0	0

P_1	P_2
↓	↓
.	.
.	.
<CS>	<CS>
number[0] = 0	number[1] = 0

★ while (——);

means :—

```
while (...)
{
  do nothing
}
```

★ In this algorithm there is bounded waiting. It also support mutual exclusion.

18th July 09

PAGE NO.
DATE :

Que :-

```
boolean waiting[i]; boolean lock ;   => global variable
do
{
  waiting[i] = true ;
  key = true ;                       => local variable
  while (waiting[i] && key)
  {
    key = TestAndSet(lock);
  }
  waiting[i] = false ;
  <CS>
  j = (j+1) % n;
  while ((j != i) && waiting[j])
  {
    j = (j+1) % n;
  }
  if (j == i)
  {
    lock = = false ;
  }
  else
  waiting[i] = false;
  (initial value = false;
```

You have to find this soln provides :-

(i) mutual exclusion ?

(ii) Progress ?

(iii) Bounded Waiting.

Ans :-

PAGE NO.
DATE :

Classical Problems of Process Synchronization :-

1. Producer and consumer problem (Soln using semaphore)
2. Reader writer problem (soln using semaphore)
3. Dining philosopher problem (soln using semaphore)

Producer and consumer problem :-

1. def :- ① Two processes producer and consumer
   The processes exactly concurrently
   ② Producer produces item and append in buffer of size N.
   ③ Consumer takes the items from buffer and consumes it.
   ④ Buffer is shared global variable between producer and consumer, and its size = N.
2. process Synchronization are :-
   ① Mutual Exclusion (Producer & consumer can't access buffer simultaneously)
   ② Producer cant append item in full buffer.
   ③ Consumer can't take item from empty buffer.
3. Soln using semaphore :-

```
Semaphore mutex = 1    // mutual exclusion
Semaphore full = 0     // no of full cells
Semaphore empty = N    // no of empty cells
```

Producer Code	Consumer Code
do {	do { wait(full); // if full = 0 consumer block
produce item;	wait(mutex);
wait(empty); // N--	take the item from buffer <CS> ⑪
wait(mutex);	signal(mutex);
append item <CS> ①	signal(empty);
signal(mutex);	
signal(full);	
} while(1);	} while(1);

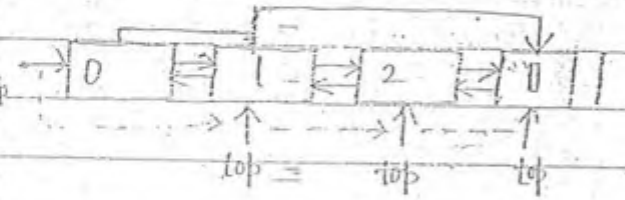
PAGE NO.
DATE :

| 0 | 1 | 2 | 0 | 3 | 2 | 0 | 3 | 4 | | | |

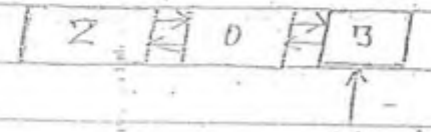
ault
How

⇒ time to use is more then it is reused recently use LRU which page has less time to use is removed.

using link list :—



page 1 is removed.



⇒ LRU is time consuming because for each value it maintain data structure so Approx LRU algo is used in sys^m. LRU needs hw support.

4. APPROXIMATE LRU :

① Reference bit Algo ④ Enhance Second Chance Algo.
② Additional Reference bit Algo
③ Second Chance Algo

1. Reference bit Algo :—

page table	V/I	Reference bit
page 0	I	0
page 1	V	1
page 2	V	1
page 3	V	1
page 4	I	0

reference string - 1, 1, 2, 3, 4

1	2	3			

nder for

⇒ If the reference bit = 1 then sys^m will not remove page but if 0 then it can remove & V/I bit should be 0 for deletion.

PAGE NO.
DATE :

⇒ Given table CPU can't decide that which page is removed but if there were some page which are in memory in starting then this algo will work

Suppose 0, pages are available before -

page		V/I	0
1		I/V	0/1
2		I/V	0/1
3		I/V	0/1
4		I	0

0 1 2 3 1 2

⇒ This algo fails when all bits are 1, so 2nd algo is used.

2. Additional Reference Bit Algo :-

Sys^m takes 5 (for 5 pages) register and size = 4 bit (Assume)

page 0	0				
1	1				
2	1				
3		1			
4					

page	V/I	reference bit
1	I/V	0/1/0
2	I/V	0/1/0
3	I	0/0
4	I	0

reference string = 1, 1, 2, 3, 1, 4

⇒ After some time copy the reference bits to register and reset all reference bit. Initially all reg are 0 after copying reset all reference bit

1	2	3	1	4	3

⇒ Before copying value to register it will shift all the value of reg. in 1-bit right

PAGE NO.
DATE :

Explanation :-

Step 1:-

page 1	V	1
page 2	V	1

Copy reference bit into register msb

page 0	0
page 1	1
page 2	1
page 3	0

page 3	V	1

page 4 required page replacement. So it first shift register value and copy next reference bit

					Convert to decimal
page 0	0	0	0	0	0
1	1	1	0	0	⇒ 12
2	0	1	0	0	⇒ 4
3	1	0	0	0	⇒ 8

4 is less so it is least recently used so it is replaced so page no. 2 will be replaced.

page 4	V	1

⇒ If all msb's will be high the and remaining bits values are same then we can't decide which value page is replaced, Eg :-

page 0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0

⇒ In this case we will use FIFO Technique.

PAGE NO.
DATE

3. Second Chance Algo :—

V/I Reference bit ⇒ Circular list

page 0	V	1	⇒	page is in memory & referred bytes
page 1	V	1		
page 2	I	0	⇒ page is not in MM	
page 3	V	0	⇒ page	is in MM but not referred by CPU
page 4	I	0		
page 5	I	0		

0	1	3

When search for removing a page it change ref bit 1 to 0 here

It start searching from page 0 & change ref bit from 0 to 1 & when find V w/ 0

Suppose I/O to page 0

page 1	V	0
page 2	I	0
page 3	I	0
page 4	V	1
page 5	I	0

0	1	4

5

Its first bit page get 0 it removes that page and insert the new page

⇒ no of iteration = 1+1

now for page 5 it will start searching from page 0 and first it will find page 0 with V – 0 so page 0 will be remove

page 0	I	0
page 1	V	0
2	I	0
3	I	0
4	V	0
5	V	1

5	1	4

PAGE NO.
DATE :

4. Enhanced Second Chance Algo :—

⇒ It is based on reference bit & modified bit. so total no of combⁿ = 4

Reference bit	Modified bit	
0	0	⇒ removed first
0	1	⇒ not possible
1	0	⇒ removed at 2nd place
1	1	⇒ removed in last.

for removing it first CPU replace it and send it to secondary memory and then bring page from secondary memory to main memory

⇒ In third case only one operation bcoz page is not modified so while replacing no need to copy it into secondary memory. only new page will be brought to MM from SM.

⇒ no. of iteration will be > 1.

SOME OTHER TECHNICS :—

① Most frequently used page

② least frequently used page

⇒ In most frequently used page, the page which are most are removed first.

⇒ In least frequently used pages, the page which are used least are removed first

⇒ It can't be decide that which is better in this ; it depend on reference string.

PAGE NO.
DATE :

FRAME ALLOCATION :— How many frames will be allocated to process

MM

no. of frames = f

$P_1, P_2, P_3, \cdots P_n$

size of process

$S_1, S_2, S_3 \cdots S_n$

$$\sum_{i=1}^{n} S_i = S_1 + S_2 + S_3 + \cdots + S_n$$

no. of frames allocated to process P_1 =

$$\frac{S_i}{\sum_{i=1}^{n} S_i} \times f = \frac{\text{size of process}}{\sum_{i=1}^{n} S_i} \times \text{no. of frames}$$

⇒ If size of process is less than less no of frames are allocated and if size of process is large then more no. of frames are allocated to the process

min no. of frames ?

code	ADD M_1, M_2	page 0
data	$M_1 = 5$	page 1
data	$M_2 = 10$	page 2

ADD M_1, M_2

$[M_1] \leftarrow [M_1] + [M_2]$

page 0
page 1
page 2

min no. of frames for this instruction = 3

PAGE NO.
DATE :

⇒ no. of min frames depend on the instruction upon instruction set architecture. Instructions are depend upon computer Architecture so it will also depend upon comp. Arch..

max no. of frames :-

max no. of frames = max no. of frames into memory.

Local Vs Global Allocation (Page Replacement)

Page 0	f0	P_1 {0,1,2,3}
Page 1	f1	
Page 4	f2	P_2 {4,5,6,7,8,9}
Page 5	f3	
Page 6	f4	
Page 7	f5	
Page 10	f6	P_3 {10,11,12}
Page 11	f7	

P_1 during execution generate reference page 2. Then process P_1 will replace its own page. This type of replacement is called Local Replacement. This technique is less flexible (because it has only 2 frames and after replacement no. of frames will not increase)

A process can replace pages of any other process this type of replacement is called global page replacement. Let page no 5 is replaced of P_2 by page 2 of process P_1 Here no of frames are increasing of P_2 and no. of frames decrease of P_1, so it is flexible.

PAGE NO.
DATE :

Active Page :- used by process frequently.

If active page in MM the less no. of page fault

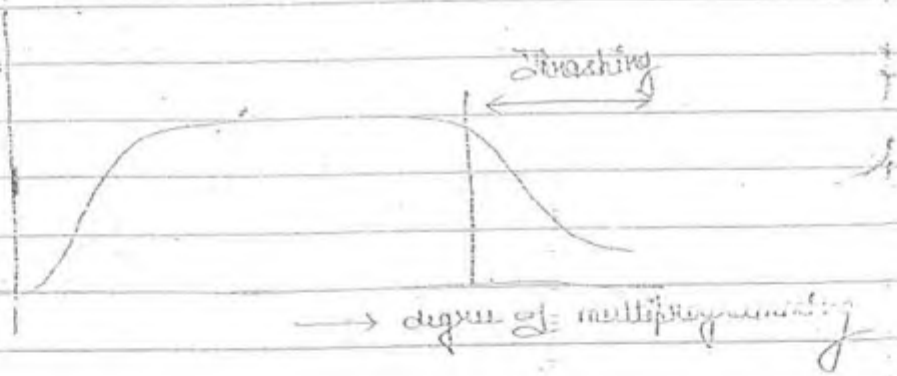
degree of multiprogramming :- no of process loaded in MM at a time.

→ degree of multiprogramming is increased to increase the CPU & resource utilization.

If page P_1 demand the active page of P_2 then when P_2 execute it need active page here occure page fault then P_2 will remove any other process Active page. This is called Thrashing.

working window of size 4

Thrashing is a case of sys when sys is in page replacement instead of execution of process.



Thrashing will always occure in Global replacement. It will not occure in Local replacement.

Global Replacement	Local Replacement
① Thrashing occure	Thrashing does not occure
② more flexible	less flexible
③ more robust	less robust

PAGE NO.
DATE :

How to Avoid Page fault :—

① Load All active pages of the each process to the MM.

WORKING PAGE MODEL :—

decides which is active page & which is passive.

$P_2 = t1$

1, 2, 1, 3, 4, 1, 5, 6, 7, 1, 3

working set ⇐ window size 4

{1, 3, 4}

↳ set of Active page.

⇒ Active page decision depend on size of working set window.

⇒ If size of working set is small than some active process it will not contain and if it is big then it will contain some passive pages. So size of working page model should be optimized.

Remove degree of multiprogramming to reduce page fault :—

Assume that there is n no. of process then —

$P_1, P_2, \cdots P_n$

WSS1, WSS2, ..., WSSn ⇒ set of active pages of each process.

$$\sum_{i=1}^{n} WSS_i \leq \text{no. of frames}$$

If it is not satisfied then Any process active page will not be in MM and page fault will occurs and and it will again lead to page fault.

If this cond^n occurs the OS will choose a process then unroll all page in SM and suspend the process. Allocate the free frames to active pages of other process means OS is decreasing degree of multiprogramming till cond^n achieved.

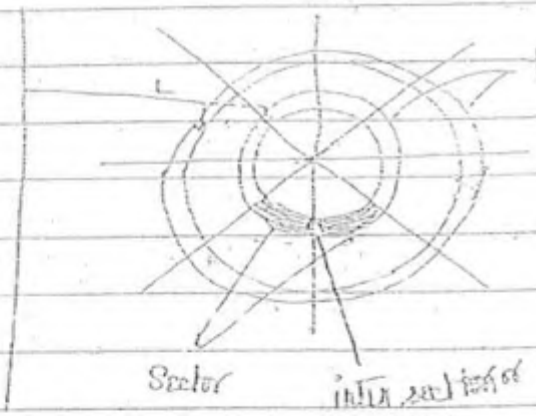
PAGE NO.
DATE :

DISK STRUCTURE AND SCHEDULING :-

⇒ Platter may be single coated or double coated

⇒ HDD contains 16 platter

⇒ track is physical entity & is divided into sector.

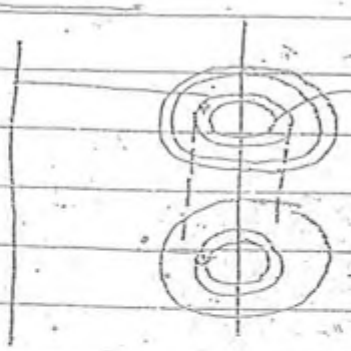


platter
track
sector
< platter no.
side
track
sector >

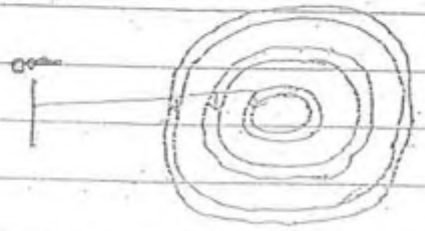
10th Aug

PAGE NO.
DATE :

DISK ACCESS TIME



no of cylinders = no of tracks.



Seek time : Any time taken by head from one track to another.

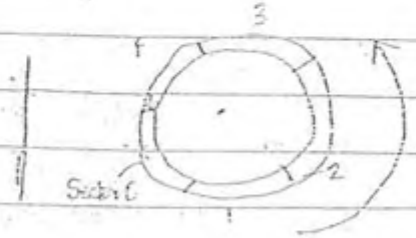
⇒ movement time from one track to other is not same so we take avg time.

track 0 to track 1 — 1 ms
track 1 to track 2 — 1 ms
track 0 to track 1 time taken by head = 10 ms
track 1 to track 2 time taken by head = 10 ms

Average seek time = 20/10 ms

time taken by head to go from track
to track 2 = 10 ms

Rotational delay or Latency :



(constant avg velocity)

PAGE NO.
DATE :

sector no_1 is needed by sys^m then it will take time to go under head.

time taken by track 1's sector1 > time taken by track 1's sector2.

⇒ Rotational Latency is not fixed. so we take avg value.

Rotational latency = one complete revolution time / 2

If Angular speed of disk = 5200 r/min (revolution time)

1 min no. of revolution = 5200

1 sec " " " = 5200/60

time taken by disk in one complete revolution = 1/5200/60

= 60/5200

Rotational Latency = 60/5200×2 sec.

Block Transfer Time :-

⇒ Block is logical entity.

→ Acc to OS HDD is devided into block and size of block is fixed.

suppose block size = Size B.

Transfer rate (tr) = capacity of track / one complete revolution time

block transfer time = size of block / transfer rate

PAGE NO.
DATE :

Block Access Time = seek time + rotational time + block transfer time.

Ques:- A HDD sys^m has following parameter

no. of track = 500

no. of sector = 100/sector

no of byte = 500/sector

Time taken by head to move from one track to adjacent track = 1 ms.

rotational speed = 600 rpm

What is the avg time taken for transferring 250 byte from the disk.

⇒ Here size of data is less that the size of sector so the time = 1 latency time.

Avg seek time = $\frac{499 \text{ ms}}{2}$

= 249.5 ms

Rotational delay =

speed = 600 rpm

600 revolution in 1 min

600 " " 60 sec

in 1 sec = $\frac{600}{60}$ 10.

1 sec track = 1/10 = .1 sec

Rotation delay = .1/2 = 0.05 sec

.05 sec × 1000ms = 50 ms

Data Transfer Rate = 100 × 500 (one track capacity)

0.1 sec

= 5×10^5 B/sec

If track 3 then distance = 2 cm

so within track = 500

then distance 100 cm

PAGE NO.
DATE :

Transfer time = $\frac{50}{250 B}$ $= 5 \times 10^5$ B/sec

5×10^5 B/sec

$= 50 \times 10^5$ B/sec

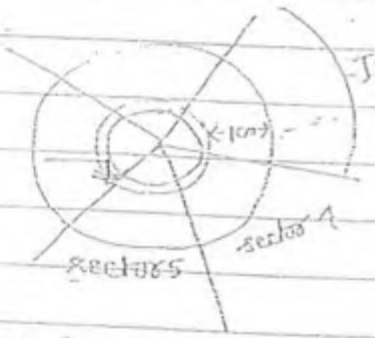
$= 50 \times 10^{-2}$ ms

$= .5$ msec

$= (249.5 + 50 + .5)$ ms

$=$ **300 ms.**

2. If disk has 20 sector/track and head is currently at the end of 5th sector of innermost track. Head can move at the speed of 10 m/sec. Disk is rotating with constant angular velocity 600 rpm. How much time will it take to read 1MB contiguous data, starting from the sector 4 of the middle most track.



seek time = $\frac{\pi \text{ cm}}{10 \text{ m/sec}}$

$= \frac{\pi}{10 \times 10^2}$ cm/sec

$= \pi \times 10^{-3}$ sec

$=$ **7 msec**

time taken by head to reach at sector 4 = $\frac{\text{one complete revolution time}}{20}$

one comp. rev time = $\frac{60}{600}$ rpm

$\frac{600}{60} =$ **10 no. of time / sec.**

PAGE NO.
DATE :

$= 1/100$ sec

$= 1000/100$ ms

$= \boxed{10 \text{ ms.}}$

Time taken by one sector to pass during head $= \frac{10 \text{ ms}}{20}$

$= \boxed{.5 \text{ ms}}$

one comp sec takes 10 ms

so 1 sector is less bcoz we have to find sector 4, then time taken = 9.0 ms. (bcoz head is also moving, and disk is rotating at same time so at the time head will go to track 1 and at same time disk will rotate and head will go to start of sector 4. so time taken by head in moving and not to count).

Transfer rate $= \frac{10 \text{ MB}}{10 \text{ ms}} = \frac{10 \text{ MB}}{10 \times 10^{-3}} = \boxed{10^3 \text{ MB/s}}$

(capacity of one comp / track sec time)

Transfer time $= \frac{\text{Data size}}{\text{Transfer rate}}$

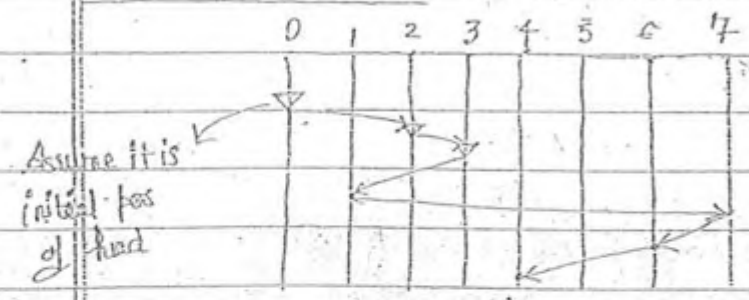
$= \frac{1 \text{ MB}}{10^3 \text{ MB/s}} = \boxed{1 \text{ msec}}$

Total time = 9 ms + 1 ms

$= \boxed{10 \text{ ms.}}$

PAGE NO.
DATE :

DISK SCHEDULING :— Only seek time will be considered.



SM buffer has no requests

2, 3, 1, 7, 6, 4

Thus all the track no. from which head has to read the data.

1. FCFS :—

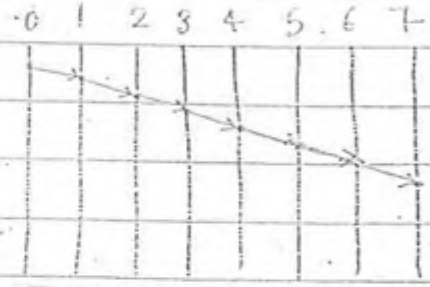
no. of head movement

$= (2-0)+(3-2)+(3-1)+(7-1)+(7-6)+(6-4)$

$= 2+1+2+6+1+2 = \boxed{14}$

⇒ simple but no. of head movement is more

2. Shortest Seek Time First :



no. of head movement $= (1-0)+(2-1)+(3-2)+(4-2)+(6-4)+(7-6) =$

$= 1+1+1+1+2+1$

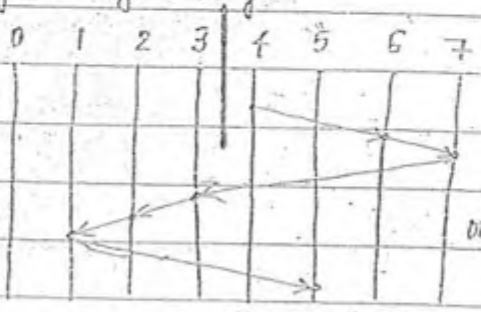
$= \boxed{7}$

⇒ It is best as compare to FCFS but it is not necessary that it will always give best result.

PAGE NO.
DATE :

3. SCAN :- Head moves from left to right and comes at last track from where it changes the direction and move from left to right. Also called as ELEVATOR.

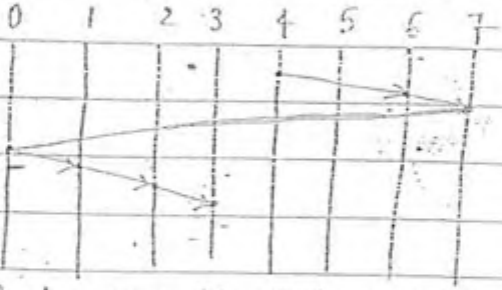
Initial pos. of head at track no. 4 and initial direction of head from left to right.



⇒ Head always chang direction either start of track or end of track

no. of head moves = (4-4) + (6-4) + (7-6) + (7-3) + (3-2) + (2-1)
= 0 + 2 + 1 + 4 + 1 + 1
= 9

4. CSCAN (Circular SCAN) :- Head can scan the req. in one direction either left to right or right to left and change the direction either start or end of the track.

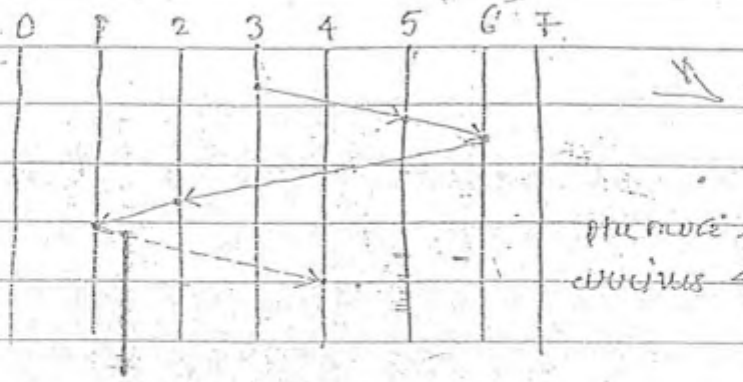


no. of head moves :- (4-4) + (6-4) + (7-6) + (1-0) + (2-1) + (3-2)
= 0 + 2 + 1 + 1 + 1 + 2
= 7

11th Aug 09

PAGE NO.
DATE :

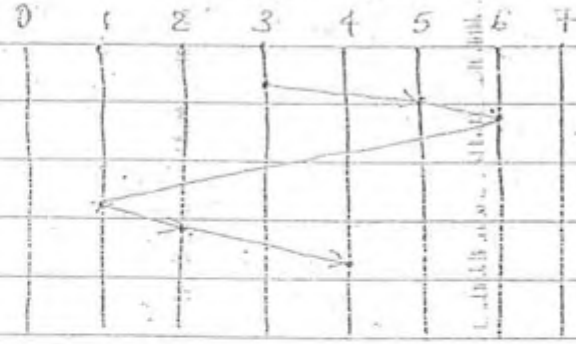
Look :- Head can change the direction in between tracks.



request are - 1, 2, 5, 6
initially head at - 3
initial direction - left to right

no. of head moves = (5-3) + (6-5) + (6-2) + (2-1) + (4-3)
= 2 + 1 + 4 + 1 + 3
= 9 + 3
= 11

C-Look :- Head can change direction i b/w track but can serve in only one direction



initially head at - 3
direction left to right

no. of head movement = (5-3) + (6-5) + (6-1) + (2-1) + (4-3)
= 2 + 1 + 5 + 1 + 2
= 11

PAGE NO.
DATE :

Look & SSTF are implemented in h/w. generally these two are supposed good and SSTF gives half head movment as compare to FCFS.

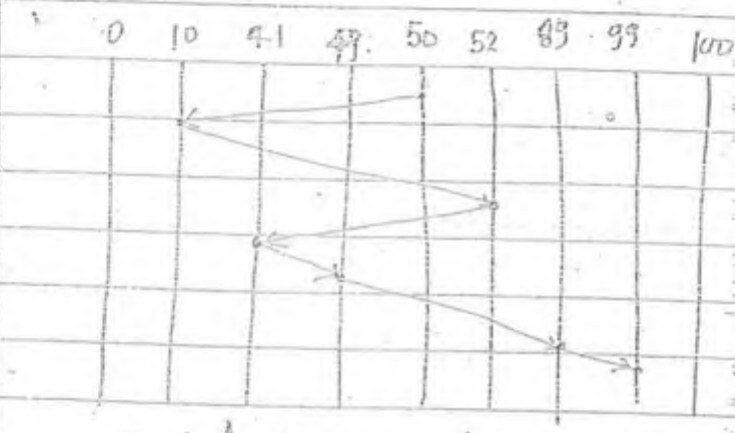
Qu:- Disk has 100 no. of track ; all track are equidistance and distance b/w two adjacent track = 1 cm. Head move horizontally with speed 1 m/sec.

Request are 10, 52, 41, 49, 89, 99

Algorithm = FCFS, SSTF & Look

Find the average seek time for head. Assume that initial pos of head at track no 50 and direction is left to right.

1. FCFS :-



No. of head moves = (50-10) + (52-10) + (52-41) + (49-41) + (89-49) + (99-89)

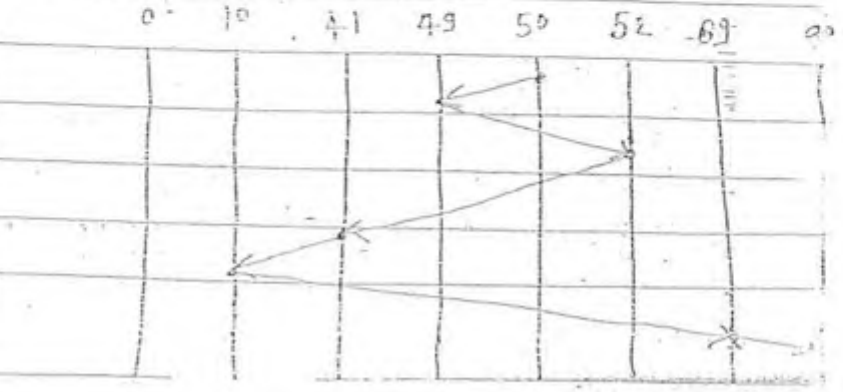
= 40 + 42 + 11 + 08 + 40 + 10

= 151

Speed = 1 m/sec = 100 cm/sec time = 151 cm / 100 cm/sec

Avg seek time = 1.5 sec / 6 = 10

2. SSTF :-

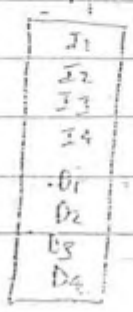


PAGE NO.
DATE :

Qu 45 :- Consider a system with 2-level paging scheme in which unpaged memory access takes 150 nsec. Servicing a page fault is 8 msec. An average instruction takes 100 nsec of CPU time and 2 memory access. TLB hit ratio is 90% and page fault rate is one in every 10,000 instructions. What is the effective average instruction execution time.

1 instruction takes = 100 ns + 2 (memory access time)

total no. of memory



$t_t = 0$

TLB

Access time of memory

$t_m = 100$ ns

page table

90%

hit ratio = 1/20,000

1 page fault in 2000 instruction

2 memory reference so page fault = 20000

⇒ Address divided into two part page & offset. Page is searched into TLB, and then memory is searched.

Average effective time w/o page fault =

$= .9 \times (t_t + t_m) + (1 - .9) \times (t_t + t_m + t_m)$

$= .9 \times (0 + 150) + (.1) \times (0 + 150 + 150)$

$= 90 + 30 = 135 + 30$

$= 165$

Effective Access time with page fault =

(1 − page fault rate) (memory access time) + page fault rate × page service time

page fault rate = $.5 \times 10^{-4}$

$= \left(1 - .5 \times 10^{-4}\right)(165) + .5 \times 10^{-4} \times 8$ msec.

$= \left(1 - .5 \times 10^{-4}\right)(165) + .5 \times 10^{-4} \times 8 \times 10^6$ ns

$= 165 + 4 \times 10^2$ ns

$= 565$ ns.

Actual effective time = $100 + 2 \times (565)$

$= 100 + 1130$ ns

$= 1230$ ns.

PAGE NO.
DATE :

Average = $p_1 \times (t_1 + t_m) + (1-p_1)(t_t + t_m + t_m)$

1 − pagefault × Average
+ pagefault × page service time

(2004, CS)

Qu 66 :-

Locality of reference

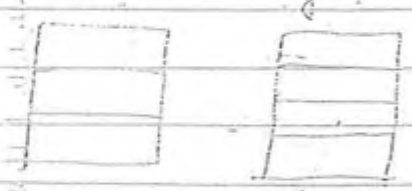
temporal locality of reference (related to time) — Spatial locality of reference (related to space)

Ans If instruction i is referenced by CPU in time $t = t_1$ then this will again reference in near future is known temporal locality of reference.

Ans If instruction i is referenced by CPU then its near by instructions will reference by CPU in near future. In Spatial Locality of reference.

→ Belady's Anamoly occurs beez of stack class algorithm.

Stack Class Algorithm :- System 1 has n no. of frame.
System 2 has (n+1) no. of frames.



frame = 3 frame = 4

reference string = 2, 1, 1, 3, 4, 1, 2, 0, 1

PAGE NO.
DATE :

- At any time if an algo satisfies this condⁿ

no. of page in sysᵐ 1 $\subseteq$ no. of pages in sysᵐ 2
(n frames) (n+1) frames

Then it comes under stack class algorithm.

⇒ FCFS doesn't satisfy this condⁿ. so Belady's Anamoly occurs.

Optimal Size of Page :

Suppose avg page size = s'
page size = p
and size of each entry in page table = e
total overhead = space required to store page table + internal fragmentation

no. of page required = s/p
no. of entries in page table = s/p
size of one entry = e
total space = s/p × e

Total overhead = $\frac{s}{p} \times e + \frac{p}{2}$ (suppose internal frag. is half page)

diff with respect to p (here p is variable)

$= -\frac{se}{p^2} + \frac{1}{2}$ (

$-\frac{se}{p^2} + \frac{1}{2} = 0$

$\frac{s}{p^2} = \frac{1}{2}$ $p^2 = 2se \Rightarrow \boxed{p = \sqrt{2se}}$

PAGE NO.
DATE :

FILE SYSTEM

File Sys^m is a ds maintain by os to control the files. when secondary memory is formatted by os.

HDD

- low level formatting (Track & sectors are created)
- high level formatting (file sys^m is created & SM is divided into logical entity called blocks.)

⇒ Every block has no. and size of block is decided by os during formatting. (Some times user may decide size of blocks)

⇒ If size of SM = 2^{64} B

size of block = 512 B

= 2^9 B

no. of block in SM = $\frac{\text{size of SM}}{\text{size of block}}$

= $\frac{2^{64}\ B}{2^9\ B}$

= 2^{55} B

So every block no. represented by 55 bits.

File :- An abstraction provided by os so that user can store their data w/o knowing the internal structure of SM.

File 1 — 8 KB block size = 4 KB

PAGE NO.
DATE :

Allocation algm = 11
time required = 8
19

Que :- Free disk space can be kept track of using a free list or bit map. Disk addresses required d bit. For a disk with b blocks, f of which are free, state the condn under which the free list uses less space than bitmap.

bit map
Size = B bits
Free list
free blocks = f every free block keep pointer of next free block.
Address of one block = d bits
Size of data structure = f×d bits
f×d bits < B bits.

Gate 1999 :-

Que ; Sysm calls are usually invoked by :-
(a) Sw interrupt [✓] (b) Polling
(c) Inclined jump (d) Privilidge Instruction [ ]

Polling - checking status of register continuously.

Que :- Advantage of virtual memory
(a) faster access of memory on an average [X]
(b) Process can be given protected [X]
(c) Linker can assign add independent of Program where it is loaded [ ]
(d) Program larger than size of physical memory can run [✓]

PAGE NO.
DATE :

Qu :- Producer.

```
Spinal-
    produce item;
    if count = 1 then sleep
    Place item in buffer
    count = = 1
    wakeup (consumer);
forever;
```

Consumer

```
Spinal
    if count = = 0 then sleep
    Remove item from buffer
    count = = 0
    wakeup (producer)
forever consume item;
```

using buffer size = 1 & Assume initial value of count = 0.
Assume that fetching & assignment are atomic.
Is it possible that both process will go in sleep mode at same time.

Not possible [✓] because count is a single variable. so either it will be 0 or 1. So only one process sleep at a time.

PAGE NO.
DATE :

How OS allocate blocks to data :—

⇒ In SM internal fragmentation always be and when OS allocate blocks in contiguous manner then external fragmentation is also possible.

Directory Structure is data structure that contain info of file.

File=Info :— general attribute (file info. depend on file sys^m)

1. File name
2. Size of file
3. Owner of file
4. Physical loc in disk where file is store.
5. protection info (read, write, execute)
6. Date of creation, date of last modified.

⇒ File info depend on file sys^m.

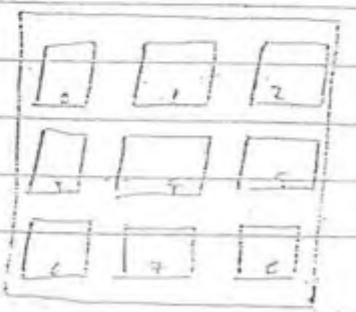
⇒ Directory Structure should not be greater in size as it is stored in MM. So some critical info is stored in directory structure and other are stored in FCB. (File Control Block)

⇒ Every file has its own FCB.

⇒ Every partition has its own directory structure.

BLOCKS ALLOCATION TO FILE :— There are no. of technique the simplest one is contiguous allocation of block.

1. Contiguous Block Allocation :—



PAGE NO.
DATE :

File 1 = 800 B

File 2 = 40 B

File 1 - block 0, 1
File 2 - block 2

⇒ Problem is external fragmentation.

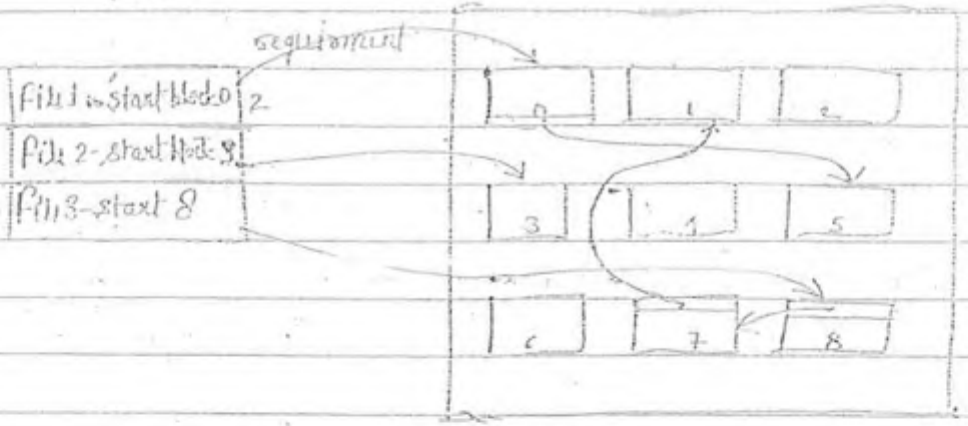
⇒ Benefit :- ① easy to implement
② read & write are fast

2. Linked Allocation :- (related to FAT file sys^m) -
Block can be allocated anywhere.

File 1 = 800 B

File 2 = 40 B

File 3 =



~~request arrives for file 3 block 1~~

1st movement for block 8 (read content, transfer pointer)

2nd movement for block 7

3rd movement for block 1.

- for reading one block sys^m has to read 3 blocks.

so which stored in RT so it will take more time.

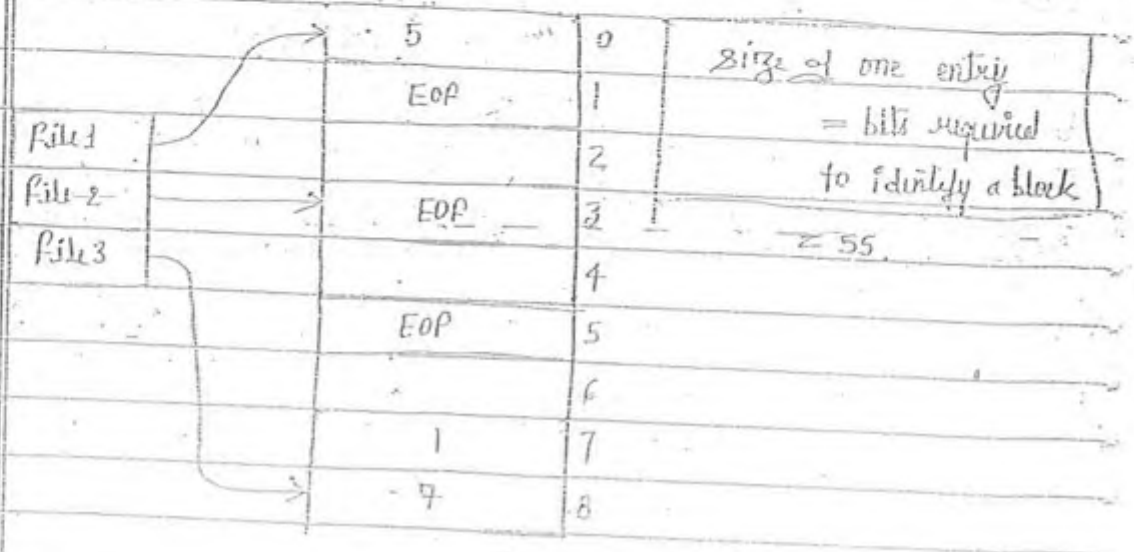
PAGE NO.
DATE :

⇒ FAT (File Allocation Table) is created in Linked Allocation.

FAT

Stored inside SM but when OS execute it brings FAT to Main Memory.

no. of entries in FAT = no. of blocks in SM



Bcoz FAT is in main memory so time taken to find contents of table is very less. than reading blocks in SM.

FAT SIZE = (no. of entries = no of block size in SM) X (size of one entry = no. of bits required to identify a block.)

FAT 16 means :-

size of one entry in FAT = 16 bit

FAT 32 means :-

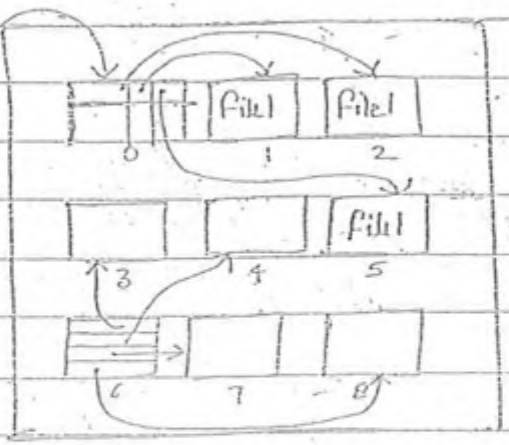
size of one entry in FAT = 32 bit

If FAT is corrupted then data can't be accessed.

PAGE NO.
DATE :

3. Indexed Allocation Table :-

	requirement
File1 - start 0	8 block
File2 - start 6	4 block



File 1 data is stored in 1, 2, 5 block.

File 2 data is stored in 3, 4, 7, 8 block, index block = 6

size of pointer (address) = 32 bit = 4 B

size of one block = 512 byte

max size of file supported by this indexed data structure = 512/4 B

= 128 no. of pointer

total size = (no. of block of pointer) × size of block

= 128 × 512 B

= $2^7 \times 2^9$ B = 2^{16} B

⇒ File greater than max. size can't stored.

PAGE NO.
DATE :

GATE 2008 - 1998

Gate 1998 :-

Que :- counting semaphore was initialized to 10

6

Que :- Consider five 8-tape drive with n processes. Each process may need 2 drive. What is the max value of n for which sys will be deadlock free.

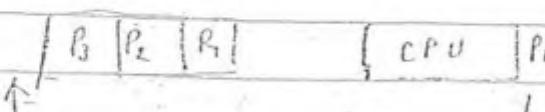
max need < m + n process = n

2n < 8 + n max need = 2n

n < 8

max value of n = (5).

Que :- n no. of processes sharing CPU in RR fashion. The context switch of each process takes s sec, what must be quantum size q such that overhead resulting from process switch is minimized but at the same time each process is guaranted to get its turn at the CPU at least every t sec :-



s sec (P1 context switch)
q (P2 execute)
s (P2 context switch)
q (P3 execute)
s (P3 context switch)

PAGE NO.
DATE :

total overhead - $ns + (n-1)q$ (max value)

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

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

$q \geq \frac{t-ns}{(n-1)}$

Ans :- Instruction takes i μsec and page fault take additional j μsec. The eff. Instruction time if only one a page fault occurs every k instruction.

Page fault = $1/k$

Page hit = $1 - 1/k$

Eff Access time = $(1-k)i + 1/k \times (i+j)$

$= i - i/k + i/k + j/k$

$= i + j/k$

Ans :- Partition size in KB 4K, 8K, 20K & 2K

Job size in KB. = 2K, 14K, 3K, 6K, 6K, 10K, 20K, 7K

Execution time = 4 10 2 1 4 1 8 6

Computer sys^m uses best fit algo for allocating memory space to this job. When will the 20K job complete.

20K	14K/6K/10K	Empty at $10+1 = 11$
8K	5K	Empty at 4
4K	3K	Empty at 2
2K	2K	Empty at 1

PAGE NO.
DATE :

Goth 2000 :-

Que :- Which of the following need not necessary saved on a context switch b/w processes.

(a) GPRs. (b) Translation look-aside buffer [✓]

(c) Prog Counter (d) All the above [✗]

Que :- Let m[0] ... m[4] mutexes.

P[0] ... P[4] processes.

wait(m[i])

wait(m[(i+1)%4])

—

—

release(m[i])

release(m[(i+1)%4])

deadlock [✓]

Que :- Suppose the time to service a page fault on the avg is 10 ms. while memory access takes 1 μsec. Then 99.99% hit ratio result in avg memory access ?.

Hit ratio = 99.99% = .9999

Avg Access Time = 0.9999 × 1 μs + (1 - .9999) × 10 ms

= 0.9999 × 10^{-6} sec + (1 - .9999) × 10 ms

= 0.9999 × 10^{-6} sec + (1 - .9999) × 10 × 10^{-3} sec

= 1.9999 μsec.

Que :- The soln of readex writer problem given, complete code. - using single binary semaphore. (Hint use

PAGE NO.
DATE :

Int R=0, W=0

Reader()
{
L1: wait (mutex)
if (W==0) {
R = R+1;
{ [ ] 1
else { [ ] 2
go to L1;
}
/* do read */
wait(mutex)
R=R-1
Signal (mutex)

Writer()
{
L2: wait (mutex)
if ([3]) {
Signal (mutex);
go to L2;
}
W = 1;
Signal (mutex);
/* do write */
wait (mutex);
W=0
Signal (mutex);

Gate

Que

Que

for

no.

A

(b)

(c)

(d)

Que

Que

Value

PAGE NO.
DATE :

1: signal(mutex)
2: signal(mutex)
3. R>0 or W>0.

Gate 2001 :-

Que :- Where does the swap space reside
= disk [✓] (secondary memory)

Que :- Virtual memory sys^n, with FIFO page replacement for an arbitary page access pattern increasing the no. of frames in MM will :-
(A) Always decrease no. of page fault
(b) Always increas no of page fault
(c) Sometimes increase the page fault [✓]
(d) never affect the no. of page fault

Que :- Which of the following doesnot interrupt a running process
(a) device (b) Timer
(c) Power failure (d) Scheduler process [✓]

Que :- mutual
flag [i] = true
turn = j
while (P) do no op ;
<CS>
flag [i] = false ;
<RS>
while false
Value of P ? value -

PAGE NO.
DATE :

flag[1] = true && turn = 1 [✓]

Que:- Consider the disk with following specifications-
20 - surfaces
1000 track / surface
16 sector / track
1 kB data density / sector
3000 rpm rotation speed

(a) total capacity of disk ?
(b) data transfer rate ?

Capacity = 20 × 1000 × 16 × 1 kB
= 320000 kB
= $2 \times 10^4 \times 2^4 \times 2^{10}$ B
= $2^{15} \times 5^4$ B

transfer rate :-
capacity of one track = 16 KB
rotational speed = 3000 rpm
1 min revolution = 3000
1 " " = $\frac{3000}{60} = 50$
1 rev takes = 1/50 sec
transfer rate = $\frac{16}{1/50}$
= 16 × 50 KB/sec.

Que :- Two concurrent processes P_1 & P_2 and two resources R_1 & R_2. Processes use the resources in mutual exclusion manner. Initially R_1 & R_2 are free.

P_1	P_2
S_1 while (R_1 is busy) do nop;	Q_1 while (R_2 is busy) do nop;
S_2 Set $R_1 \leftarrow$ busy;	Q_2 Set $R_2 \leftarrow$ busy
S_3 while (R_2 is busy) do nop;	Q_3 while (R_1 is busy) do nop;
S_4 set $R_2 \leftarrow$ busy;	Q_4 set $R_1 \leftarrow$ busy
S_5 use R_1 & R_2;	Q_5 use R_1 & R_2
S_6 set $R_1 \leftarrow$ free;	Q_6 set $R_2 \leftarrow$ free;
S_7 set $R_2 \leftarrow$ free;	Q_7 set $R_1 \leftarrow$ free;

(a) mutual exclusion guarded on R_1 & R_2

(b) can deadlock occur

Change Q_1 to Q_3 and Q_2 to Q_4 then –

mutual Exclusion is not guaranteed on R_1 & R_2 [V]

deadlock can't occur

Que: Disk with 100 tracks numbered from 0 to 99. Rotating with 3000 rpm. no. of sectors/track is 100. The time to move the head b/w two successive tracks.

Consider a set of disk request to read data from track. Algo is SCAN elevator & initially track no 25. moving up (toward larger track no.).

What is the total seek time for servicing the request.

50, 7, 45, 5 & 10

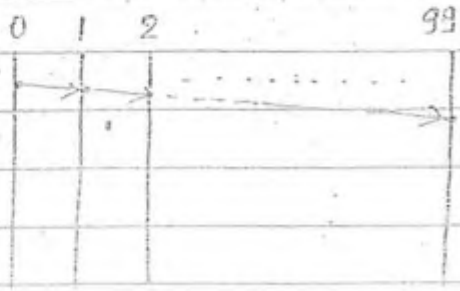
PAGE NO.
DATE :

$[(52-25)+(15-32)+(93-45)+(99-10)+(10-2)+(2-5)]$

$\times 0.2$ ms

$= 33.6$ ms.

[b] Consider the initial set of hundred arbitary disk request and assume that no new disk request occurs while servicing these request. If the head is initially track no. 0 & elevator algo is used to schedule disk request. what is the worst case time to complete all the request.



In worst case Head has to used whole track. So

seek time = $99 \times .2$ ms

rotational latency + block transfer time

= 100 [one complete revolution time]

rpm = 3000

60 $\frac{3000}{\text{sec}}$

1 complete revolution time = $1/50$ sec

rotational latency = $\frac{100 \times 1}{50}$

= 2 sec

19.8 ms + 2 sec = 19.8 + 2000 ms

= 2019.8 msec

PAGE NO.
DATE :

GATE 2002

Que :- following features will characterize an OS as a multi program OS.

(1) more than one prog can be loaded into main memory at same time for execution.

(2) Program wait for certain event such for i/o, another program immidiately scheduled for execution

(3) Execution of program terminate another program immidiately, schedule for execution.

(a) I (b) I & II [✓]

(c) both none (d) no one

Que :- Index allocation scheme of a block to a file, max size of file depend on

(a) size of block & size of address of block

(b) no. of block used for index & size of block

(c) size of block, no of blocks used for indexing & size of address of block. [✓]

(d) none of the above

Que :- Test And Set

```
int TestAndSet (int *x)
{
int y;
A1: y = *x;
A2: *x = 1;
A3: return y;
}
```

PAGE NO.
DATE :

for achieving mutual exclusion.

```
int mutex = 0
void enter_cs()
{
while ( 1 )
{
<CS>
void leave_cs()
{
2
}
```

1. TestAndSet (& mutex)
2. mutex = 0;

Ex :- Computer uses 32 bit virtual address & physical address. Physical memory is byte addressible. Page size = 4 KB. Two level page table for translation virtual address to physical address. Equal level of bits should be used for indexing 1st level & 2nd level of table. Size of each page table entry is 4 B.

[a] What is no. of page table entries that can contain in each page ?

page size = 4 KB
Address = 32 bit offset = 12 bit
remaining part will be divided in 2 part
20 bit for indexing in 1st level
10 bit " " " 2nd level.

entry size = 4 B

no. of entries in one page = page size / size of entry
= 4 KB / 4
= 1 K = 1024

PAGE NO.
DATE :

b] How many bits are available for storing protection & other info in each page table entry.

Size of physical memory = 2^{32} byte

frame size = size of page = 4 KB

$= 2^{12}$ B

No. of frames in MM = $\frac{2^{32} B}{2^{12} B}$

$= 2^{20}$ frames.

20 bits are used for address

So 32 − 20 = 12 bits used for protection & other info.

Que :-

```
# define BUFFSIZE 100
buffer buf[BUFFSIZE];
int first = last = 0
semaphore b_full = 0;
semaphore b_empty = BUFFSIZE
void producer ()
{
  while (1)
  { produce an item }
  f1 [ ]
  buf[first] = item;
  first = (first + 1) % BUFFSIZE
  f2 [ 1 ]
}
void consumer ()
{
  while (1)
```

PAGE NO.
DATE :

{

G_1 [ ]

take the item from buffer

buf [size]

last = (last + 1) % BUFFSIZE

G_2 [ ]

consume item

}

}

⇒ code for one producer & consumer

P_1: wait (b.empty) C_1: wait (b.full)

P_2 signal (b.full) C_2: signal (b.empty)

Another semaphore variable. Insert one stmt just after P_1.

mutex = 1

After P_1: wait (mutex) After C1: wait (mutex)

Before P_2: signal (mutex) Before C2: signal (mutex)

Gate 2005

Que 1:- Ans (a)

Que 2:- VAS = 32 bit pages = $2^{32} / 2^{10}$ B = 2^{22} B

page size = 1 KB page size will increase so

Ans (c) main memory have large memory overhead.

PAGE NO.
DATE :

Gate 2001 :-

User prog are prevented to handling

I/O Instructions

I/O mapped (I/O has explicit instruction)

Memory Mapped. (All memory related instruction are used for I/O)

consists privileged instruction

executed only in kernel mode.

after

Gate 2009 :-

⇒ CPU checks the interrupt after completion of instruction and before executing next instruction

⇒ Belady anomaly occurs in FIFO.

⇒ Essential entry in page table - page frame no.

Que 43 :-

four type of resources - R1 R2 R3 & R4

to

Unit 3 2 3 2

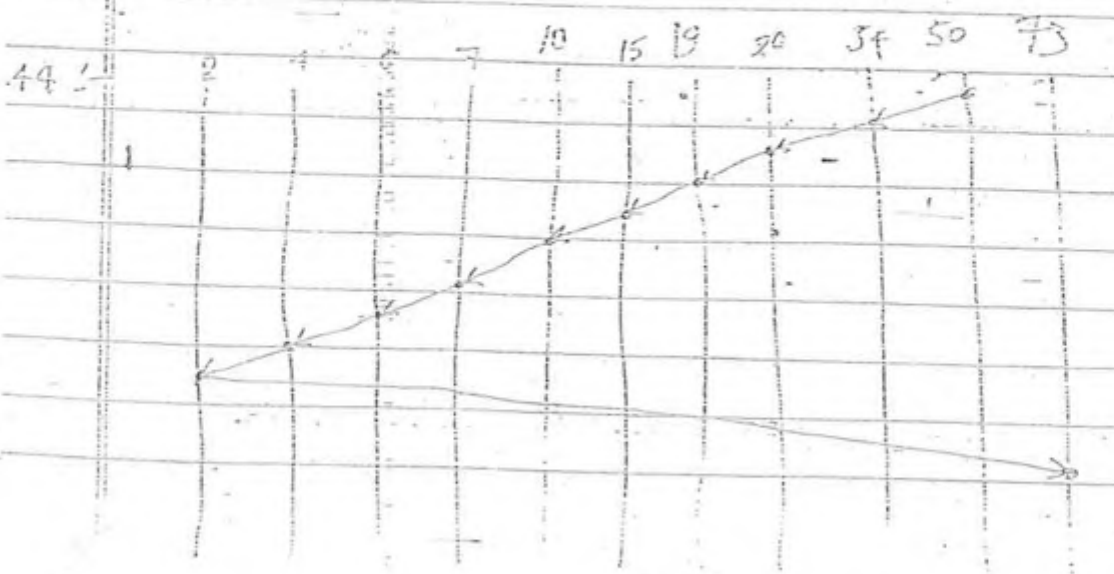
P1 P2 P3

t = 0 request

PAGE NO.
DATE :

	R_1	R_2	R_3	R_4
$t=0$	3	2	3	2
$t=0$	2	0	1	1
$t=1$	3	0	0	1
$t=2$				
$t=3$	1	0	0	0
$t=4$	0	3	0	0
$t=5$	0	0	0	0
$t=6$	0	0	1	0
$t=7$	1	0	1	0
$t=8$				
$t=09$				
$t=10$				

	P_1	P_2	P_3
$=0$	$R_2=2$	$R_3=2$	$R_4=1$
$=2$	$R_3=1$	$R_4=1$	$R_1=2$
$=3$	waiting (R_1)		
$=4$		$R_1=1$	
$=5$	$R_1=2$	$R_1=$	



PAGE NO.
DATE :

⇒ Different heads are always at same cylinder no.

(80-34) + (34-20) + (20-19) + (19-15) + (15-10) + (10-7)
+ (7-6) + (6-4) + (4-2) + (73-2)

= 15 + 14 + 1 + 4 + 5 + 3 + 1 + 2 + 2 + 71

= 78 + 71

= 119 ms

Ques 48:-

Note 2008

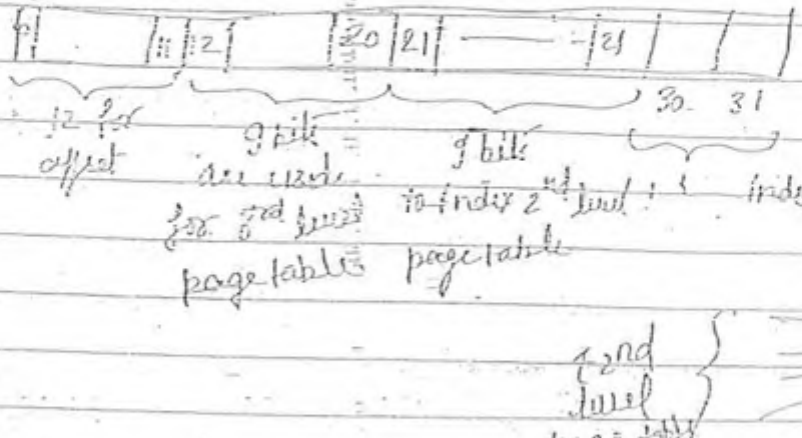
Ques 49:-

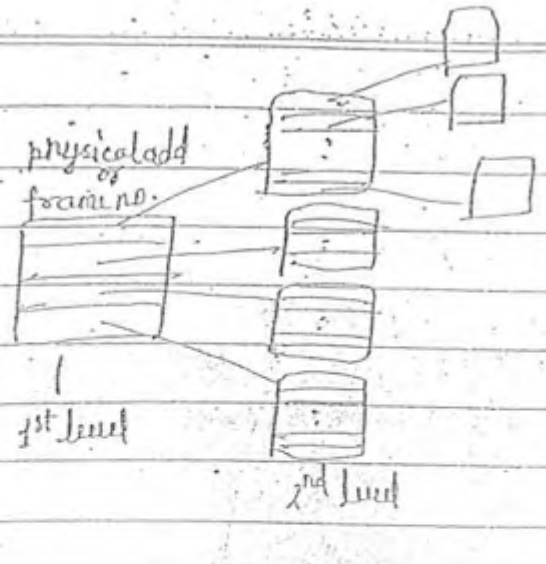
Physical address = 36 bit

VAS = 32 bit

Page frame size = 4 KB

Each page table entry size = 4 B





No. of frames in physical memory = $2^{36}B$ / $2^{12}B$ = 2^{24}

Every page table contain 2^{24} bit to point next level of page table

Que 87

Two type of I/O

Synchronous	Asynchronous
Process (sys call)	Process
Device Driver	Device Driver
Kernel	Kernel
H/w	H/w

open("file1"); // block.

⇒ In syn I/O sys call is blocked till I/O completes.

⇒ In asyn I/O sys I/O fn return some value w/o executing the code.

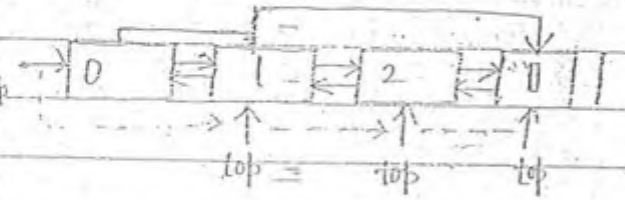
PAGE NO.
DATE :

| 0 | 1 | 2 | 0 | 3 | 2 | 0 | 3 | 4 | | | |

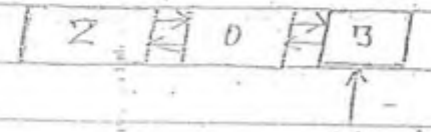
ault
How

⇒ time to use is more then it is reused recently use LRU which page has less time to use is removed.

using link list :—



page 1 is removed.



⇒ LRU is time consuming because for each value it maintain data structure so Approx LRU algo is used in sys^m. LRU needs hw support.

4. APPROXIMATE LRU :

① Reference bit Algo ④ Enhance Second Chance Algo.
② Additional Reference bit Algo
③ Second Chance Algo

1. Reference bit Algo :—

page table	V/I	Reference bit
page 0	I	0
page 1	V	1
page 2	V	1
page 3	V	1
page 4	I	0

reference string - 1, 1, 2, 3, 4

1	2	3			

nder for

⇒ If the reference bit = 1 then sys^m will not remove page but if 0 then it can remove & V/I bit should be 0 for deletion.

PAGE NO.
DATE :

⇒ Given table CPU can't decide that which page is removed but if there were some page which are in memory in starting then this algo will work

Suppose 0, pages are available before -

page		V/I	0
1		I/V	0/1
2		I/V	0/1
3		I/V	0/1
4		I	0

0 1 2 3 1 2

⇒ This algo fails when all bits are 1, so 2nd algo is used.

2. Additional Reference Bit Algo :-

Sys^m takes 5 (for 5 pages) register and size = 4 bit (Assume)

page 0	0				
1	1				
2	1				
3		1			
4					

page	V/I	reference bit
1	I/V	0/1/0
2	I/V	0/1/0
3	I	0/0
4	I	0

reference string = 1, 1, 2, 3, 1, 4

⇒ After some time copy the reference bits to register and reset all reference bit. Initially all reg are 0 after copying reset all reference bit

1	2	3	1	4	3

⇒ Before copying value to register it will shift all the value of reg. in 1-bit right

PAGE NO.
DATE :

Explanation :-

Step 1:-

page 1	V	1
page 2	V	1

Copy reference bit into register msb

page 0	0
page 1	1
page 2	1
page 3	0

page 3	V	1

page 4 required page replacement. So it first shift register value and copy next reference bit

					Convert to decimal
page 0	0	0	0	0	0
1	1	1	0	0	⇒ 12
2	0	1	0	0	⇒ 4
3	1	0	0	0	⇒ 8

4 is less so it is least recently used so it is replaced so page no. 2 will be replaced.

page 4	V	1

⇒ If all msb's will be high the and remaining bits values are same then we can't decide which value page is replaced, Eg :-

page 0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0

⇒ In this case we will use FIFO Technique.

PAGE NO.
DATE

3. Second Chance Algo :—

V/I Reference bit ⇒ Circular list

page 0	V	1	⇒	page is in memory & referred bytes
page 1	V	1		
page 2	I	0	⇒ page is not in MM	
page 3	V	0	⇒ page	is in MM but not referred by CPU
page 4	I	0		
page 5	I	0		

0	1	3

When search for removing a page it change ref bit 1 to 0 here

It start searching from page 0 & change ref bit from 0 to 1 & when find V w/ 0

Suppose I/O to page 0

page 1	V	0
page 2	I	0
page 3	I	0
page 4	V	1
page 5	I	0

0	1	4

5

Its first bit page get 0 it removes that page and insert the new page

⇒ no of iteration = 1+1

now for page 5 it will start searching from page 0 and first it will find page 0 with V – 0 so page 0 will be remove

page 0	I	0
page 1	V	0
2	I	0
3	I	0
4	V	0
5	V	1

5	1	4

PAGE NO.
DATE :

4. Enhanced Second Chance Algo :—

⇒ It is based on reference bit & modified bit. so total no of combⁿ = 4

Reference bit	Modified bit	
0	0	⇒ removed first
0	1	⇒ not possible
1	0	⇒ removed at 2nd place
1	1	⇒ removed in last.

for removing it first CPU replace it and send it to secondary memory and then bring page from secondary memory to main memory

⇒ In third case only one operation bcoz page is not modified so while replacing no need to copy it into secondary memory. only new page will be brought to MM from SM.

⇒ no. of iteration will be > 1.

SOME OTHER TECHNICS :—

① Most frequently used page

② least frequently used page

⇒ In most frequently used page, the page which are most are removed first.

⇒ In least frequently used pages, the page which are used least are removed first

⇒ It can't be decide that which is better in this ; it depend on reference string.

PAGE NO.
DATE :

FRAME ALLOCATION :— How many frames will be allocated to process

MM

no. of frames = f

$P_1, P_2, P_3, \cdots P_n$

size of process

$S_1, S_2, S_3 \cdots S_n$

$$\sum_{i=1}^{n} S_i = S_1 + S_2 + S_3 + \cdots + S_n$$

no. of frames allocated to process P_1 =

$$\frac{S_i}{\sum_{i=1}^{n} S_i} \times f = \frac{\text{size of process}}{\sum_{i=1}^{n} S_i} \times \text{no. of frames}$$

⇒ If size of process is less than less no of frames are allocated and if size of process is large then more no. of frames are allocated to the process

min no. of frames ?

code	ADD M_1, M_2	page 0
data	$M_1 = 5$	page 1
data	$M_2 = 10$	page 2

ADD M_1, M_2

$[M_1] \leftarrow [M_1] + [M_2]$

page 0
page 1
page 2

min no. of frames for this instruction = 3

PAGE NO.
DATE :

⇒ no. of min frames depend on the instruction upon instruction set architecture. Instructions are depend upon computer Architecture so it will also depend upon comp. Arch..

max no. of frames :-

max no. of frames = max no. of frames into memory.

Local Vs Global Allocation (Page Replacement)

Page 0	f0	P_1 {0,1,2,3}
Page 1	f1	
Page 4	f2	P_2 {4,5,6,7,8,9}
Page 5	f3	
Page 6	f4	
Page 7	f5	
Page 10	f6	P_3 {10,11,12}
Page 11	f7	

P_1 during execution generate reference page 2. Then process P_1 will replace its own page. This type of replacement is called Local Replacement. This technique is less flexible (because it has only 2 frames and after replacement no. of frames will not increase)

A process can replace pages of any other process this type of replacement is called global page replacement. Let page no 5 is replaced of P_2 by page 2 of process P_1 Here no of frames are increasing of P_2 and no. of frames decrease of P_1, so it is flexible.

PAGE NO.
DATE :

Active Page :- used by process frequently.

If active page in MM the less no. of page fault

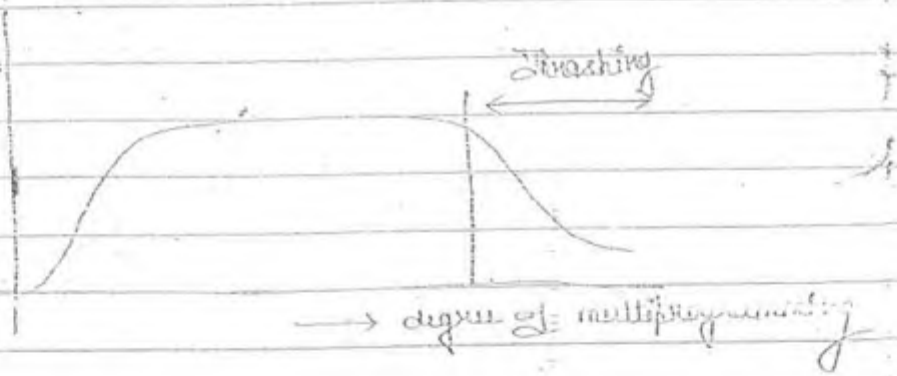
degree of multiprogramming :- no of process loaded in MM at a time.

→ degree of multiprogramming is increased to increase the CPU & resource utilization.

If page P_1 demand the active page of P_2 then when P_2 execute it need active page here occure page fault then P_2 will remove any other process Active page. This is called Thrashing.

working window of size 4

Thrashing is a case of sys when sys is in page replacement instead of execution of process.



Thrashing will always occure in Global replacement. It will not occure in Local replacement.

Global Replacement	Local Replacement
① Thrashing occure	Thrashing does not occure
② more flexible	less flexible
③ more robust	less robust

PAGE NO.
DATE :

How to Avoid Page fault :—

① Load All active pages of the each process to the MM.

WORKING PAGE MODEL :—

decides which is active page & which is passive.

$P_2 = t1$

1, 2, 1, 3, 4, 1, 5, 6, 7, 1, 3

working set ⇐ window size 4

{1, 3, 4}

↳ set of Active page.

⇒ Active page decision depend on size of working set window.

⇒ If size of working set is small than some active process it will not contain and if it is big then it will contain some passive pages. So size of working page model should be optimized.

Remove degree of multiprogramming to reduce page fault :—

Assume that there is n no. of process then —

$P_1, P_2, \cdots P_n$

WSS1, WSS2, ..., WSSn ⇒ set of active pages of each process.

$$\sum_{i=1}^{n} WSS_i \leq \text{no. of frames}$$

If it is not satisfied then Any process active page will not be in MM and page fault will occurs and and it will again lead to page fault.

If this cond^n occurs the OS will choose a process then unroll all page in SM and suspend the process. Allocate the free frames to active pages of other process means OS is decreasing degree of multiprogramming till cond^n achieved.

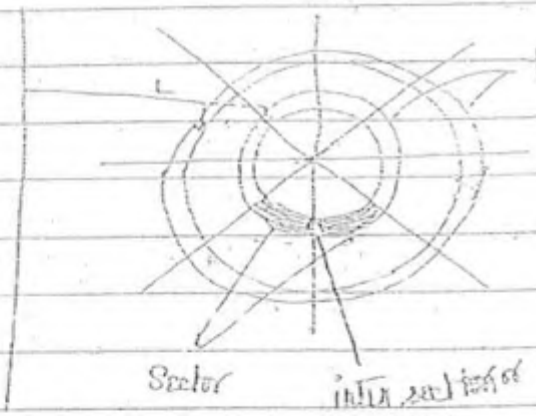
PAGE NO.
DATE :

DISK STRUCTURE AND SCHEDULING :-

⇒ Platter may be single coated or double coated

⇒ HDD contains 16 platter

⇒ track is physical entity & is divided into sector.

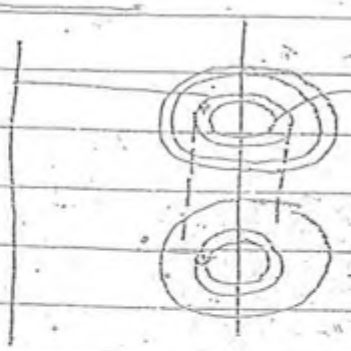


platter
track
sector
< platter no.
side
track
sector >

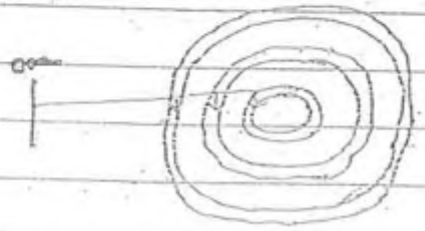
10th Aug

PAGE NO.
DATE :

DISK ACCESS TIME



no of cylinders = no of tracks.



Seek time : Any time taken by head from one track to another.

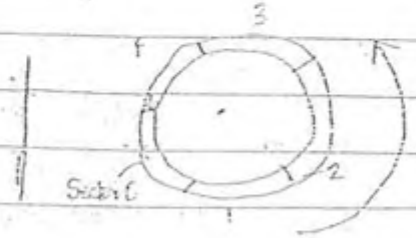
⇒ movement time from one track to other is not same so we take avg time.

track 0 to track 1 — 1 ms
track 1 to track 2 — 1 ms
track 0 to track 1 time taken by head = 10 ms
track 1 to track 2 time taken by head = 10 ms

Average seek time = 20/10 ms

time taken by head to go from track
to track 2 = 10 ms

Rotational delay or Latency :



(constant avg velocity)

PAGE NO.
DATE :

sector no_1 is needed by sys^m then it will take time to go under head.

time taken by track 1's sector1 > time taken by track 1's sector2.

⇒ Rotational Latency is not fixed. so we take avg value.

Rotational latency = one complete revolution time / 2

If Angular speed of disk = 5200 r/min (revolution time)

1 min no. of revolution = 5200

1 sec " " " = 5200/60

time taken by disk in one complete revolution = 1/5200/60

= 60/5200

Rotational Latency = 60/5200×2 sec.

Block Transfer Time :-

⇒ Block is logical entity.

→ Acc to OS HDD is devided into block and size of block is fixed.

suppose block size = Size B.

Transfer rate (tr) = capacity of track / one complete revolution time

block transfer time = size of block / transfer rate

PAGE NO.
DATE :

Block Access Time = seek time + rotational time + block transfer time.

Ques:- A HDD sys^m has following parameter

no. of track = 500

no. of sector = 100/sector

no of byte = 500/sector

Time taken by head to move from one track to adjacent track = 1 ms.

rotational speed = 600 rpm

What is the avg time taken for transferring 250 byte from the disk.

⇒ Here size of data is less that the size of sector so the time = 1 latency time.

Avg seek time = $\frac{499 \text{ ms}}{2}$

= 249.5 ms

Rotational delay =

speed = 600 rpm

600 revolution in 1 min

600 " " 60 sec

in 1 sec = $\frac{600}{60}$ 10.

1 sec track = 1/10 = .1 sec

Rotation delay = .1/2 = 0.05 sec

.05 sec × 1000ms = 50 ms

Data Transfer Rate = 100 × 500 (one track capacity)

0.1 sec

= 5×10^5 B/sec

If track 3 then distance = 2 cm

so within track = 500

then distance 100 cm

PAGE NO.
DATE :

Transfer time = $\frac{50}{250 B}$ $= 5 \times 10^5$ B/sec

5×10^5 B/sec

$= 50 \times 10^5$ B/sec

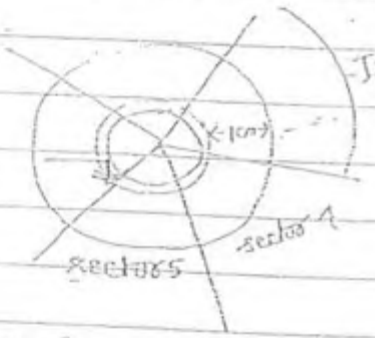
$= 50 \times 10^{-2}$ ms

$= .5$ msec

$= (249.5 + 50 + .5)$ ms

$=$ **300 ms.**

2. If disk has 20 sector/track and head is currently at the end of 5th sector of innermost track. Head can move at the speed of 10 m/sec. Disk is rotating with constant angular velocity 600 rpm. How much time will it take to read 1MB contiguous data, starting from the sector 4 of the middle most track.



seek time = $\frac{\pi \text{ cm}}{10 \text{ m/sec}}$

$= \frac{\pi}{10 \times 10^2}$ cm/sec

$= \pi \times 10^{-3}$ sec

$=$ **7 msec**

time taken by head to reach at sector 4 = $\frac{\text{one complete revolution time}}{20}$

one comp. rev time = $\frac{60}{600}$ rpm

$\frac{600}{60} =$ **10 no. of time / sec.**

PAGE NO.
DATE :

$= 1/100$ sec

$= 1000/100$ ms

$= \boxed{10 \text{ ms.}}$

Time taken by one sector to pass during head $= \frac{10 \text{ ms}}{20}$

$= \boxed{.5 \text{ ms}}$

one comp sec takes 10 ms

so 1 sector is less bcoz we have to find sector 4, then time taken = 9.0 ms. (bcoz head is also moving, and disk is rotating at same time so at the time head will go to track 1 and at same time disk will rotate and head will go to start of sector 4. so time taken by head in moving and not to count).

Transfer rate $= \frac{10 \text{ MB}}{10 \text{ ms}} = \frac{10 \text{ MB}}{10 \times 10^{-3}} = \boxed{10^3 \text{ MB/s}}$

(capacity of one comp / track sec time)

Transfer time $= \frac{\text{Data size}}{\text{Transfer rate}}$

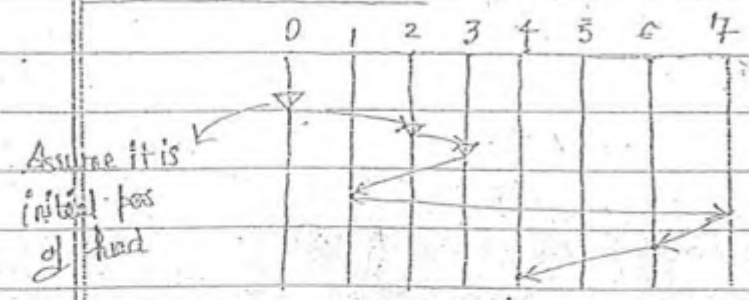
$= \frac{1 \text{ MB}}{10^3 \text{ MB/s}} = \boxed{1 \text{ msec}}$

Total time = 9 ms + 1 ms

$= \boxed{10 \text{ ms.}}$

PAGE NO.
DATE :

DISK SCHEDULING :— Only seek time will be considered.



SM buffer has no requests

2, 3, 1, 7, 6, 4

Thus all the track no. from which head has to read the data.

1. FCFS :—

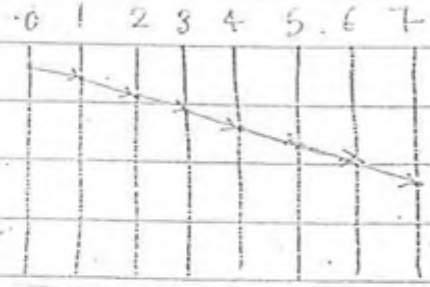
no. of head movement

$= (2-0)+(3-2)+(3-1)+(7-1)+(7-6)+(6-4)$

$= 2+1+2+6+1+2 = \boxed{14}$

⇒ simple but no. of head movement is more

2. Shortest Seek Time First :



no. of head movement $= (1-0)+(2-1)+(3-2)+(4-2)+(6-4)+(7-6) =$

$= 1+1+1+1+2+1$

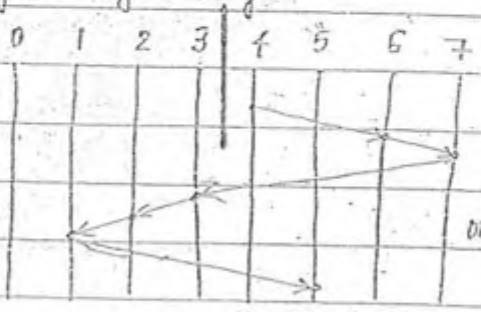
$= \boxed{7}$

⇒ It is best as compare to FCFS but it is not necessary that it will always give best result.

PAGE NO.
DATE :

3. SCAN :- Head moves from left to right and comes at last track from where it changes the direction and move from left to right. Also called as ELEVATOR.

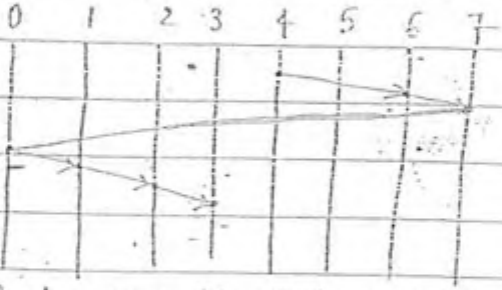
Initial pos. of head at track no. 4 and initial direction of head from left to right.



⇒ Head always chang direction either start of track or end of track

no. of head moves = (4-4) + (6-4) + (7-6) + (7-3) + (3-2) + (2-1)
= 0 + 2 + 1 + 4 + 1 + 1
= 9

4. CSCAN (Circular SCAN) :- Head can scan the req. in one direction either left to right or right to left and change the direction either start or end of the track.

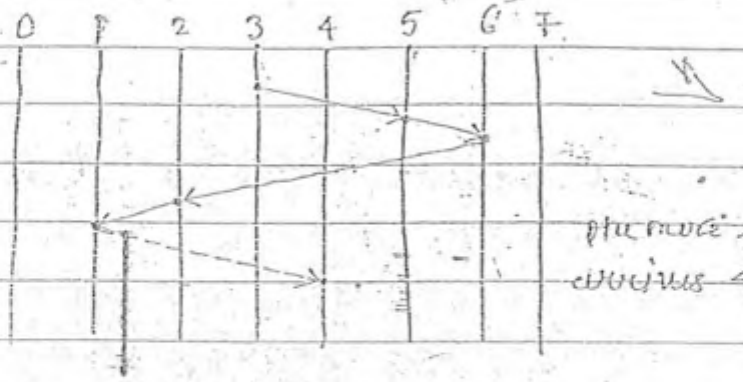


no. of head moves :- (4-4) + (6-4) + (7-6) + (1-0) + (2-1) + (3-2)
= 0 + 2 + 1 + 1 + 1 + 2
= 7

11th Aug 09

PAGE NO.
DATE :

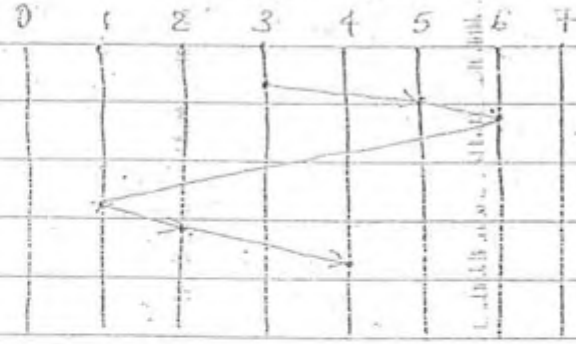
Look :- Head can change the direction in between tracks.



request are - 1, 2, 5, 6
initially head at - 3
initial direction - left to right

no. of head moves = (5-3) + (6-5) + (6-2) + (2-1) + (4-3)
= 2 + 1 + 4 + 1 + 3
= 9 + 3
= 11

C-Look :- Head can change direction i b/w track but can serve in only one direction



initially head at - 3
direction left to right

no. of head movement = (5-3) + (6-5) + (6-1) + (2-1) + (4-3)
= 2 + 1 + 5 + 1 + 2
= 11

PAGE NO.
DATE :

Look & SSTF are implemented in h/w. generally these two are supposed good and SSTF gives half head movment as compare to FCFS.

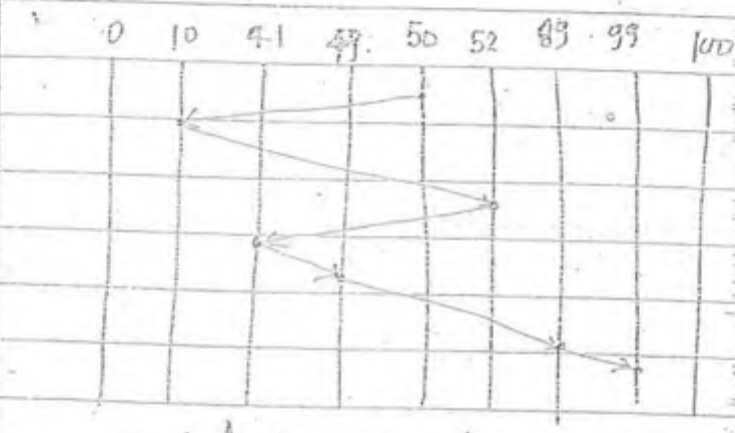
Qu:- Disk has 100 no. of track ; all track are equidistance and distance b/w two adjacent track = 1 cm. Head move horizontally with speed 1 m/sec.

Request are 10, 52, 41, 49, 89, 99

Algorithm = FCFS, SSTF & Look

Find the average seek time for head. Assume that initial pos of head at track no 50 and direction is left to right.

1. FCFS :-



No. of head moves = (50-10) + (52-10) + (52-41) + (49-41) + (89-49) + (99-89)

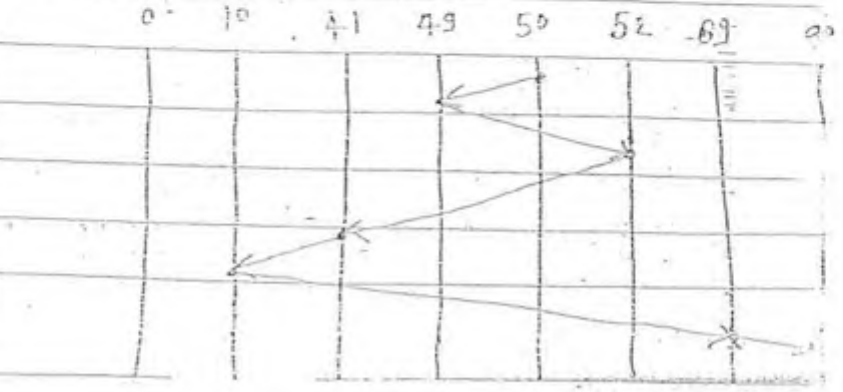
= 40 + 42 + 11 + 08 + 40 + 10

= 151

Speed = 1 m/sec = 100 cm/sec time = 151 cm / 100 cm/sec

Avg seek time = 1.5 sec / 6 = 10

2. SSTF :-

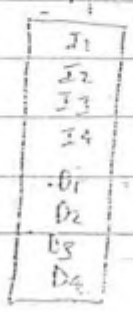


PAGE NO.
DATE :

Qu 45 :- Consider a system with 2-level paging scheme in which unpaged memory access takes 150 nsec. Servicing a page fault is 8 msec. An average instruction takes 100 nsec of CPU time and 2 memory access. TLB hit ratio is 90% and page fault rate is one in every 10,000 instructions. What is the effective average instruction execution time.

1 instruction takes = 100 ns + 2 (memory access time)

total no. of memory



$t_t = 0$

TLB

Access time of memory

$t_m = 100$ ns

page table

90%

hit ratio = 1/20,000

1 page fault in 2000 instruction

2 memory reference so page fault = 20000

⇒ Address divided into two part page & offset. Page is searched into TLB, and then memory is searched.

Average effective time w/o page fault =

$= .9 \times (t_t + t_m) + (1 - .9) \times (t_t + t_m + t_m)$

$= .9 \times (0 + 150) + (.1) \times (0 + 150 + 150)$

$= 90 + 30 = 135 + 30$

$= 165$

Effective Access time with page fault =

(1 − page fault rate) (memory access time) + page fault rate × page service time

page fault rate = $.5 \times 10^{-4}$

$= \left(1 - .5 \times 10^{-4}\right)(165) + .5 \times 10^{-4} \times 8$ msec.

$= \left(1 - .5 \times 10^{-4}\right)(165) + .5 \times 10^{-4} \times 8 \times 10^6$ ns

$= 165 + 4 \times 10^2$ ns

$= 565$ ns.

Actual effective time = $100 + 2 \times (565)$

$= 100 + 1130$ ns

$= 1230$ ns.

PAGE NO.
DATE :

Average = $p_1 \times (t_1 + t_m) + (1-p_1)(t_t + t_m + t_m)$

1 − pagefault × Average
+ pagefault × page service time

(2004, CS)

Qu 66 :-

Locality of reference

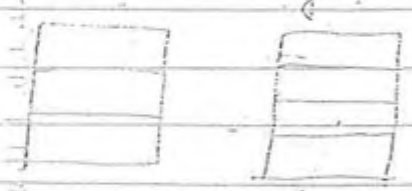
temporal locality of reference (related to time) — Spatial locality of reference (related to space)

Ans If instruction i is referenced by CPU in time $t = t_1$ then this will again reference in near future is known temporal locality of reference.

Ans If instruction i is referenced by CPU then its near by instructions will reference by CPU in near future. In Spatial Locality of reference.

→ Belady's Anamoly occurs beez of stack class algorithm.

Stack Class Algorithm :- System 1 has n no. of frame.
System 2 has (n+1) no. of frames.



frame = 3 frame = 4

reference string = 2, 1, 1, 3, 4, 1, 2, 0, 1

PAGE NO.
DATE :

- At any time if an algo satisfies this condⁿ

no. of page in sysᵐ 1 $\subseteq$ no. of pages in sysᵐ 2
(n frames) (n+1) frames

Then it comes under stack class algorithm.

⇒ FCFS doesn't satisfy this condⁿ. so Belady's Anamoly occurs.

Optimal Size of Page :

Suppose avg page size = s'
page size = p
and size of each entry in page table = e
total overhead = space required to store page table + internal fragmentation

no. of page required = s/p
no. of entries in page table = s/p
size of one entry = e
total space = s/p × e

Total overhead = $\frac{s}{p} \times e + \frac{p}{2}$ (suppose internal frag. is half page)

diff with respect to p (here p is variable)

$= -\frac{se}{p^2} + \frac{1}{2}$ (

$-\frac{se}{p^2} + \frac{1}{2} = 0$

$\frac{s}{p^2} = \frac{1}{2}$ $p^2 = 2se \Rightarrow \boxed{p = \sqrt{2se}}$

PAGE NO.
DATE :

FILE SYSTEM

File Sys^m is a ds maintain by os to control the files. when secondary memory is formatted by os.

HDD

- low level formatting (Track & sectors are created)
- high level formatting (file sys^m is created & SM is divided into logical entity called blocks.)

⇒ Every block has no. and size of block is decided by os during formatting. (Some times user may decide size of blocks)

⇒ If size of SM = 2^{64} B

size of block = 512 B

= 2^9 B

no. of block in SM = $\frac{\text{size of SM}}{\text{size of block}}$

= $\frac{2^{64}\ B}{2^9\ B}$

= 2^{55} B

So every block no. represented by 55 bits.

File :- An abstraction provided by os so that user can store their data w/o knowing the internal structure of SM.

File 1 — 8 KB block size = 4 KB

PAGE NO.
DATE :

Allocation algm = 11
time required = 8
19

Que :- Free disk space can be kept track of using a free list or bit map. Disk addresses required d bit. For a disk with b blocks, f of which are free, state the condn under which the free list uses less space than bitmap.

bit map
Size = B bits
Free list
free blocks = f every free block keep pointer of next free block.
Address of one block = d bits
Size of data structure = f×d bits
f×d bits < B bits.

Gate 1999 :-

Que ; Sysm calls are usually invoked by :-
(a) Sw interrupt [✓] (b) Polling
(c) Inclined jump (d) Privilidge Instruction [ ]

Polling - checking status of register continuously.

Que :- Advantage of virtual memory
(a) faster access of memory on an average [X]
(b) Process can be given protected [X]
(c) Linker can assign add independent of Program where it is loaded [ ]
(d) Program larger than size of physical memory can run [✓]

PAGE NO.
DATE :

Qu :- Producer.

```
Spinal-
    produce item;
    if count = 1 then sleep
    Place item in buffer
    count = = 1
    wakeup (consumer);
forever;
```

Consumer

```
Spinal
    if count = = 0 then sleep
    Remove item from buffer
    count = = 0
    wakeup (producer)
forever consume item;
```

using buffer size = 1 & Assume initial value of count = 0.
Assume that fetching & assignment are atomic.
Is it possible that both process will go in sleep mode at same time.

Not possible [✓] because count is a single variable. so either it will be 0 or 1. So only one process sleep at a time.

PAGE NO.
DATE :

How OS allocate blocks to data :—

⇒ In SM internal fragmentation always be and when OS allocate blocks in contiguous manner then external fragmentation is also possible.

Directory Structure is data structure that contain info of file.

File=Info :— general attribute (file info. depend on file sys^m)

1. File name
2. Size of file
3. Owner of file
4. Physical loc in disk where file is store.
5. protection info (read, write, execute)
6. Date of creation, date of last modified.

⇒ File info depend on file sys^m.

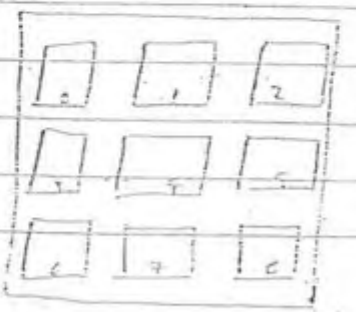
⇒ Directory Structure should not be greater in size as it is stored in MM. So some critical info is stored in directory structure and other are stored in FCB. (File Control Block)

⇒ Every file has its own FCB.

⇒ Every partition has its own directory structure.

BLOCKS ALLOCATION TO FILE :— There are no. of technique the simplest one is contiguous allocation of block.

1. Contiguous Block Allocation :—



PAGE NO.
DATE :

File 1 = 800 B

File 2 = 40 B

File 1 - block 0, 1
File 2 - block 2

⇒ Problem is external fragmentation.

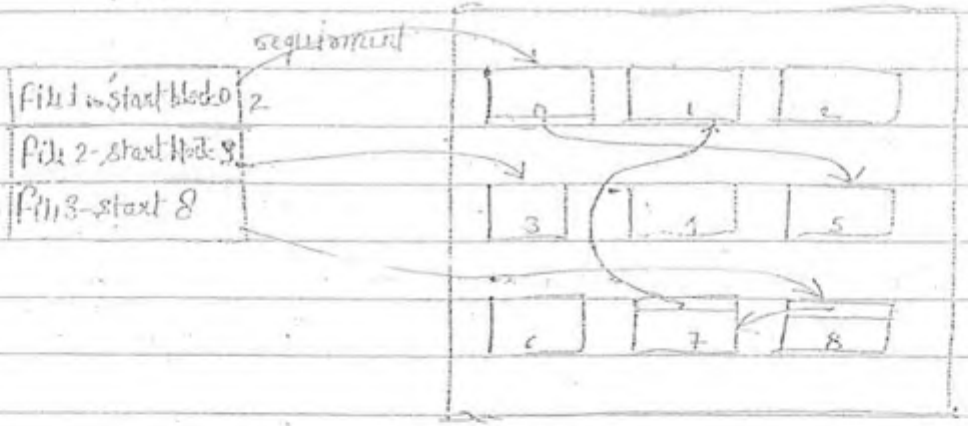
⇒ Benefit :- ① easy to implement
② read & write are fast

2. Linked Allocation :- (related to FAT file sys^m) -
Block can be allocated anywhere.

File 1 = 800 B

File 2 = 40 B

File 3 =



~~request arrives for file 3 block 1~~

1st movement for block 8 (read content, transfer pointer)

2nd movement for block 7

3rd movement for block 1.

- for reading one block sys^m has to read 3 blocks.

so which stored in RT so it will take more time.

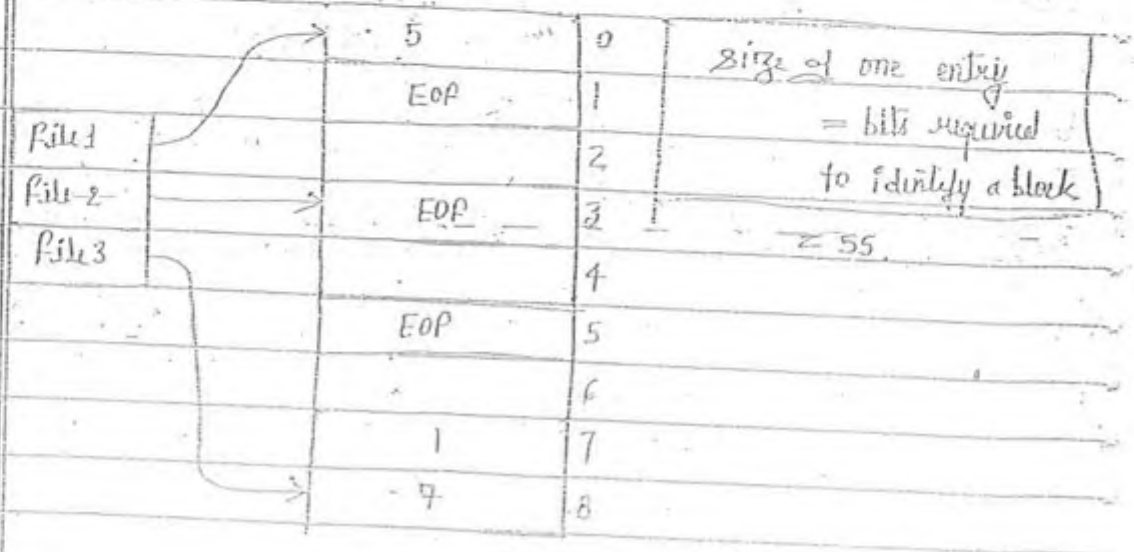
PAGE NO.
DATE :

⇒ FAT (File Allocation Table) is created in Linked Allocation.

FAT

Stored inside SM but when OS execute it brings FAT to Main Memory.

no. of entries in FAT = no. of blocks in SM



Bcoz FAT is in main memory so time taken to find contents of table is very less. than reading blocks in SM.

FAT SIZE = (no. of entries = no of block size in SM) X (size of one entry = no. of bits required to identify a block.)

FAT 16 means :-

size of one entry in FAT = 16 bit

FAT 32 means :-

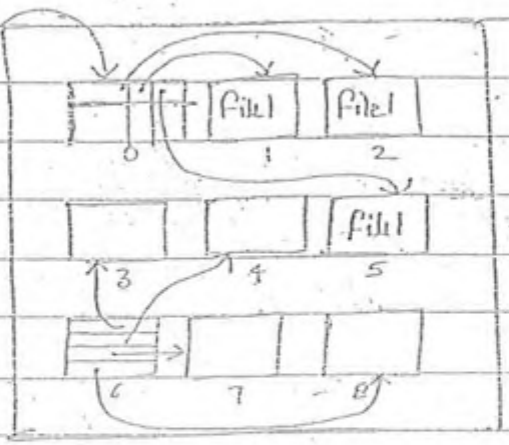
size of one entry in FAT = 32 bit

If FAT is corrupted then data can't be accessed.

PAGE NO.
DATE :

3. Indexed Allocation Table :-

	requirement
File1 - start 0	8 block
File2 - start 6	4 block



File 1 data is stored in 1, 2, 5 block.

File 2 data is stored in 3, 4, 7, 8 block, index block = 6

size of pointer (address) = 32 bit = 4 B

size of one block = 512 byte

max size of file supported by this indexed data structure = 512/4 B

= 128 no. of pointer

total size = (no. of block of pointer) × size of block

= 128 × 512 B

= $2^7 \times 2^9$ B = 2^{16} B

⇒ File greater than max. size can't stored.

PAGE NO.
DATE :

GATE 2008 - 1998

Gate 1998 :-

Que :- counting semaphore was initialized to 10

6

Que :- Consider five 8-tape drive with n processes. Each process may need 2 drive. What is the max value of n for which sys will be deadlock free.

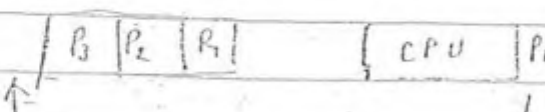
max need < m + n process = n

2n < 8 + n max need = 2n

n < 8

max value of n = (5).

Que :- n no. of processes sharing CPU in RR fashion. The context switch of each process takes s sec, what must be quantum size q such that overhead resulting from process switch is minimized but at the same time each process is guaranted to get its turn at the CPU at least every t sec :-



s sec (P1 context switch)
q (P2 execute)
s (P2 context switch)
q (P3 execute)
s (P3 context switch)

PAGE NO.
DATE :

total overhead - $ns + (n-1)q$ (max value)

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

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

$q \geq \frac{t-ns}{(n-1)}$

Ans :- Instruction takes i μsec and page fault take additional j μsec. The eff. Instruction time if only one a page fault occurs every k instruction.

Page fault = $1/k$

Page hit = $1 - 1/k$

Eff Access time = $(1-k)i + 1/k \times (i+j)$

$= i - i/k + i/k + j/k$

$= i + j/k$

Ans :- Partition size in KB 4K, 8K, 20K & 2K

Job size in KB. = 2K, 14K, 3K, 6K, 6K, 10K, 20K, 7K

Execution time = 4 10 2 1 4 1 8 6

Computer sys^m uses best fit algo for allocating memory space to this job. When will the 20K job complete.

20K	14K/6K/10K	Empty at $10+1 = 11$
8K	5K	Empty at 4
4K	3K	Empty at 2
2K	2K	Empty at 1

PAGE NO.
DATE :

Goth 2000 :-

Que :- Which of the following need not necessary saved on a context switch b/w processes.

(a) GPRs. (b) Translation look-aside buffer [✓]

(c) Prog Counter (d) All the above [✗]

Que :- Let m[0] ... m[4] mutexes.

P[0] ... P[4] processes.

wait(m[i])

wait(m[(i+1)%4])

—

—

release(m[i])

release(m[(i+1)%4])

deadlock [✓]

Que :- Suppose the time to service a page fault on the avg is 10 ms. while memory access takes 1 μsec. Then 99.99% hit ratio result in avg memory access ?.

Hit ratio = 99.99% = .9999

Avg Access Time = 0.9999 × 1 μs + (1 - .9999) × 10 ms

= 0.9999 × 10^{-6} sec + (1 - .9999) × 10 ms

= 0.9999 × 10^{-6} sec + (1 - .9999) × 10 × 10^{-3} sec

= 1.9999 μsec.

Que :- The soln of readex writer problem given, complete code. - using single binary semaphore. (Hint use

PAGE NO.
DATE :

Int R=0, W=0

Reader()
{
L1: wait (mutex)
if (W==0) {
R = R+1;
{ [ ] 1
else { [ ] 2
go to L1;
}
/* do read */
wait(mutex)
R=R-1
Signal (mutex)

Writer()
{
L2: wait (mutex)
if ([3]) {
Signal (mutex);
go to L2;
}
W = 1;
Signal (mutex);
/* do write */
wait (mutex);
W=0
Signal (mutex);

Gate

Que

Que

for

no.

A

(b)

(c)

(d)

Que

Que

Value

PAGE NO.
DATE :

1: signal(mutex)
2: signal(mutex)
3. R>0 or W>0.

Gate 2001 :-

Que :- Where does the swap space reside
= disk [✓] (secondary memory)

Que :- Virtual memory sys^n, with FIFO page replacement for an arbitary page access pattern increasing the no. of frames in MM will :-
(A) Always decrease no. of page fault
(b) Always increas no of page fault
(c) Sometimes increase the page fault [✓]
(d) never affect the no. of page fault

Que :- Which of the following doesnot interrupt a running process
(a) device (b) Timer
(c) Power failure (d) Scheduler process [✓]

Que :- mutual
flag [i] = true
turn = j
while (P) do no op ;
<CS>
flag [i] = false ;
<RS>
while false
Value of P ? value -

PAGE NO.
DATE :

flag[1] = true && turn = 1 [✓]

Que:- Consider the disk with following specifications-
20 - surfaces
1000 track / surface
16 sector / track
1 kB data density / sector
3000 rpm rotation speed

(a) total capacity of disk ?
(b) data transfer rate ?

Capacity = 20 × 1000 × 16 × 1 kB
= 320000 kB
= $2 \times 10^4 \times 2^4 \times 2^{10}$ B
= $2^{15} \times 5^4$ B

transfer rate :-
capacity of one track = 16 KB
rotational speed = 3000 rpm
1 min revolution = 3000
1 " " = $\frac{3000}{60} = 50$
1 rev takes = 1/50 sec
transfer rate = $\frac{16}{1/50}$
= 16 × 50 KB/sec.

Que :- Two concurrent processes P_1 & P_2 and two resources R_1 & R_2. Processes use the resources in mutual exclusion manner. Initially R_1 & R_2 are free.

P_1	P_2
S_1 while (R_1 is busy) do nop;	Q_1 while (R_2 is busy) do nop;
S_2 Set $R_1 \leftarrow$ busy;	Q_2 Set $R_2 \leftarrow$ busy
S_3 while (R_2 is busy) do nop;	Q_3 while (R_1 is busy) do nop;
S_4 set $R_2 \leftarrow$ busy;	Q_4 set $R_1 \leftarrow$ busy
S_5 use R_1 & R_2;	Q_5 use R_1 & R_2
S_6 set $R_1 \leftarrow$ free;	Q_6 set $R_2 \leftarrow$ free;
S_7 set $R_2 \leftarrow$ free;	Q_7 set $R_1 \leftarrow$ free;

(a) mutual exclusion guarded on R_1 & R_2

(b) can deadlock occur

Change Q_1 to Q_3 and Q_2 to Q_4 then –

mutual Exclusion is not guaranteed on R_1 & R_2 [V]

deadlock can't occur

Que: Disk with 100 tracks numbered from 0 to 99. Rotating with 3000 rpm. no. of sectors/track is 100. The time to move the head b/w two successive tracks.

Consider a set of disk request to read data from track. Algo is SCAN elevator & initially track no 25. moving up (toward larger track no.).

What is the total seek time for servicing the request.

50, 7, 45, 5 & 10

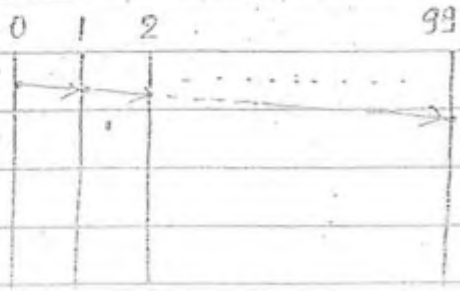
PAGE NO.
DATE :

$[(52-25)+(15-32)+(93-45)+(99-10)+(10-2)+(2-5)]$

$\times 0.2$ ms

$= 33.6$ ms.

[b] Consider the initial set of hundred arbitary disk request and assume that no new disk request occurs while servicing these request. If the head is initially track no. 0 & elevator algo is used to schedule disk request. what is the worst case time to complete all the request.



In worst case Head has to used whole track. So

seek time = $99 \times .2$ ms

rotational latency + block transfer time

= 100 [one complete revolution time]

rpm = 3000

60 $\frac{3000}{\text{sec}}$

1 complete revolution time = $1/50$ sec

rotational latency = $\frac{100 \times 1}{50}$

= 2 sec

19.8 ms + 2 sec = 19.8 + 2000 ms

= 2019.8 msec

PAGE NO.
DATE :

GATE 2002

Que :- following features will characterize an OS as a multi program OS.

(1) more than one prog can be loaded into main memory at same time for execution.

(2) Program wait for certain event such for i/o, another program immidiately scheduled for execution

(3) Execution of program terminate another program immidiately, schedule for execution.

(a) I (b) I & II [✓]

(c) both none (d) no one

Que :- Index allocation scheme of a block to a file, max size of file depend on

(a) size of block & size of address of block

(b) no. of block used for index & size of block

(c) size of block, no of blocks used for indexing & size of address of block. [✓]

(d) none of the above

Que :- Test And Set

```
int TestAndSet (int *x)
{
int y;
A1: y = *x;
A2: *x = 1;
A3: return y;
}
```

PAGE NO.
DATE :

for achieving mutual exclusion.

```
int mutex = 0
void enter_cs()
{
while ( 1 )
{
<CS>
void leave_cs()
{
2
}
```

1. TestAndSet (& mutex)
2. mutex = 0;

Ex :- Computer uses 32 bit virtual address & physical address. Physical memory is byte addressible. Page size = 4 KB. Two level page table for translation virtual address to physical address. Equal level of bits should be used for indexing 1st level & 2nd level of table. Size of each page table entry is 4 B.

[a] What is no. of page table entries that can contain in each page ?

page size = 4 KB
Address = 32 bit offset = 12 bit
remaining part will be divided in 2 part
20 bit for indexing in 1st level
10 bit " " " 2nd level.

entry size = 4 B

no. of entries in one page = page size / size of entry
= 4 KB / 4
= 1 K = 1024

PAGE NO.
DATE :

b] How many bits are available for storing protection & other info in each page table entry.

Size of physical memory = 2^{32} byte

frame size = size of page = 4 KB

$= 2^{12}$ B

No. of frames in MM = $\frac{2^{32} B}{2^{12} B}$

$= 2^{20}$ frames.

20 bits are used for address

So 32 − 20 = 12 bits used for protection & other info.

Que :-

```
# define BUFFSIZE 100
buffer buf[BUFFSIZE];
int first = last = 0
semaphore b_full = 0;
semaphore b_empty = BUFFSIZE
void producer ()
{
  while (1)
  { produce an item }
  f1 [ ]
  buf[first] = item;
  first = (first + 1) % BUFFSIZE
  f2 [ 1 ]
}
void consumer ()
{
  while (1)
```

PAGE NO.
DATE :

{
C_1 [ ]
take the item from buffer
buf [size]
last = (last + 1) % BUFFSIZE
C_2 [ ]
consume item
}
}

⇒ code for one producer & consumer

privileged instruction
└ executed

P_1: wait (b.empty) C_1 : wait (b.full)
P_2 signal (b.full) C_2 : signal (b. empty)

Another semaphore variable. Insert one stmt just after P_1.

mutex = 1

After P_1 : wait (mutex) After C_1 : wait (mutex)
Before P_2 : signal (mutex) Before C_2 : signal (mutex)

Gate 2003

Que 1 :- Ans (a)

Que 2 :- VAS = 32 bit
page size = 1 KB
Ans (c)

pages = $2^{32} / 2^{10} B = 2^{22} B$
page size will increase so main memory have large memory overhead.

PAGE NO.
DATE :

Gate 2001 :-

User prog are prevented to handling

I/O Instructions

I/O mapped (I/O has explicit instruction)

Memory Mapped. (All memory related instruction are used for I/O)

consists privileged instruction

executed only in kernel mode.

after

Gate 2009 :-

⇒ CPU checks the interrupt after completion of instruction and before executing next instruction

⇒ Belady anomaly occurs in FIFO.

⇒ Essential entry in page table - page frame no.

Que 43 :-

four type of resources - R1 R2 R3 & R4

to

Unit 3 2 3 2

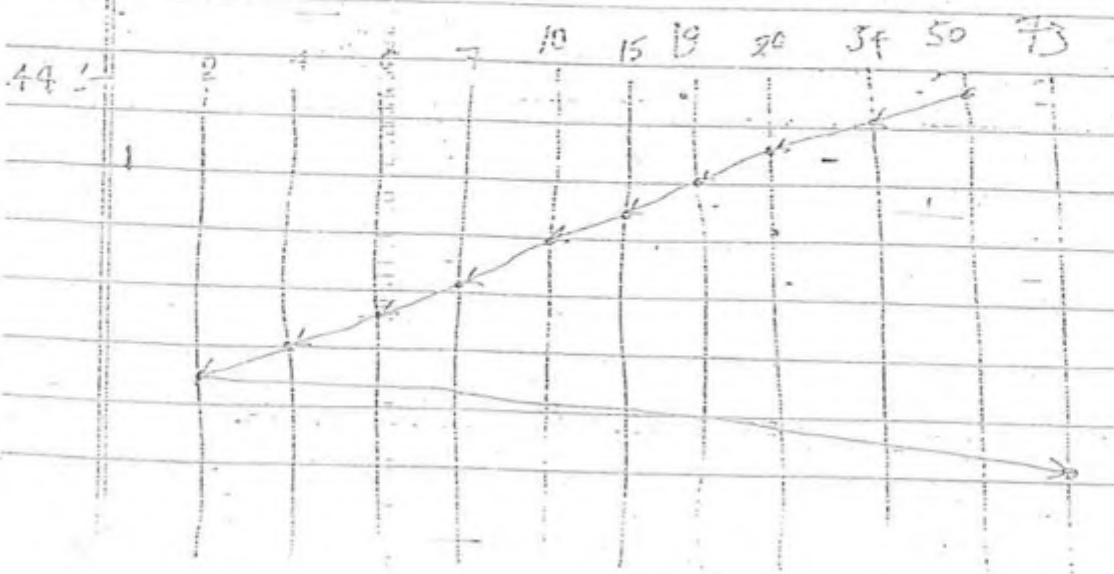
P1 P2 P3

t = 0 request

PAGE NO.
DATE :

	R_1	R_2	R_3	R_4
$t=0$	3	2	3	2
$t=0$	2	0	1	1
$t=1$	3	0	0	1
$t=2$				
$t=3$	1	0	0	0
$t=4$	0	3	0	0
$t=5$	0	0	0	0
$t=6$	0	0	1	0
$t=7$	1	0	1	0
$t=8$				
$t=09$				
$t=10$				

	P_1	P_2	P_3
$=0$	$R_2=2$	$R_3=2$	$R_4=1$
$=2$	$R_3=1$	$R_4=1$	$R_1=2$
$=3$	waiting (R_1)		
$=4$		$R_1=1$	
$=5$	$R_1=2$	$R_1=$	



PAGE NO.
DATE :

⇒ Different heads are always at same cylinder no.

(80-34) + (34-20) + (20-19) + (19-15) + (15-10) + (10-7)
+ (7-6) + (6-4) + (4-2) + (73-2)

= 15 + 14 + 1 + 4 + 5 + 3 + 1 + 2 + 2 + 71

= 78 + 71

= 119 ms

Ques 48:-

Note 2008

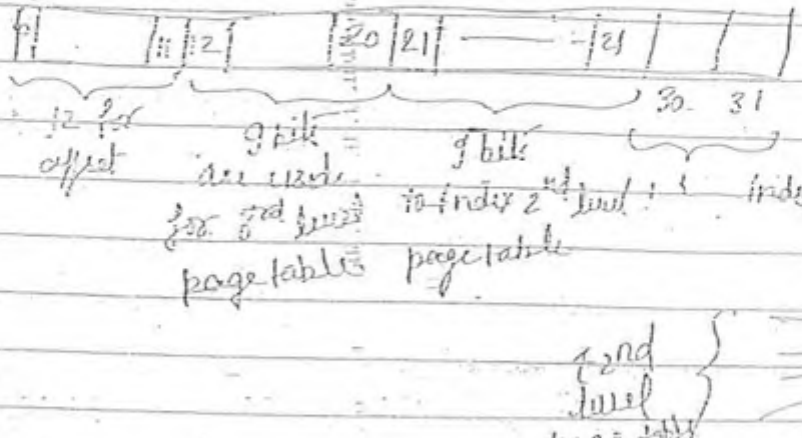
Ques 49:-

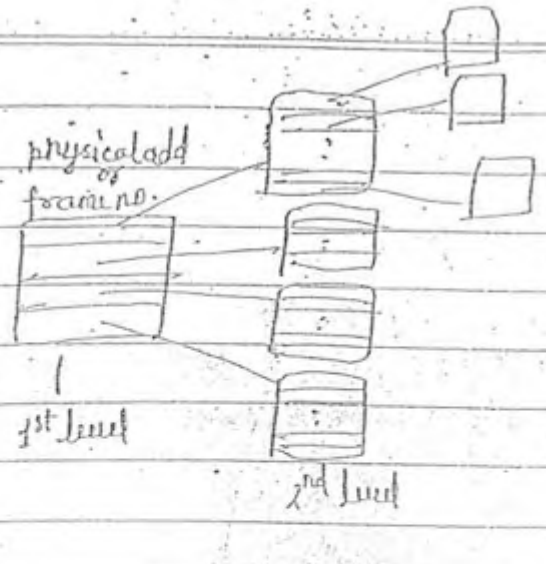
Physical address = 36 bit

VAS = 32 bit

Page frame size = 4 KB

Each page table entry size = 4 B





No. of frames in physical memory = $2^{36}B$ / $2^{12}B$ = 2^{24}

Every page table contain 2^{24} bit to point next level of page table

Que 87

Two type of I/O

Synchronous	Asynchronous
Process (sys call)	Process
Device Driver	Device Driver
Kernel	Kernel
H/w	H/w

open("file1"); // block.

⇒ In syn I/O sys call is blocked till I/O completes.

⇒ In asyn I/O sys I/O fn return some value w/o executing the code.

PAGE NO.
DATE :

gate 2007 (CO)

Ques :- $16 \times 128 \times 256 \times 512$

$2^4 \times 2^7 \times 2^8 \times 2^9$

$= 2^{28}$ B capacity of disk pack

$= 256$ MB

no. of sectors =

$= 2^4 \times 2^7 \times 2^8$

$= 2^{19}$

so 19 bits are required to identify sectors.

Lottery Scheduling :-

P_1 P_2 P_3 lottery

design no. to each process. OS generate a random no. and process which have that no. is served by CPU.

$P_1 - 50 - 49\%$

$P_2 - 5 - 10\%$

$P_3 - 2 - 40\%$

$P_4 - 3 - 30\%$

If dint request for sooner than it gives all its lotteries to server. Process can exchange their lotteries.

⇒ lottery is assigned according to CPU time for the process.

PAGE NO:
DATE :

Gate 2007 :-

Que 30 :-

	x	y	z
	S	S'	S'

After Allocation

	D	I	2
	2	1	3
	2	3	4

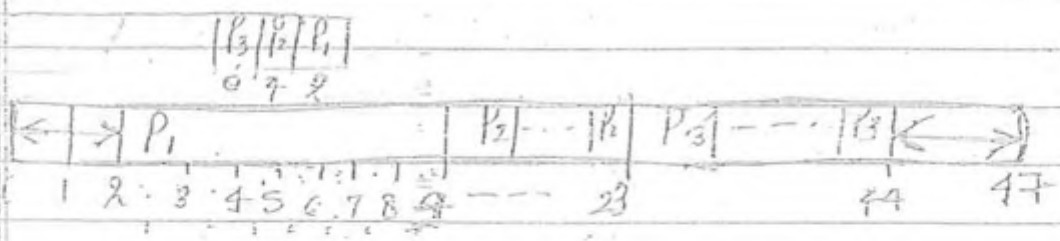
$\langle P_1, P_0, P_2 \rangle$

Que 31 :-

Gate 2005

Que 22 :-

	P_1	P_2	P_3
Arrival time	0	0	0
total Execution time	10	20	30
I/O	2	4	6
CPU	7	14	21
I/O	1	2	3

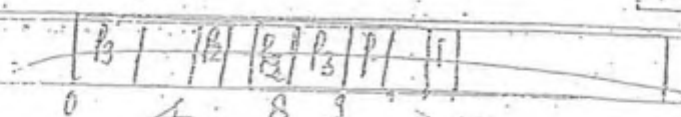


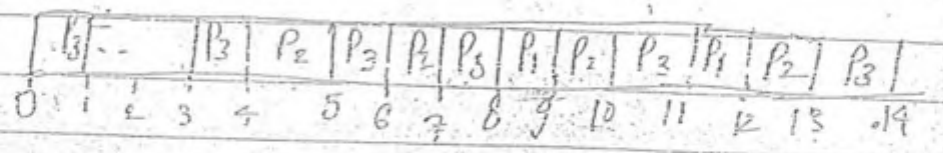
$= \frac{5}{47} \times 100 = 15.6\%$

Que 21 :-

	burst time	I/O	arrival time
P_1	2	0	0
P_2	4	1	0
P_3	8	2	0

PAGE NO.
DATE :





$P_1 = 12 - 0 = 12$

$P_2 = 13 - 0 = 13$

$P_3 = 14 - 0 = 14$

$39/3 = 13$

Code Quos -

Que 16 :-

u, v

x & y

a +10 = x

v ≠ y

31/2004

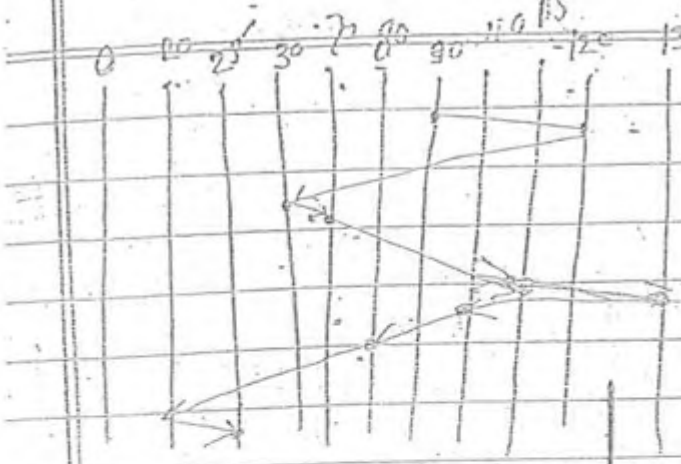
Que 62 :- 0 to 199. 20 tracks.

It was serving the request from track no. 120.

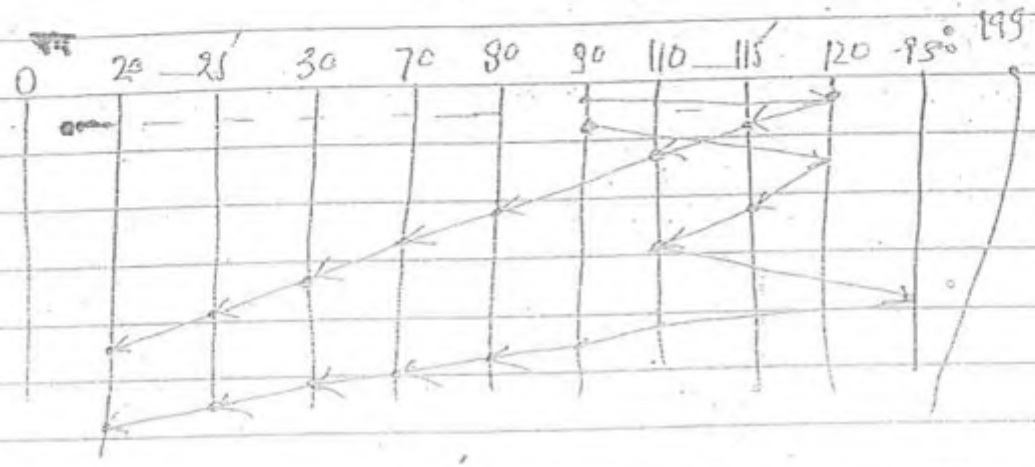
previous request = 90.

30, 70, 115, 110, 82, 20, 25.

(1) SSTF (2) FCFS



Head changes direction = 4 times



(5) Deadlock prevention Technique :-

Assign time stamp to each process. and is associated with same time stamp if it is killed. Let Pn be the process holding resour R. Pr be a process requesting for same resource.

T(Pn) — timestamp

T(Pr) — timestamp

decision of wait & kill is based on this algo —

if (T(Pr) < T(Pn)

kill Pr

else

Wait

PAGE NO.
DATE :

⇒ Here processes are killed so there is no deadlock but there is starvation.

Workbook —

Chapter — 7

Que 12 :— semaphore S = 0, S1 = 0

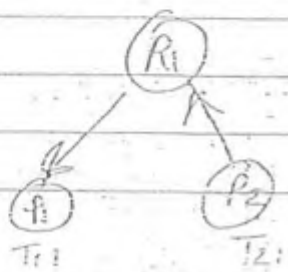
P1	P2
wait(S)	
use R1	use R1
	signal(S)
	wait(S1)
use R2	
signal(S1)	
wait(S)	
use R3	use R3
	signal(S)
	wait(S1)
use R4	use R4
signal(S1)	

Que — shared resource R_1, R_2

Process — P_1 & P_2

Each process has certain priority over resource.

Try denote the priority of P_i over R_i



① $T_{11} > T_{21}$ ④

② $T_{12} > T_{22}$

③ $T_{11} < T_{21}$

④ $T_{12} < T_{22}$

PAGE NO.
DATE :

Ques :-

Computer sys^{m} have 4 files —

(i) 11050 B
(ii) 4990 B
(iii) 5170 B
(iv) 12640 B

block size = 100 B & 200 B

for each block used to store the file required 4 bytes of book keeping info^{n}

total space used to store the file = file required space + book keeping info^{n}

Single block can't contain the file data & book keeping info^{n}.

find total space required to store the files using 100 B block & 200 B block.

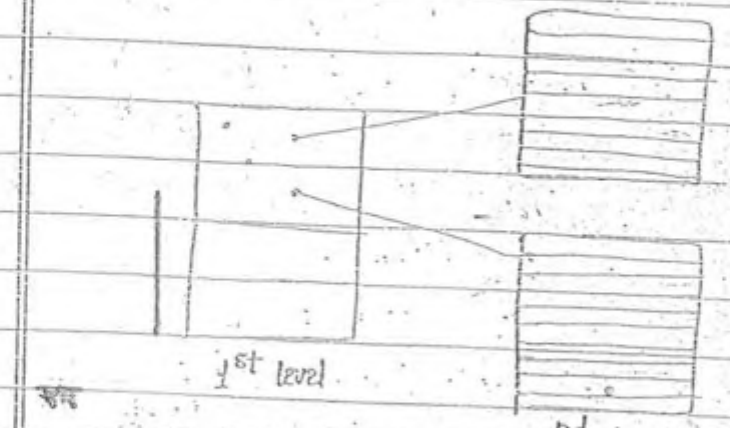
Ans - 35600 & 35400

12th Aug 09

PAGE NO.
DATE :

ALLOCATION of BLOCKS-TO FILES :-

Multilevel index allocation



Total no of pointer = (128) X (128)

= No. of second level × No. of pointer in one block

$= 2^7 \times 2^7 \Rightarrow 2^{14}$

Total size of data =

2^{14} X size of one block

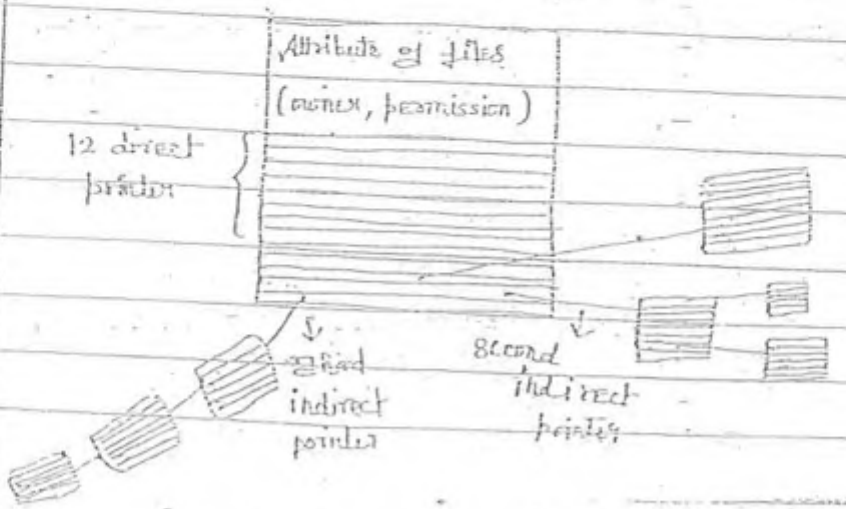
$= 2^{14} \times 2^9 \Rightarrow 2^{23}$ B

$= 2^3 \times 2^{20}$

= 8 MB

In unix concept PCB is allocated INODE

INODE :-



PAGE NO.
DATE :

Block size = 512B

pointer size = 32 bit = 4B

no. of pointer = $512/4 = 2^7$

Max size of data pointed by INODE :—

12 × (size of one block) +
128 × (size of one block) +
128 × 128 (size of one block) +
128 × 128 × 128 (size of one block)

$\Rightarrow (12 + 2^7 + 2^7 \times 2^7 + 2^7 \times 2^7 \times 2^7)$ size of one block

dominating

$\Rightarrow (2^7 \times 2^7 \times 2^7) \times 512B$

$\Rightarrow 2^{21} \times 2^9$

$\Rightarrow 2^{30}$

$\Rightarrow 1GB$

∴ max size of data pointed by INODE

Qu :- If block size = 1024 B
address of one block = 18 bit
data structure INODE (10 direct pointer + 1 single indirect,
1 double & 1 triple indirect)

Soln :- $(10 \times 2^{10}) + (2^9 \times 2^{10}) + (2^9 \times 2^9 \times 2^{10}) + (2^9 \times 2^9 \times 2^9 \times 2^{10})$

$= (2^9 \times 2^9 \times 2^9) \times 2^{10}$

$= 2^{27} \times 2^{10}$

$= 2^{37}$

$= 128\ GB$

no. of pointer = $2^{10}/2 = 2^9$

or

$\frac{1024}{2} = 2^9$

Qu :-
Each
addres
Also
Sol

INOD
bits
File

FREE

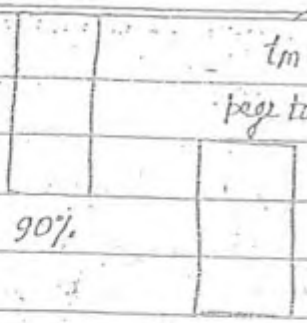
Each da

9

$t_t = 0$
TLB

Access time of memory

PAGE NO.
DATE :



Hit ratio = 1/20,000

1 page fault in 20000 instruction

2 memory reference so page fault = 20000

⇒ Address divided into two part page & offset. Page is searched into TLB, and then memory is searched.

Average effective time w/o page fault =

$$.9 \times (t_t + t_m) + (1-.9) \times (t_t + t_m + t_m)$$

$$= .9 \times (0 + 150) + (.1) \times (0 + 150 + 150)$$

$$= 90 + 30 = 135 + 30$$

~~$= 110$ nsec~~ $= 165$

Effective Access time with page fault =

(1 - page fault rate) (memory access time) + page fault rate × page fault time

page fault rate = $.5 \times 10^{-4}$

$$= (1 - .5 \times 10^{-4})(165) + .5 \times 10^{-4} \times 8 \text{ msec.}$$

$$= (1 - .5 \times 10^{-4})(165) + .5 \times 10^{-4} \times 8 \times 10^{6} \text{ ns}$$

$$= 165 + 4 \times 10^{2} \text{ ns}$$

$$= 565 \text{ ns.}$$

~~Actual effective time~~ $= 100 + 2 \times (565)$

$= 100 + 1130$ ns

$= 1230$ ns.

PAGE NO.
DATE :

Average = $h_1 \times (t_1 + t_m) + (1 - h_1)(t_2 + t_m + t_m)$

1 - pagefault x Average
+ pagefault x page service time

(2007, 8S)

Que 86 :-

Locality of reference

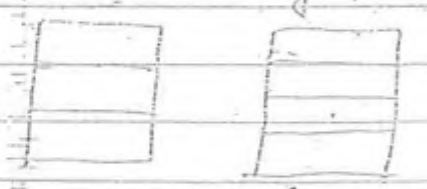
temporal locality of reference (related to time) | Spatial locality of reference (related to space)

If instruction i is referenced by CPU in time $t = t_1$ then this will again reference in near future is known temporal locality of reference.

If instruction i is referenced by CPU then its near by instructions will reference by CPU in near future. In Spatial locality of reference.

⇒ Belady's Anamoly occurs in of stack class algorithm.

Stack Class Algorithm :- System 1 has n no. of frame.
System 2 has (n+1) no. of frame.



frame = 3 frame = 4

reference string = 2, 1, 1, 3, 4, 1, 2, 0, 1

PAGE NO.
DATE :

- At any time if an algo satisfies this condⁿ

no. of page in sysᵐ 1 $\subseteq$ no. of pages in sysᵐ 2
(n frames) (n+1) frames

Then it comes under stack class algorithm.

⇒ FIFO doesn't satisfy this condⁿ. so Belady's Anamoly occurs.

Optimal Size of Page :-

Suppose avg page size = s'
page size = p
and size of each entry in page table = e
total overhead = space required to store page table + internal fragmentation

no. of page required = s/p
no. of entries in page table = s/p
size of one entry = e
total space = s/p × e

Total Overhead = $\frac{s}{p} \times e + \frac{p}{2}$ (suppose internal frag. is half page)

diff with respect to p (here p is variable)

$$= -\frac{se}{p^2} + \frac{1}{2}$$

$$\frac{se}{p^2} + \frac{1}{2} = 0$$

$$\frac{s^2}{p^2} = \frac{1}{2} \quad p^2 = 2s^2 \Rightarrow \boxed{p = \sqrt{2se}}$$

PAGE NO.
DATE :

- FILE SYSTEM

File Sysm is a ds maintain by os to control the files. when secondary memory is formatted by os.

HDD

low level formatting	high level formatting
(track sectors are created)	(file sysm is created & SM is divided into logical entity called blocks.)

⇒ Every block has no. and size of block is decided by os during formatting. (Some times user may decide size of blocks)

⇒ If size of SM = 2^{64} B

size of block = 512 B

= 2^9 B

no. of block in SM = $\frac{\text{size of SM}}{\text{Size of blocks}}$

$= \frac{2^{64}\ B}{2^9\ B}$

$= 2^{55}$ B

So every block no. represented by 55 bits.

File :- An abstraction provided by os so that user can store their data w/o knowing the internal structure of SM.

File 1 - 56KB Block size = 24 kB

PAGE NO.
DATE :

How OS allocate blocks to data :—

⇒ In SM internal fragmentation always be and when OS allocate blocks in contiguous manner then external fragmentation is also possible.

Directory Structure is data structure that contain info of file.

File-Info :— general attributes (file info depend on file sys^m)

1. File name
2. Size of file
3. Owner of file
4. Physical loc in disk where file is store
5. protection info (read, write, execute)
6. Date of creation, date of last modified.

⇒ File info depend on file sys^m.

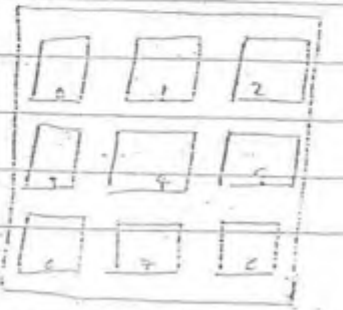
⇒ Directory Structure should not be greater bcoz it is stored in MM. So some critical info is stored in directory structure and other are stored in FCB. (File Control Block)

⇒ Every file has its own FCB.

⇒ Every partition has its own directory structure.

BLOCKS ALLOCATION TO FILE :— There are no. of technique the conflict are is contiguous allocation of block.

1. Contiguous Block Allocation :—



Size of block = 1

PAGE NO.
DATE :

File 1 = 800B

File 2 = 40B

File 1 - block 0, 1
File 2 - block 2

⇒ Problem is external fragmentation.

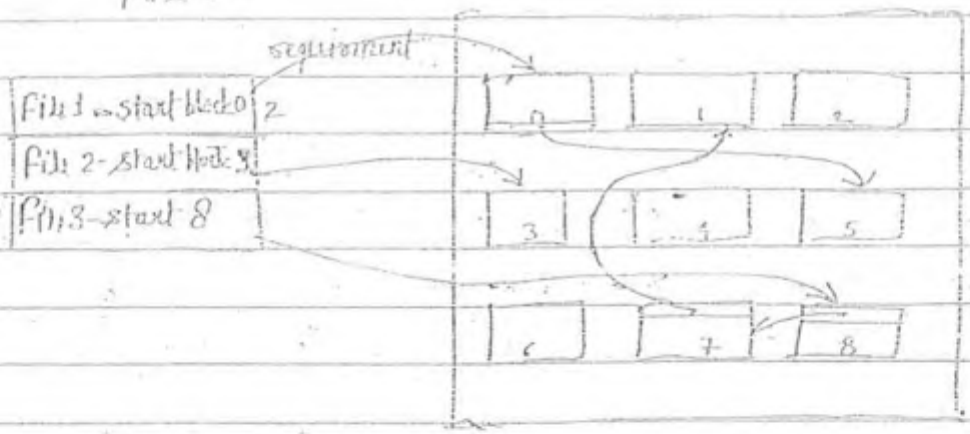
⇒ Benefit :- ① easy to implement
② read & write are fast

2. Linked Allocation :- (related to FAT file sys^{m}) -
Block can be allocated anywhere.

File 1 = 800B

File 2 = 40B

File 3 =



~~request arrive for file 3 block 1~~

1st movement for block 8 (read content, transfer pointer)

2nd movement for block 7

3rd movement for block 1.

For reading one block sys^{m} has to read 3 blocks.

So which stored in 81 so it will take more time.

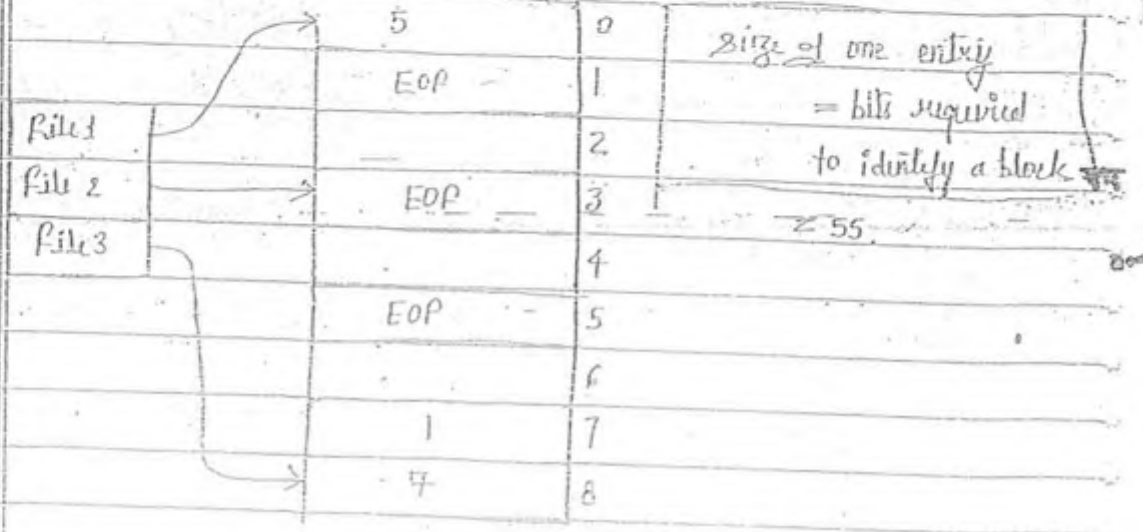
PAGE NO.
DATE :

⇒ FAT (File Allocation Table) is created in Linked Allocation.

FAT

Stored inside SM but when OS execute it brings FAT to Main Memory.

no. of entries in FAT = no. of blocks in SM



Being FAT is in main memory so time taken to find contents of table is very less. than reading blocks in SM.

FAT SIZE = (no. of entries = no of block size in SM) X
(size of one entry = no. of bits required to identify a block.)

FAT 16 means :-
size of one entry in FAT = 16 bits

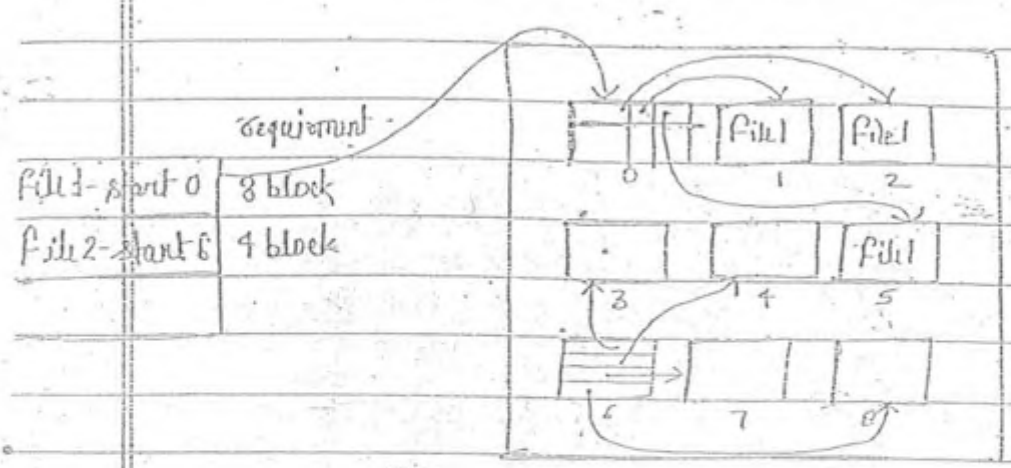
FAT 32 means :-
size of one entry in FAT = 32 bits

pointer

If FAT is corrupted then data can't be accessed.

PAGE NO.
DATE :

3. Indexed Allocation Table :-



file 1 data is stored in 1, 2, 5 block.

file 2 data is stored in 3, 4, 7, 8 block, index block = 6

size of pointer (address) = 32 bit = 4 B

size of one block = 512 byte

max size of file supported by this indexed data structure = $512/4$ B.

= 128 no. of pointer

total data = 128 (no. of block of pointer) × size of block

= 128 × 512 B

= $2^7 \times 2^9$ B = 2^{16} B

⇒ File greater than max size can't stored.

PAGE NO.
DATE :

GATE 2008 - 1998

Gate 1998 :—

Qu :- counting semaphore was initialized to 10

Qu :- Computer have 5- tape drive with n processes. Each process may need 2 drive. What is the max value of n for which sys will be deadlock free.

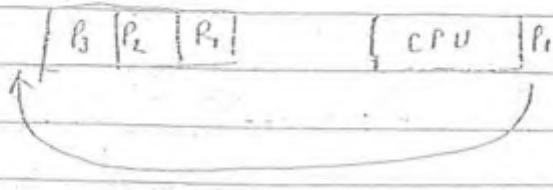
max need < m + n process = n

2n < 5 + n max need = 2n.

n < 5

max value of n = (5).

Qu :- n no. of processes sharing CPU in RR fashion. The context switch of each process takes s sec, what must be quantum size q such that overhead resulting from process switch is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t sec.



s sec (P1 context switch)
q (P2 execute)
s (P2 context switch)
q (P3 execute)
s (P3 context switch)

PAGE NO.
DATE :

total overhead - $ns + (n-1)q$ (max value)

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

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

$q \geq \frac{t - ns}{(n-1)}$

Ans :- Instruction takes 1 μsec and page fault take additional 1 μsec. The eff. Instruction time if only one a page fault occurs every k instruction.

Page fault = $1/k$

Page hit = $1 - 1/k$

Eff Access time = $(1-k)i + 1/k \times (i+1)$

$= i - 1/k \, i + 1/k + 1/k$

$= i + 1/k$

Ans :- Partition size in kB 4K, 8K, 20K & 2K

Job size in kB - 2K, 14K, 3K, 6K, 6K, 10K, 20K, 2K

Execution time = 4 10 2 1 4 1 8 6

Computer sys^m uses best fit algo for allocating memory space to this job. When will the 20K job complete.

20K	14K/6K/10K	Empty at $10 + 1 = 11$
8K	6K	Empty at 4
4K	3K	Empty at 2
2K	2K	Empty at 4

PAGE NO.
DATE :

Allocation time = 11
time required = 8
19

Que :- free disk space can be kept track of using a free list or bit map. Disk addresses required d bit. For a disk with b blocks, f of which are free. State the condⁿ under which the free list uses less space than bitmap.

bit map
Size = B bits
Free list
free blocks = f every free block keep pointer of next free block.
Address of one block = d bits
Size of data structure = f×d bits
f×d bits < B bits

Gate 1999 :-

Que :- Sysm calls are usually invoked by :
(a) Sw interrupt [✓] (b) Polling
(c) Indirect jump (d) Privilledge Instruction [ ]

Polling - checking status of register continuously.

Que :- Advantage of virtual memory
(a) faster access of memory in an average [X]
(b) Process can be given protected [X]
(c) Linker can assign add. Independent of Program where it is loaded
(d) Program larger than size of physical memory can run [✓]

PAGE NO.
DATE :

Qu :- Producer -

```
sujeet
    produce item;
    if count == 1 then sleep
    Place item in buffer
    count == 1
    wakeup (consumer);
forever.
```

Consumer

```
sujeet
    if count == 0 then sleep
    Remove item from buffer
    count == 0
    wakeup (producer).
forever consume item;
```

using buffer size = 1 & Assume initial value of count = 0
Assume that testing & assignment are atomic.
Is it possible that both process will go in sleep mode at same time.

Not possible [✓] because count is a single variable so either it will be 0 or 1. So only one process sleep at a time.

PAGE NO.
DATE :

$1\ GB = 2^{30}\ B$

$1\ MB = 2^{20}\ B$

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

Que :- In particular UNIX sys^m each data block of size 1024 byte. Each inode has (to direct data block address) and ③ additional address (1 for single indirect block, 1 for double & 1 for triple) Also each block contain address for 128 blocks.

Sol^n :- Block size = $1024 = 2^{10}\ B$,

128 address = 2^7

No. of pointers = 2^7

$(2^7 \times 2^7 \times 2^7) \times 2^{10}$

$= 2^{31}$

$= 2\ GB$

INODE in Unix (it is a data structure)

windows ⇒ FAT

File system ⇒ ext-2 } in linux
ext-2 }

FREE SPACE MANAGEMENT :- OS manages the free space.

Disk data structure :-

① Bit Vector (Field)

| no. of bits = no. of blocks in sys^m (SM) |

| 1 is bit for free space |
| 0 is for occupied blocks |

If 2^{64} blocks is in the system
then 2^{64} bits required to implement the bit vector table

PAGE NO.
DATE :

blocks 0 block 1 block 2 block 3 ---- block n

1 0 0 1

1 2 3 4	5 6 7 8	9 10 11 12
0000	0000	0010	0111	1101

word

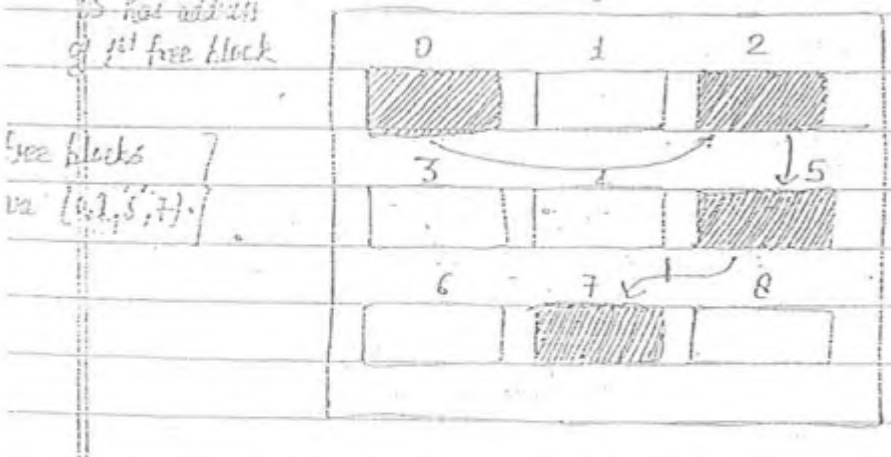
(0000) → size of word → 4 bits
and all the bits are zero
then it called zero word.

⇒ CPU can read 1 word at a time.

Position of free Space :- (size of word) × (no of zero word) + offset of first one in the non zero word.

= 4 × 2 + 3
= 11 (block no 11 is free)

2. Linked List :-



using PAT we can improve the performance of the linked list.

0	2
1	
2	5
3	
4	
5	7
6	
7	EOF

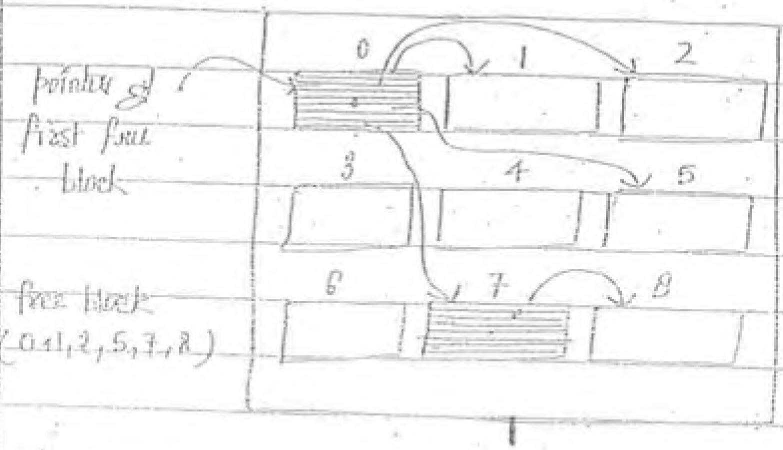
see from previous page

→ FAT is always in the MM while for the linked list it will search on the SM.

FAT: file Allocation Table.

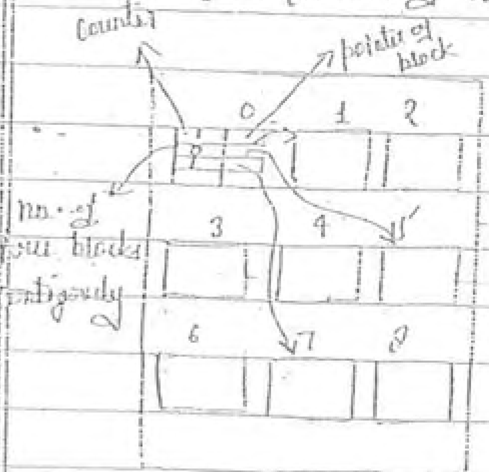
So time is very much reduced here.

Group :-



☆ If first free blocks have n no of pointer then last free block does not contain the data. It contain the pointer of next (n) free blocks.

Count (Extension of Group Technique)



Position or Minidisk or
↓
in RAM

⇒ more efficient than group in term of space.

PAGE NO.
DATE :

Convoy effect :- One long CPU bound process and many other CPU bound process are waiting.

Implement of Directory Structure :- One of the OS in the file system —

① Linear List
② Hash table

1. Linear List :-

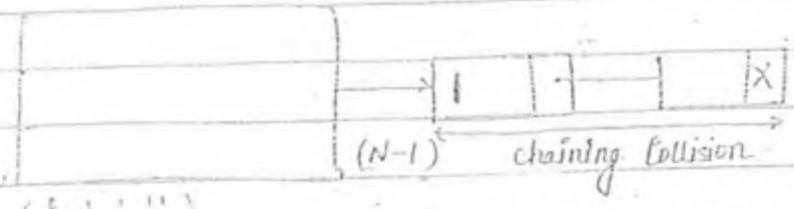
file 1 attribute	→ FCB
file 2 attribute	→ FCB
file 3 attribute	

2. Hash Table :-

hash function ⟺ file name % N

↓

total no of entries



⇒ Collision is handled by the linear chaing implementation.

File Attribute :-

operation on file :-

① Create
② Delete
③ Open
④ Rw
⑤ Reposition (seek)
a b c d
↑ ↑
⑥ Append
⑦ Concat

all operation can do with the sys. call.

PAGE NO.
DATE :

Athvi

⇒ If compiler provides called H.L. sys^m call.

⇒ If os provides low level sys^m call.

Ja file

Access of file :-

(i) Sequential Access

(ii) Random Access.

Protection of file :-

⇒

r	w	e
0	1	1

i.e. writable, executable but not readable

⇒ In unix file protection

rwe rwe rwe

(owner) (group) (universal)

Eg :- Protection file 1

111 10 110

7 4 6

(owner) (group) (universal)

file name	746	protection parameter

(Command in unix file protection)

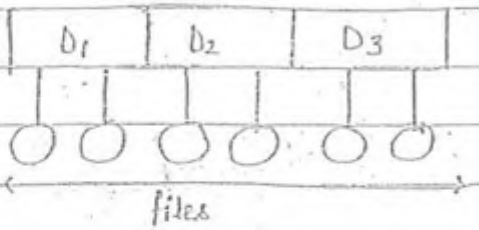
Directory Structure Organization :-

do

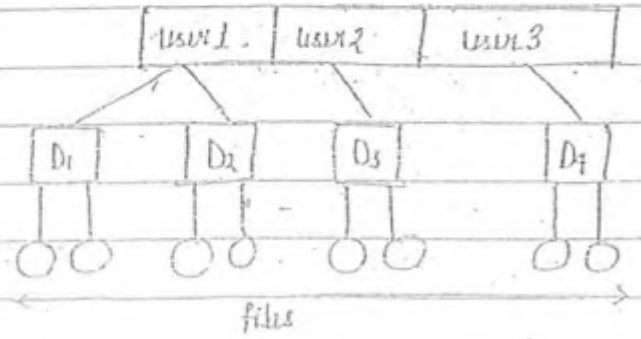
Directory Structure	Partition A
Directory Structure	Partition B
Directory Structure	Partition C.

PAGE NO.
DATE :

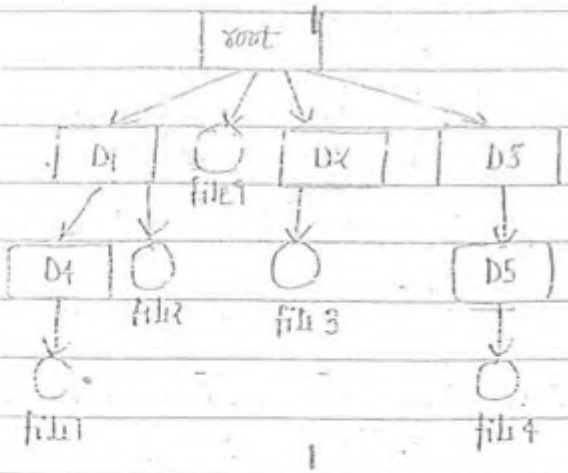
1. Single Level Directory Structure :-



2. Second Level Directory Structure :-



3. Tree Directory Structure :-



⇒ best support the concept of sharing.

⇒ Every file have path. (related to some other file)

Path ⇒ relative path or absolute

File 1 :- (root/D1/D4/file1)

File 2 :- (root/D1/D5/file5)

PAGE NO.
DATE :

file 1 ⇒ relative path :-

If D2 is current directory ← D2/root/D1/D4/file 1

⇒ Command prompt have (default) (directory) when opened.

⇒ In UNIX / LINUX ; current directory is represented by ⊙

Eg - (D2)

current directory

../root/D1/D4/file 1

D2 to root

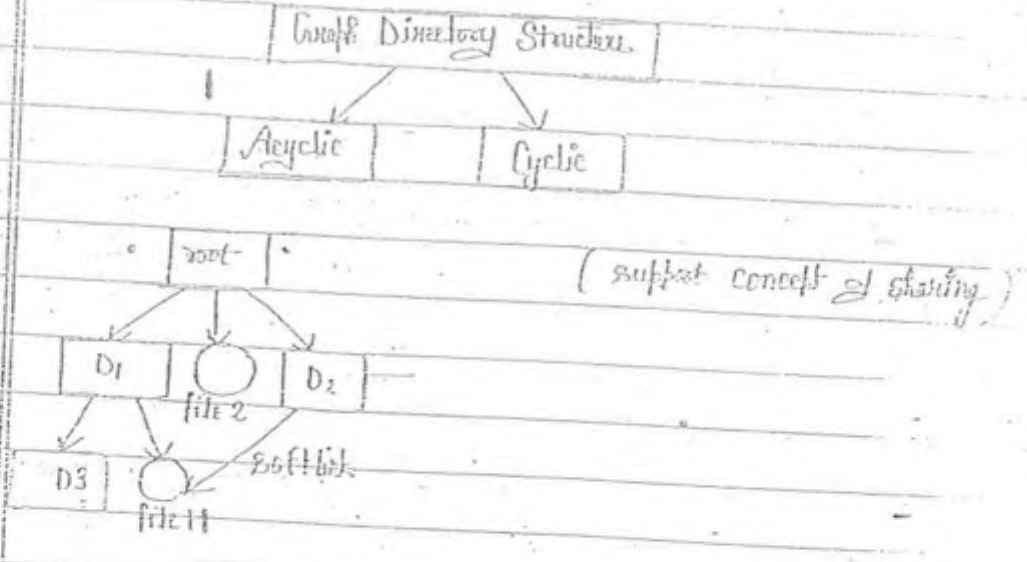
Imp ⊙ → One directory above the D2 directory.

Eg :- If D5 is current directory

then | ../root/D1/D4/file 1 |

D5 to D3 to root

⇒ (CD..) for windows



1) ⇒ Soft link :-

| WINDOW | ⟶ Shortcut

| UNIX | ⟶ Soft link.

PAGE NO.
DATE :

⇒ Soft link is created by user by the help of (sys^{m} call)

⇒ [file1] comes under only directory D_1
but have path from (D1) & (D2) both.

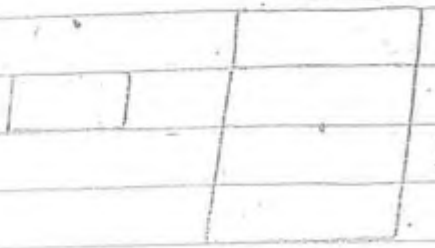
⇒ If file1 is deleted then softlink becomes (Dangling pointer)
and then OS will delete that dangling pointers from the
directory structure.

Efficient

ii) Harddisk ⇒ (Use of FCB or EMODE)

for file.

count=2 keep tracks how many path are possible

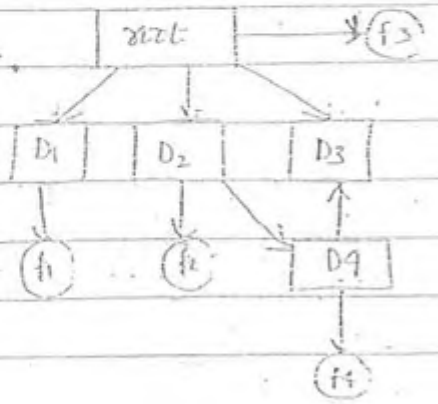


FCB or INODE { Easy for OS as if file1 is deleted then count = 0

⇒ Count technique called as Harddisk as no search for (dangling reference) here.

☞ If we delete a file then only data structure is deleted and contents of file is not deleted as blocks containing that data will considered as free frame, on new data is (Gets write with it).

(3) Cyclic Structure :-



PAGE NO.
DATE :

Book Concept

⇒ A source file is a sequence subroutines and function.
⇒ A object file is a sequence of byte organised into blocks understandable by the linker.
⇒ A executable file is a series of code sections that the loader can bring into the memory & execute.

Executable	exe, com, bin	Read to run M.L program
Object	obj o	Compiled, M/c not linked
Batch	bat, sh	commands to command interpretor

O.S IRS