Operating System

Syllabus

→ OS Concepts
→ goals, services, types of OS → 1m
→ System calls → 1m
① process management
→ PCB
→ process states → 1m (scheduler, dispatcher)
→ threads → 1m
→ CPU scheduling Algorithms
→ FCFS, SJF, Round Robin, Priority
→ multilevel scheduling, multilevel feedback scheduling → 3m
→ Interprocess communication
→ process synchronization
→ semaphores
→ deadlocks → 1m
② memory management
→ overlays
→ swapping
→ paging
→ segmentation
→ virtual memory → 1m
→ Demand paging
→ thrashing
→ Page Replacement Algorithm → 1m (FIFO, LRU, optimal)
→ Disk scheduling
→ FIFO, SSTF, SCAN, C-SCAN, Look, Clook → 1m
③ file management
→ File operations
→ File access methods
→ directory structure → 1m

operating system: An OS is a system program which acts as an intermediary b/w user & computer

→ An OS acts as an "resource manager"

↳ Here the resources are i) Processor
ii) memory
iii) I/O.

Ex:
1) windows (GUI based OS)
2) GNU/Linux (PC, work stations, ISP, File & print services etc) → Internet service provider
3) Mac OS (Apple's PC, tablets, work station)

Mobile OS: 1) Android (Java)
2) iOS (Apple)
3) Windows
4) Blackberry
5) OSX, VMS ... etc.

objectives of an OS:

1) convenience: OS makes a computer more convenient to use
2) efficiency: OS allows to use resources efficiently
3) Ability to solve problems.

Functions/services of OS

1) Boots (turning on) the computer
2) provides user interface (GUI, CLI (Command Line Interface))
→ 32 bit OS → It execute 2^{32} bytes of instructions at a time
i.e $2^2 \times 2^{30} \rightarrow 4GB$.
→ 64 bit OS → It executes 2^{64} i.e., $2^4 \times 2^{60} = 16EB$ of instructions at a time
3) Process management
4) memory management
5) Device (I/O) management
6) File mgm
7) Error handling etc.

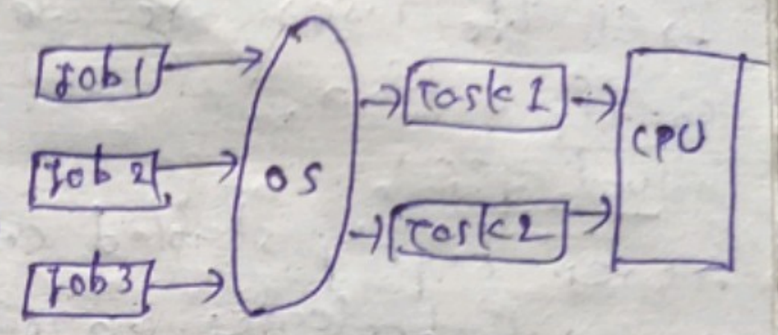
→ RAM stores the current going instructions
→ ROM stores the boot files that helps the computer to load OS into it.
→ platform ⇒ OS + processor

Types of OS

1) **Batch OS**: No direct interaction b/w user & computer
→ there is an operator which takes similar jobs having same requirements & group them into batches
→ it is the responsibility of the operators to sort the jobs with similar needs

Advantages

1) Multi users can use the batch system
2) Idle time is less
3) Easily manage large work
4) Processor knows the time required to complete a task



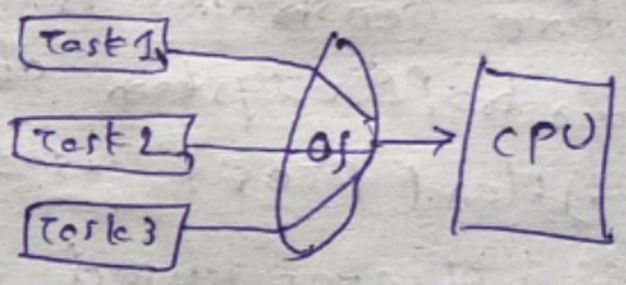
Disadvantages

→ computer operator should be well known about system
→ Hard to debug
→ It is costly sometimes
→ If any job fails, wait for unknown time to complete task.

2) **Time sharing OS**:
→ Each task gets time to execute

Advantages

1) CPU idle time is less
2) Every task get opportunity to execute

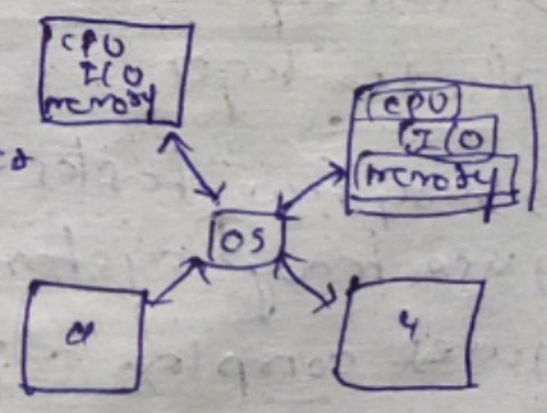


Disadvantages

1) Security & integrity, Reliability problems
2) Data communication problems
ex: multics, UNIX etc

3) **Distributed OS (Loosely Coupled systems)**:
→ various autonomous interconnected computers communi-cate each other using a shared communication n/w.
→ Remote access is possible.

Advantages



→ Failure of one system will not affect other
→ load on host computer reduced
→ scalable systems (can be extend)
→ Data processing delay is less
→ computation is fast bcz resources are shareable.

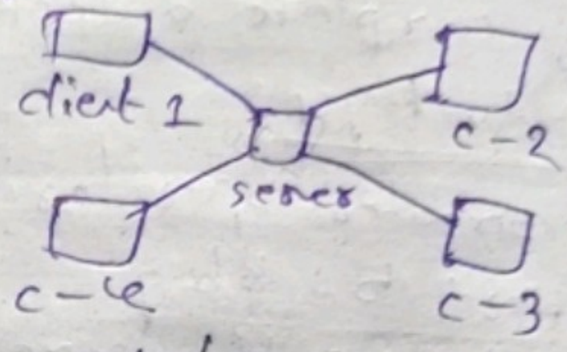
Disadvantages

→ Expensive
→ Failure of one main system n/w will affect entire system
ex: LOCUS

4) Network os (Tightly coupled Systems)
→ these systems runs on a server
→ provides the capability to manage data, users, groups
r/w functions etc.
→ Allows shared access of files, printers, applications etc
→ All the users are well-aware about underlying
configuration.

Advantages:
→ uses highly stabled centralised servers
→ N/w technologies, upgradation are easily integrated to
the system.
→ Server access is possible remotely

Disadvantages
→ servers are costly
→ users have to depend on centered server m/c's
→ maintance, updation are required regularly
Ex: Linux, MS windows server 2003/2008 etc



5) Real time os (RTOS): these systems were used where
time constraint is very strict like missile system,
air traffic system etc.
i) Hard RTOS: used where time constraint is full strict
ii) soft RTOS: used where time constraint is less strict

Advantages
→ Task shifting time is very less
→ maximum utilization of resources to get more o/p
→ Focusing on running applications
→ Error free
→ Memory allocation

Disadvantages
→ very few tasks run at same time
→ use heavy system resources
→ uses complex algorithms - difficult to write

Ex: 1) Scientific experiments 6) air traffic control system
2) Robots
3) Medical imaging systems
4) industrial control system
5) weapon system

s, groups

ications etc

ing

fed to

c-2

c-3

ontext

tem,

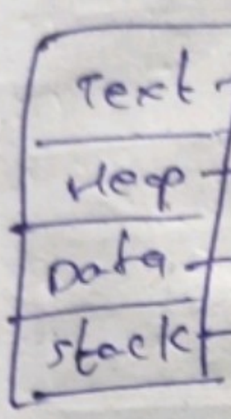
ll. strict

trict

Process management:

Process : Process is a pgm under execution

→ A single person can create any no. of processes
→ Executing the same pgm for multiple times will create different process every time



- text → current pgm
- Heap → allocates memory dynamically
- data → global variable
- stack → temporary data (function parameters) (local variables etc)

Attributes/ characteristics of a process

1) process id
2) process state (new/ ready state - etc)
3) program counter → stores the address
4) Priority
5) Register
6) I/O info. → devices allocated
7) Accounting info. → CPU time limits etc
8) pointer → ready outer switch to other process

→ Context of the process

Process state transition Diagram

(LTS) → long term scheduler

(STS) → short term scheduler

scheduler/ Dispatcher

→ the dispatcher will load the processes in running state which are scheduled from ready to running state

CPU scheduling :- scheduling & processes to finish task on time from Ready state to Run state.

→ these are diff. times with respect to a process.

i) Arrival Time (AT) : the time at which the process arrived into the ready queue

ii) Burst Time (BT) : the time taken to execute a process

iii) Completion Time (CT) : the time at which the process completes its executions.

iv) TurnAround Time (TAT) : TAT = CT - AT

v) Waiting Time (WT) : WT = TAT - BT

→ Types of CPU scheduling Algorithms

i) Preemptive CPU scheduling : stop executing the process & allocate new process forcefully to Runstate.

Ex : FCFS, SJF, LTF, Priority.

ii) Non-preemptive CPU scheduling : once the process allocates the Runstate

→ it never comeback till completion of its execution

Ex : RR, SRTF, LRTF, Priority.

1) FCFS (First come first serve) :

Criteria = AT Mode = Non-preemptive

Ex :

PNo	AT	BT	CT	TAT CT-AT	WT TAT-BT
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

19/5 Avg WT.

Gantt chart

P1 P2 P3 P4 P3 P4 P4 P5

P1	P2	P3	P4	P5

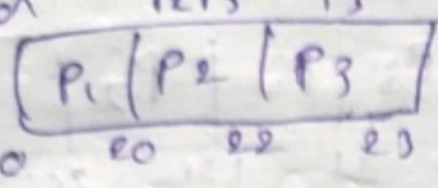
0 4 7 8 10 15

Ex-2 :

PNO	AT	BT	CT
P_1	0	20	20
P_2	1	2	22
P_3	1	1	23

convoy effect / starvation

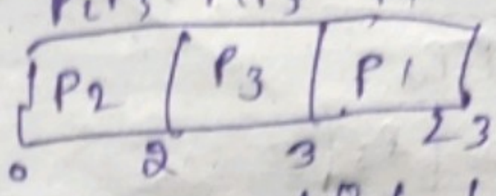
P_1 P_2P_3 P_3



Ex-3 :

PNO	AT	BT	CT
P_1	1	20	23
P_2	0	2	3
P_3	0	1	2

P_2P_3 P_1P_3 P_1

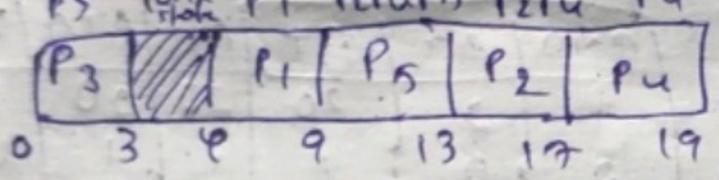


convoy effect / starvation :- If a process has high burst time the remaining process has to wait for longest time.

Ex-4 :

PNO	AT	BT	CT	TAT	WT
1	4	5	9	5	0
2	6	3	17	11	7
3	0	3	3	3	0
4	6	2	19	13	11
5	5	6	13	8	6

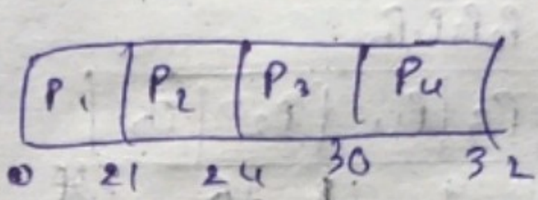
P_3 idle P_1 P_1P_5 P_2P_4 P_4



Ex-5 :

PNO	BT	CT
1	21	21
2	3	24
3	6	30
4	2	32

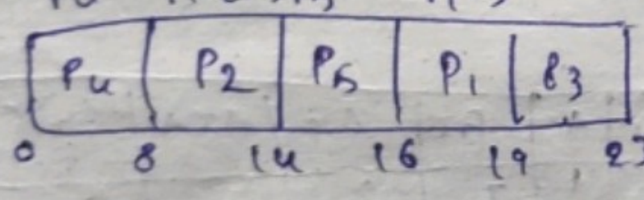
AT is not given so, we assume AT = 0 all are arrived



Ex-6 :

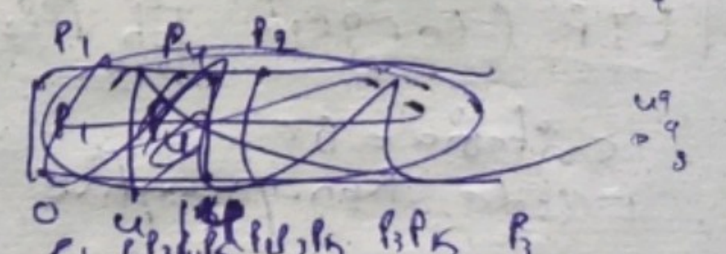
PNO	AT	BT	CT	TAT	WT
1	2	3	19	17	14
2	1	6	14	13	7
3	2	4	23	21	17
4	0	8	8	8	0
5	1	2	16	15	13

P_4 $P_1P_2P_3P_5$ P_1P_3



Ex-7 :

PNO	AT	BT	TAT
1	0	14	11
2	5	28	49
3	12	2	67
4	2	10	21
5	9	16	65



2) SJF (Shortest Job First)

Criteria = BT (shortest BT)

Mode = Non-preemptive

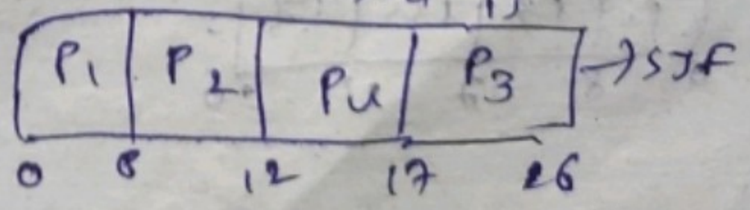
Ex (ECET-2023

PNO	AT	BT	CT	TAT	WT
1	0	8	8	8	0
2	1	4	12	11	7
3	2	9	26	24	15
4	3	5	17	14	9

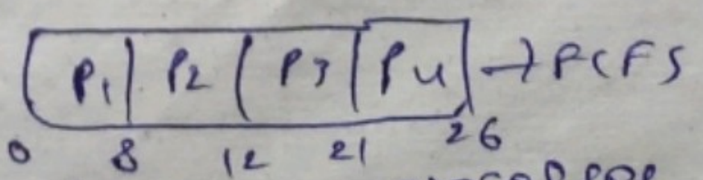
Avg WT = ?

3/4 ⇒

P_1 $P_2P_3P_4$ P_3P_4 P_3



→ SJF



→ FCFS

SJF → 17 → 3 processes

FCFS → 21 → 3 processes

Ex-2 :

PNO	AT	BT	CT	ATA	WT
1	0	12	12	12	0
2	2	4	16	14	10
3	3	6	27	24	18
4	8	5	21	13	8

P1 P2P3P4 P3P4

P1	P2	P4	P3

0 12 16 21 27

Ex-3

PNO	AT	BT	CT	ATA	WT
1	0	10	10	10	0
2	3	6	20	17	11
3	7	1	11	4	3
4	8	3	14	6	3

P1 P2P3P4 P2P4

P1	P3	P4	P2

0 10 11 14 20

Ex-4

PNO	AT	BT	CT	ATA	WT
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	6	27	19	13

P1 P2P3P4P5

C	P1	P3	P4	P2	P5

0 1 8 9 11 16 23

Ex-5

PNO	AT	BT	CT	ATA	WT
1	0	20	20	20	0
2	6	1	21	20	19
3	2	1	22	20	19

P1 P2 P3

P1	P2	P3

0 20 21 22

Ex-6 :

PNO	AT	BT	CT	ATA	WT
1	2	20	22	20	0
2	0	1	1	1	0
3	1	1	2	1	0

P2 P3 P1

P2	P3	P1

0 1 2 22

SJF → always give best throughput among all algorithms

↓

no. of processes executed per time unit is more

3| LJF (Longest Job First)

Criteria = BT
Mode = Non-preemptive

Ex-1 :

PNO	AT	BT	CT	ATA	WT
1	0	8	8	8	0
2	1	6	26	25	21
3	2	9	17	15	6
4	3	5	22	19	14

P1

P1	P3	P4	P2

0 8 17 22 26

Ex-2 :

PNO	AT	BT	CT	ATA	WT
1	1	7	8	7	0
2	2	5	21	19	14
3	3	1	24	21	20
4	6	2	23	19	17
5	5	8	16	11	3

C	P1	P5	P2	P4	P3

0 1 8 16 21 23 24

ATA	WT
10	0
17	11
4	3
6	3

0

ATA	WT
20	0
20	19
20	19

ATA	WT
20	0
1	0
1	0

WT
0
10
20
17
3

24

Ex-3 :

PNO	AT	BT	CT	ATA	WT
1	0	3	3	3	0
2	1	2	20	19	17
3	2	4	18	16	12
4	3	5	8	5	0
5	4	6	14	10	4

P1	P4	P5	P3	P2

0 3 8 14 18 20

a) Round Robin CPU scheduling :-

Criteria = AT + TQ (Time Quatum) → the processor will give time units to each process

mode = preemptive

TQ = 2

Ex :

PNO	AT	BT	CT	ATA	WT
1	0	4			
2	1	5			
3	2	2			
4	3	1			
5	4	6			
6	6	3			

BT

P1 4 2 0

P2 5 3 1 0

P3 2 0

P4 1 0

P5 6 4 2 0

P6 3 1 0

Ready queue | P1 | P2 | P3 | P1 | P4 | P5 | P2 | P6 | P5 | P2 | P6 | P5 |

P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5

0 2 4 6 8 9 11 13 15 17 18 19 20

Ex-2 : TQ = 3

PNO	AT	BT	CT	ATA	WT
1	0	10 7 4 1 0	22	22	12
2	1	4 1 0	16	15	11
3	2	5 2 0	18	16	11
4	3	3 0	12	9	6

ready queue | P1 | P2 | P3 | P4 | P1 | P2 | P3 | P1 | P1 |

P1	P2	P3	P4	P1	P2	P3	P1	P1

0 3 6 9 12 15 16 18 21 22

Ex-) TQ = 2

PNO	AT	BT	CT	ATA	WT
1	0	5	5	5	0
2	1	3	12	11	6
3	3	2	7	4	2
4	9	3	17	8	5
5	12	3	20	8	5

ready queue: P1 P2 P1 P3 P2 P4 P2 P4 P5 P4 P5 P5

P1	P2	P1	P3	P2	P4	P2	P4	P5	P4	P5	P5

2 4 5 7 9 11 12 14 16 17 19 20

ii) Non-preemptive priority CPU scheduling:

criteria = priority
mode = non-preemptive.

Ex-1:

PNO	AT	BT	priority
1	0	11	2
2	5	28	0 - low
3	12	2	3
4	2	10	1
5	9	16	4 - High

P1 P2 P4 P5 P2 P4 P3

P1	P5	P3	P4	P2

0 11 27 29 39 67

Ex-2:

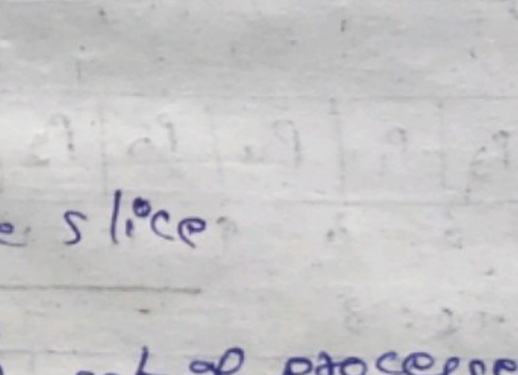
PNO	AT	BT	priority
1	0	9	
2	1	4	
3	2	5	
4	3	7	
6	4	4	

3 → time quantum is also called as time slice

* Deadlock: it is a situation. in which set of processes are blocked. each process holding resource. waiting for another resource.

Necessary conditions to occur deadlocks:

1) mutual exclusion → resources are non sharable
2) no preemption → resource can't release till completion of
3) Hold & wait a process execution
4) circular wait



Deadlock handling Mechanisms

1) Deadlock ignore → simple.
ignore the deadlock by restarting a system

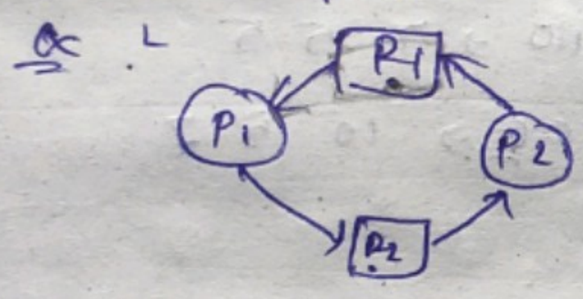
2) Deadlock prevention
3) Deadlock avoidance
4) Deadlock detection & Recovery

Deadlock prevention

1) Make resources are sharable to avoid mutual exclusion
2) use time quantum (TQ) to make premption
3) try to give all the resources prior to the process before it starts execution
4) Give numbering to all the resources to avoid circular wait. so, that the process request the resources in an increasing order.

3) Deadlock detection & Recovery

→ Deadlock detection can be done with RAG (Resource Allocation Graph)



Process - O
Resource - [i]

2 instance (means two processes will use this resource at a time)

→ Recovery for a Resource :
  i) preempt the resource
  ii) Rollback to safe state

→ Recovery for a process :
  i) kill a process
  ii) kill all processes

4) Deadlock Avoidance : ensure the system will never enter into unsafe state

→ Deadlock avoidance can be done using Bankers Algorithm

Bankers Algorithm :

Current work (Available) = Total - Allocation total

if Need ≤ work => work = work + Allocation

Ex: 1

Total A=10, B=5, C=7 can hold.

Process	Allocation A B C	Maximum A B C	(Available) current work A B C	remaining need (max - allocation) A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2	5 3 2	1 2 2
P_2	3 0 2	9 0 2	7 4 3	6 0 0
P_3	2 1 1	4 2 2	7 4 5	2 1 1
P_4	0 0 2	5 3 3	7 5 5	5 3 1
total allocation	7 2 5		10 5 7	

$P_0 \Rightarrow 7\ 4\ 3 \leq 3\ 3\ 2$ ✗

$P_1 \Rightarrow 1\ 2\ 2 \leq 3\ 3\ 2$ ✓ $\Rightarrow 3\ 3\ 2 + 2\ 0\ 0 \Rightarrow 5\ 3\ 2$

$P_2 \Rightarrow 6\ 0\ 0 \leq 5\ 3\ 2$ ✗

$P_3 \Rightarrow 2\ 1\ 1 \leq 5\ 3\ 2$ ✓ $\Rightarrow 5\ 3\ 2 + 2\ 1\ 1 \Rightarrow 7\ 4\ 3$

$P_4 \Rightarrow 5\ 3\ 1 \leq 7\ 4\ 3$ ✓ $\Rightarrow 7\ 4\ 3 + 0\ 0\ 2 \Rightarrow 7\ 4\ 5$

$P_0 \Rightarrow 7\ 4\ 3 \leq 7\ 4\ 5$ ✓ $\Rightarrow 7\ 4\ 5 + 0\ 1\ 0 \Rightarrow 7\ 5\ 5$

$P_2 \Rightarrow 6\ 0\ 0 \leq 7\ 5\ 5$ ✓ $\Rightarrow 7\ 5\ 5 + 3\ 0\ 2 \Rightarrow 10\ 5\ 7$

Safe state = $< P_1, P_3, P_4, P_0, P_2 >$

* $\rightarrow$ max allocation = maximum - allocations

$\rightarrow$

Process	allocation A B C	max A B C	current work A B C	need A B C
P_0	2 1 0	8 6 3	4 3 2	6 5 3
P_1	1 2 2	9 4 3	7 3 3	8 2 1
P_2	0 2 0	5 3 3	7 ~~4~~ ~~5~~ 3	5 1 3
P_3	3 0 1	4 2 3	9 6 3	1 2 2

$P_0 \Rightarrow 6\ 5\ 3 \leq 4\ 3\ 2$ ✗

$P_1 \Rightarrow 8\ 2\ 1 \leq 4\ 3\ 2$ ✗

$P_2 \Rightarrow 5\ 1\ 3 \leq 4\ 3\ 2$ ✗ ~~$4\ 3\ 2 + 0\ 2\ 0 \Rightarrow 4\ 5\ 2$~~

$P_3 \Rightarrow 1\ 2\ 2 \leq 4\ 3\ 2$ ✓ $4\ 3\ 2 + 3\ 0\ 1 \Rightarrow 7\ 3\ 3$

$P_0 \Rightarrow$ ~~$7\ 3\ 3 \leq$~~ $6\ 5\ 3 \leq 7\ 3\ 3$ ✗ ~~$7\ 3\ 3 + 2\ 1\ 0 \Rightarrow 9\ 4\ 3$~~

$P_1 \Rightarrow 8\ 2\ 1 \leq 7\ 3\ 3 \Rightarrow$ ✗

$P_2 \Rightarrow 5\ 1\ 3 \leq 7\ 3\ 3 \Rightarrow$ ✓ $7\ 3\ 3 + 0\ 2\ 0 \Rightarrow 7\ 5\ 3$

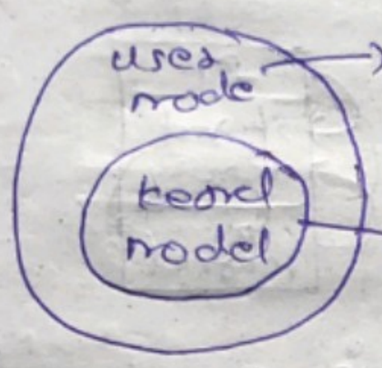
$P_0 \Rightarrow 6\ 5\ 3 \leq 7\ 5\ 3 \Rightarrow$ ✓ $\Rightarrow 5\ 3 + 2\ 1\ 0 \Rightarrow 9\ 6\ 3$

$P_1 \Rightarrow 8\ 2\ 1 \leq 9\ 6\ 3 \Rightarrow$ ✓ $9\ 6\ 3 + 1\ 2\ 2 \Rightarrow 10\ 8\ 5$

$< P_3, P_2, P_0, P_1 >$

System calls : A system call is way for pgm's to interact with the OS

→ system calls are the entry points into the kernel system
→ when a pgm in user mode, it requires access to RAM / Resources, it must ask the kernel to provide access to that resource this is done via something called "system call"



→ the pgms don't have access to any resource directly

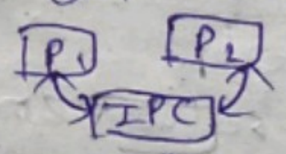
→ Can access the memory/resources directly

kernel → core part of OS, manages of all operations of computer.

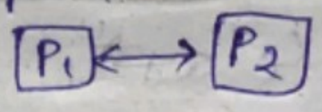
Types :

1) Process control : fork(), wait() etc
2) File control : open(), write(), close()
3) Device control : ioctl()
4) info maintain : getpid(), alarm(), sleep()
5) communication : pipe(), shmget(), mmap()
6) protection : chmod(), umask()

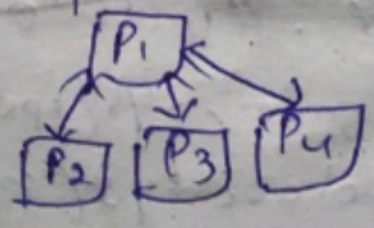
Interprocess communication : Exchange of data b/w two / more independent processes.



1) unicast communication : communication from one single process to another single process.

Ex : socket 

2) Multicast communication : Communication is from one process to grouped processes



Communication models of IPC (Mechanisms)

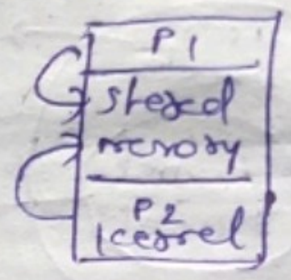
i) Message queue : In which the message is placed either private/public

creating a message queue : Header file : #include <sys/msg.h>
function : int msgget (key_t, int flag)

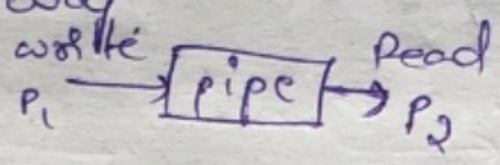
operating : msgsnd(), msgrcv()

2) Shared Memory : In which the process communicate each other

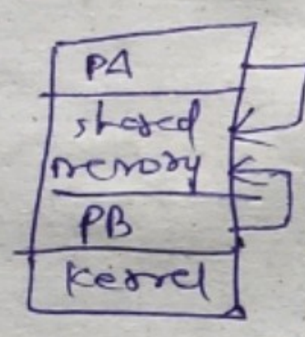
Header file : #include <sys/types.h>
#include <sys/stm.h>
#include <sys/ipc.h>



3) pipe : It provides communication b/w processes in a unidirectional way



4) message passing : It is a way of passing data b/w 2 processes



5) FIFO (named pipe) :→ It is a named pipe.

6) semaphore : It is a special type of integer variable, which used for process synchronization in multiprocessing environment.

→ used to solve critical section problem
→ make the system is not in busy state (be available)
→ there are 2 types of semaphores :
1) Binary semaphore : it takes only '0' or '1'
2) Counting semaphore : it takes any value
→ It performs 2 atomic operation
i) wait ii) signal

```
do{
  wait
  //critical section
  signal
  //remainder section
}while(true);
```

Header files : #include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>

create a semaphore : semget()
control : semctl()

placed

sys/msg.h>
ey-t, int
flag)

ate each

1
ed
ory
rel

processes

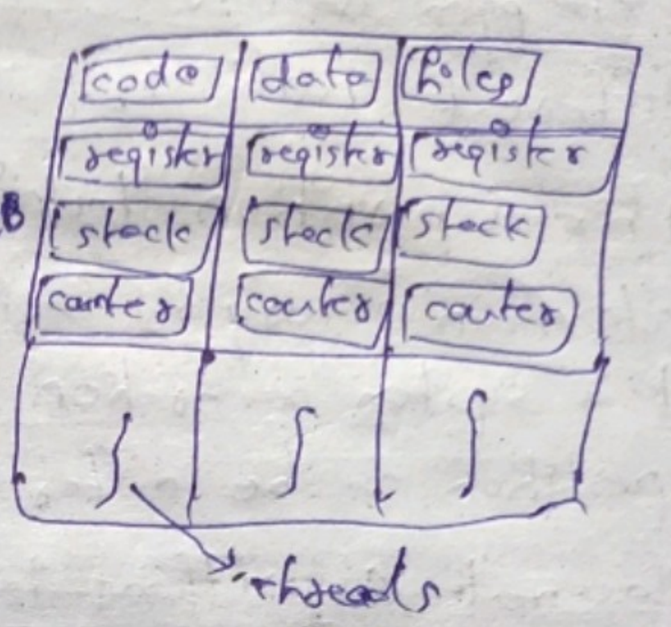
riables,
ultiprocessing

ailable)

e <sys/types.h
c.h>
em.h>
smget()
mctl()

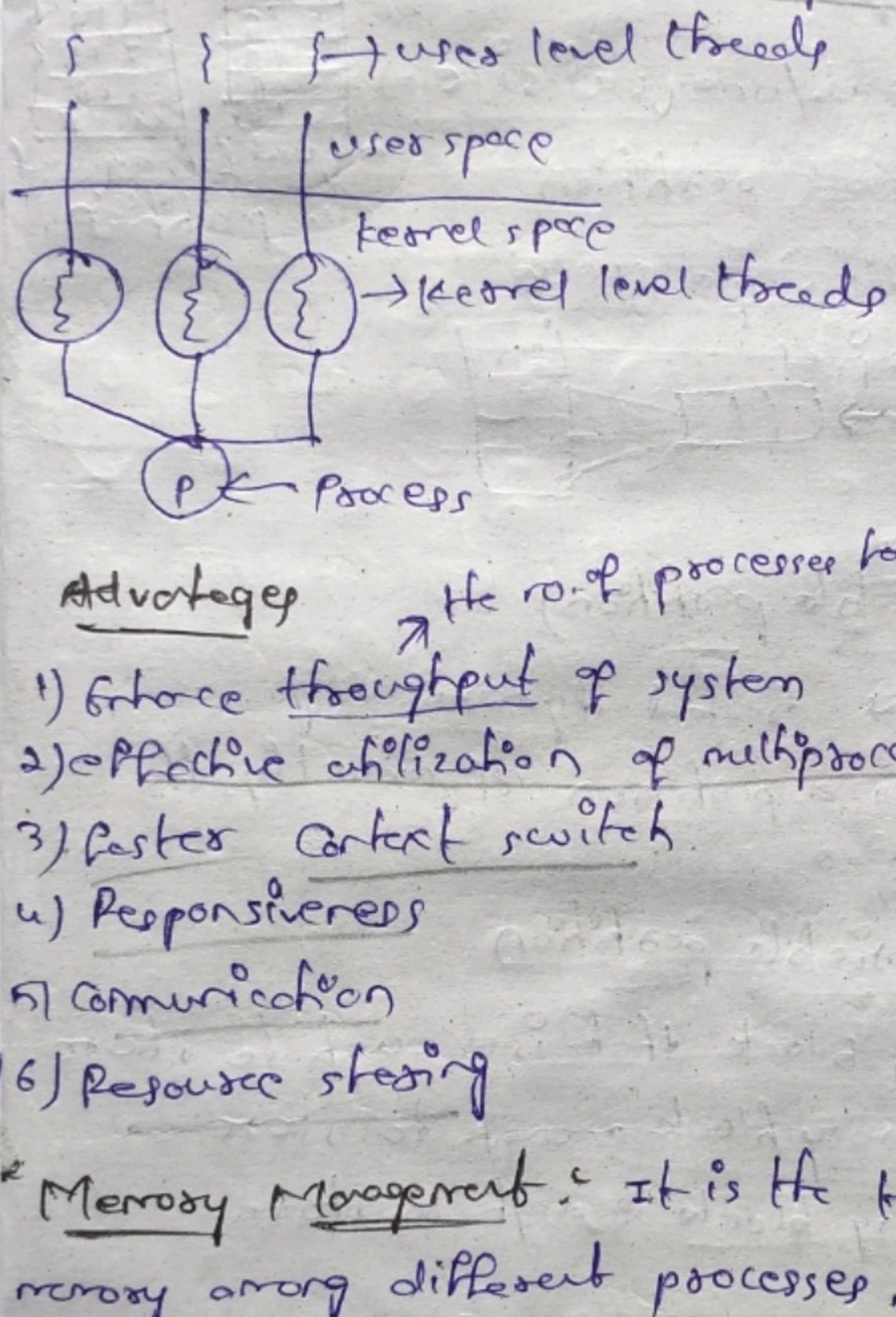
thread : It is a smallest unit of process / it is a light weighted process.

→ threads can share the common data, do not need IPC
→ threads uses to achieve parallelism
→ context switching is faster.



Type of threads :

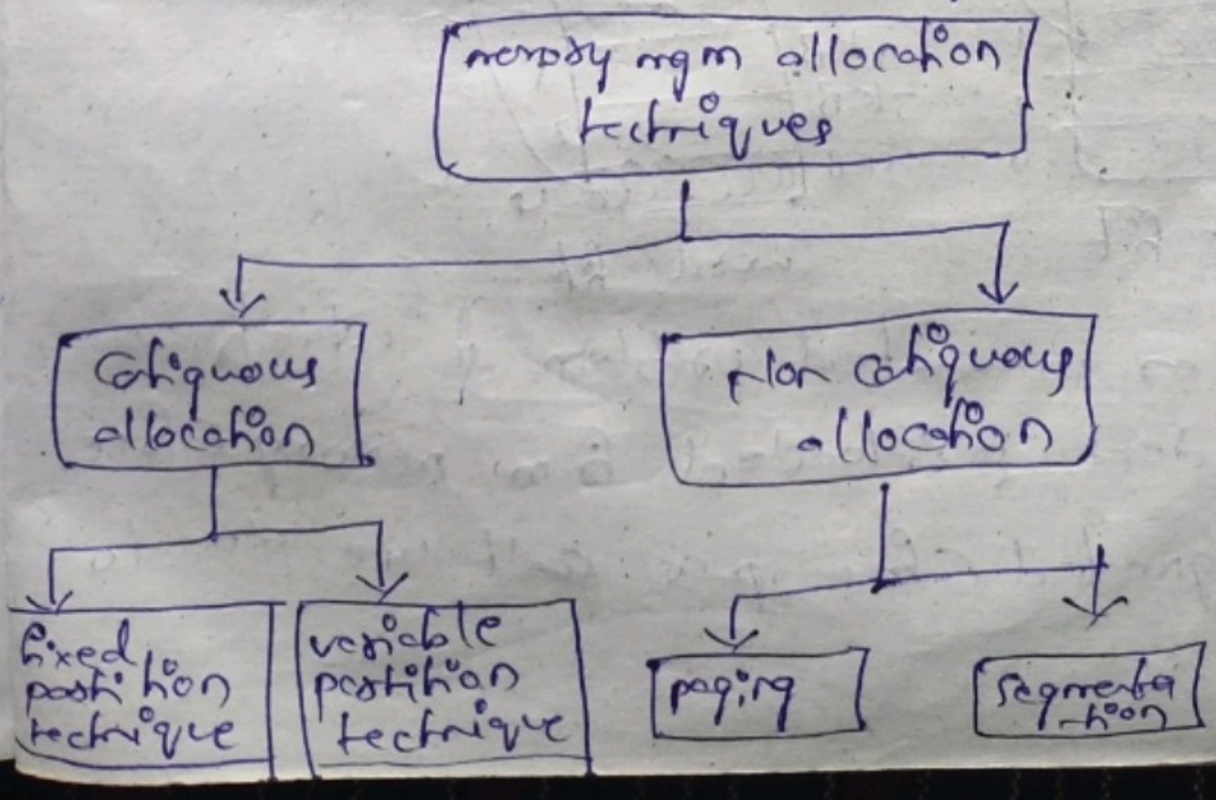
i) Kernel level thread → recognize by os
ii) user level thread → implement by user



Advantages

1) Enforce throughput (↑ the no. of processes has been executed per unit time) of system
2) effective utilization of multiprocessor system
3) Faster context switch
4) Responsiveness
5) Communication
6) Resource sharing

* Memory Management : It is the task of subdividing the memory among different processes. It manages operations b/w main memory & disk during process execution.



Contiguous memory allocation

→ Each pgm occupies a single contiguous block of storage locations

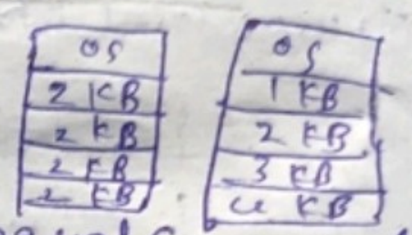
Non-Contiguous Memory allocation

→ the pgm is divided into different blocks & loaded at different portions of the memory

i) Fixed partition technique

→ the main memory is divided into no. of static partitions at a system generation time.

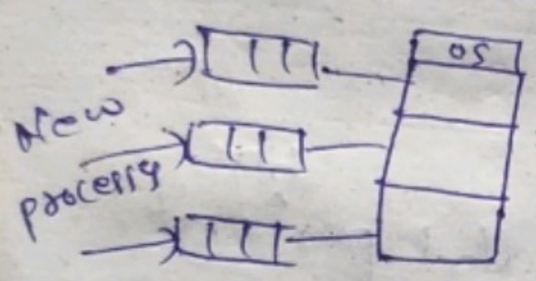
→ partition can be either equal/unequal size →



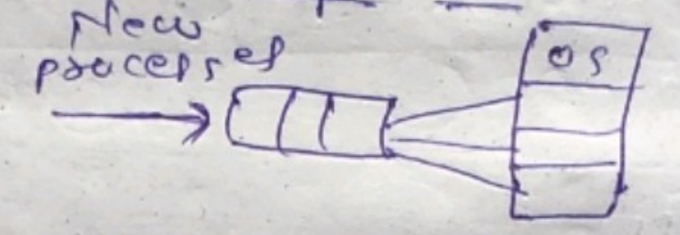
equal & unequal size partition

Placement Algorithms in fixed partition

i) one process queue



ii) single queue



Dynamic partitioning (variable partition)

→ Here, the memory partitions created dynamically

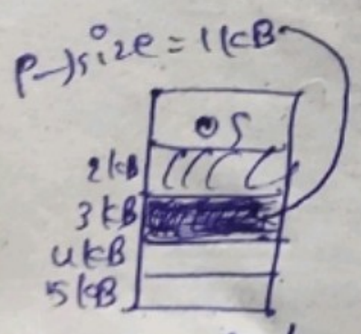
→ Each process is loaded into a partition of exactly the same size of a process.

Placement Algorithms in variable partition

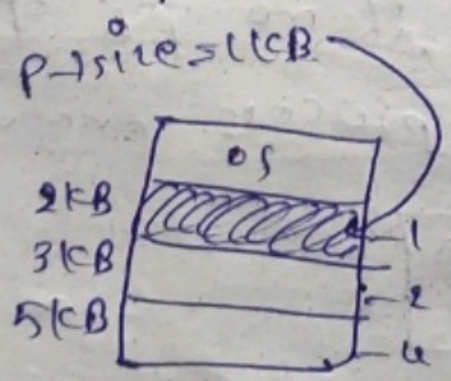
1) first fit : the first memory block that is enough to store

2) best fit : the smallest memory block enough to store

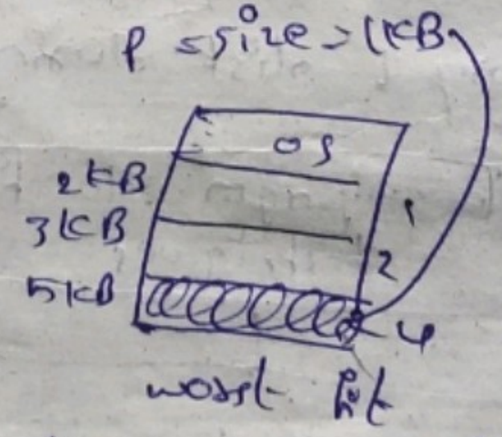
3) worst fit : the largest memory block enough to store



first fit



best fit

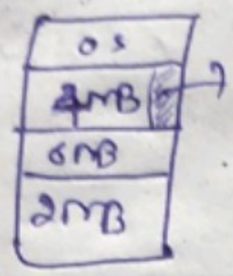


worst fit

Fragmentation : It is an unwanted problem in which the process are loaded & unloaded from memory & free memory space is fragmented (free space can't use due to small size)

Types :-

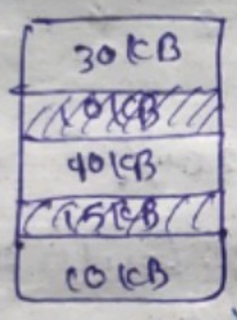
1) Internal fragmentation :- when a process is allocated to a memory block if the process is smaller than the memory request, free space is created in memory.
→ Due to this free space the memory block is unused



→ P_1 = 3MB size
1MB free space

* → To avoid internal fragmentation use dynamic partitioning technique

2) External fragmentation :- the total memory space is enough to satisfy a memory request or reside a process in it. but it is not contiguous, so it can't be used

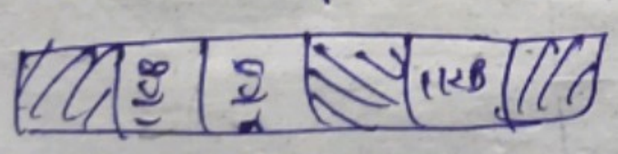


P_1 → 45 KB

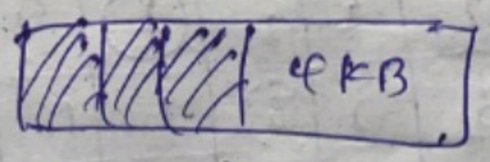
→ To remove external fragmentation use "paging/compaction"

Compaction :- shuffle the free memory blocks into one large memory block to store data.

before compaction



after compaction



Page Replacement Algorithm :- It is used to decide which page needs to replaced when a new page comes in

→ Page Fault :- if the requested page is not available the memory it is treated as page fault.

→ Page hit :- if requested page is available in memory then it is treated as page hit

→ $\text{Hit Ratio} = \frac{\text{No. of hits}}{\text{Hits + faults}} \times 100$

Types

1) FIFO (first in first out)
2) LRU (Least Recently used)
3) optimal page replacement algorithm

1) FIFO : It replaces pages that has been in the memory for the longest time

Ex : Reference string :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0
Assume no. of page frames = 3
Find no. of page faults = ?

	0	0	0	(0)	3	3	(3)	2	2	2	(2)	1	1	1
7	7	(7)	2	2	2	(2)	4	4	(4)	0	0	0	0	0
*	*	*	*	✓	*	*	*	*	*	✓	*	*	✓	

Hits = 3
Faults = 12

Hit ratio = $\frac{3}{3+12} \times 100 + \frac{3}{15} \times 100 = 20\%$

80% fault rate

Ex : 2
string = 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4
No. of frames = 3, no. of page faults = ?

	2	2	(2)	3	3	3	3	(3)	1	1	1
3	3	(3)	0	0	(0)	4	4	4	4	4	4
*	*	*	*	*	*	*	✓	✓	*	*	✓

no. of hits = 3
faults = 9

Hit ratio = $\frac{3}{3+9} \times 100 \Rightarrow \frac{3}{12} \times 100 = 25\%$

no. of frames = 4

		1	1	1	1	1	(1)	2	2	2	2
	2	2	2	2	2	(2)	3	3	3	(3)	4
3	3	3	3	3	(3)	4	4	4	(4)	0	0
*	*	*	*	✓	✓	*	*	*	*	*	✓

no. of hits = 2
faults = 10

Hit ratio = $\frac{2}{12} \times 100 = 16.6\%$

if
→ no. of frames increased the page faults will also increase → Belady's anomaly

2) LRU : In which the page will be replaced which is least recently used.

ex : string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

no. of frame = 5

ex of FIFO

hits = 6 HR = 6/20 × 100 = 30%

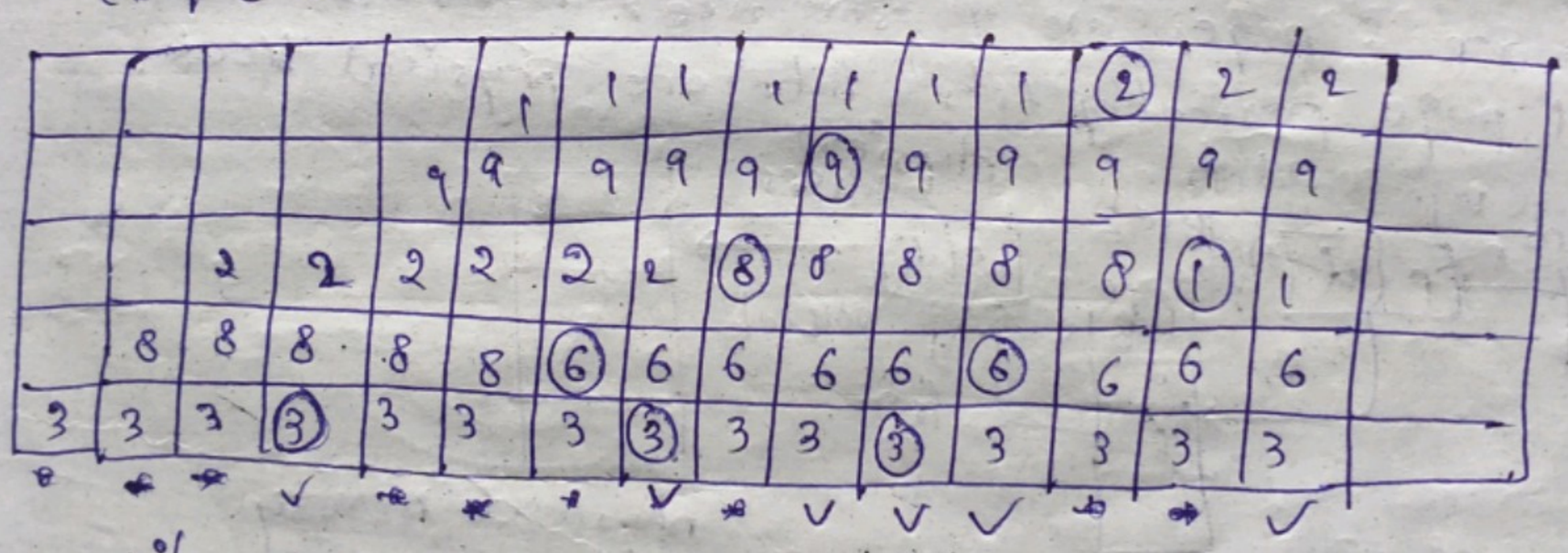
Faults = 9

3) optimal : the page that is not used longest time in the future that is replaced

ex : string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

no. of frame = 5

ex of LRU :



Hits = 6

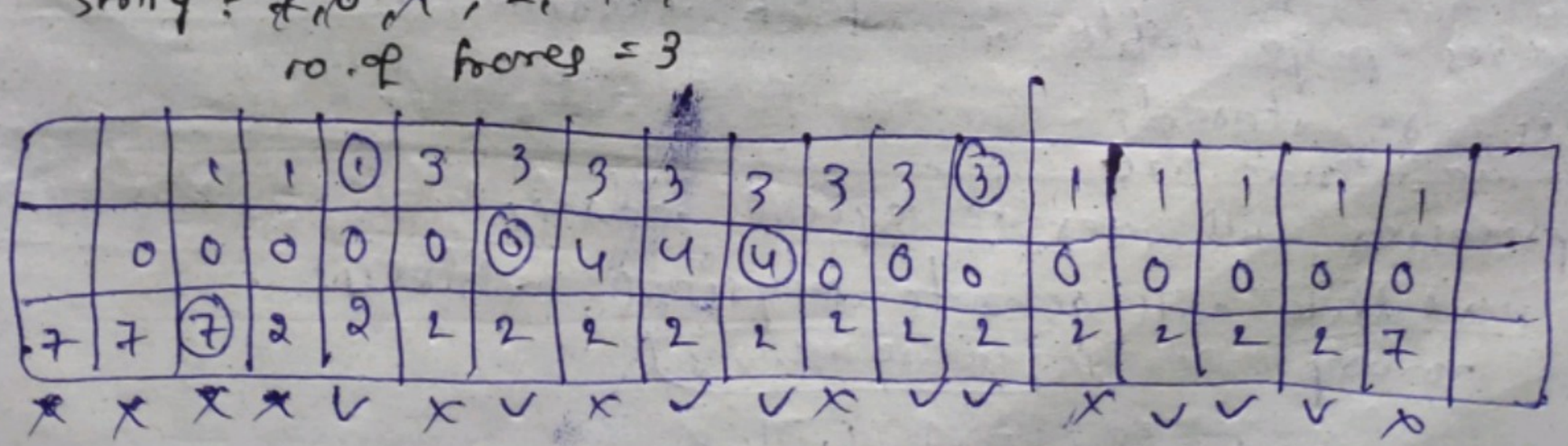
Fault = 9

HR = 6/20 × 100 = 30%

ex of optimal :

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

no. of frames = 3



7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

frame = 4

			2	2	2	4	2	3	2	1	2	2	2						
		1	1	1	1	1	4	4	4	4	4	4	4						
	0	0	0	0	0	0	0	0	0	0	0	0	0						
7	7	7	7	7	3	3	3	3	3	3	3	3	3						
*	*	*	*	✓	*	✓	*	✓	✓	✓	✓	✓	✓						

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

no of frames = 3

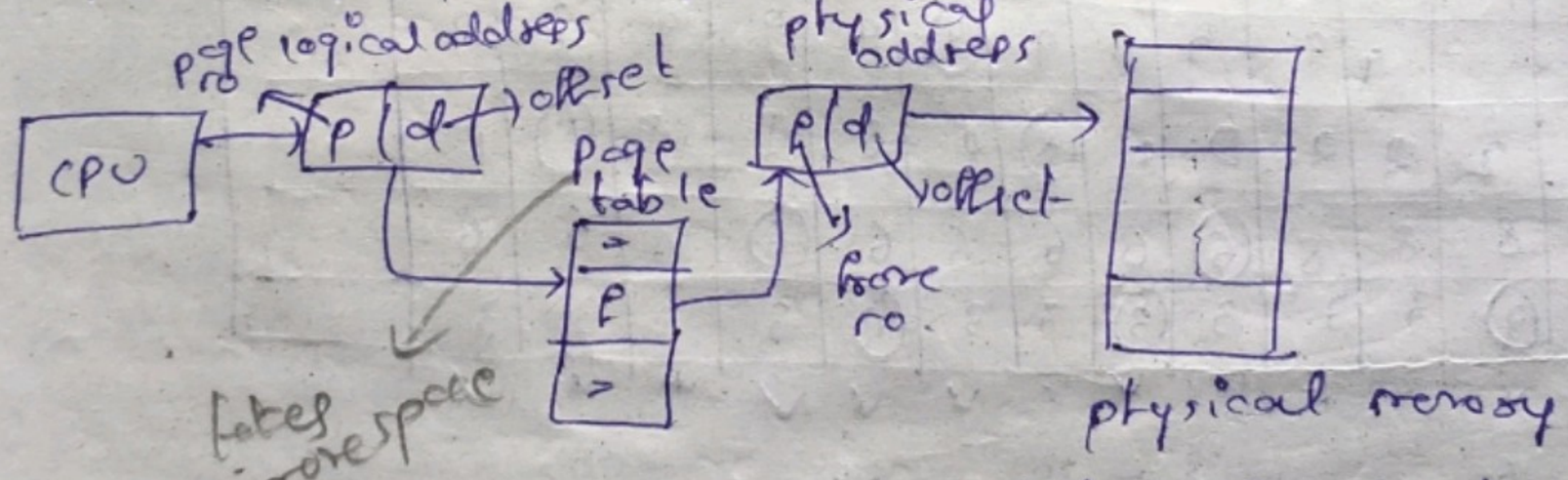
		6	6	6	6	6	2	2	2										
	7	7	7	7	7	7	7	7	2	2									
4	4	4	1	1	1	1	1	1	1										
*	*	*	*	✓	✓	✓	*	✓	✓										

hit = 5, fault = 5

1) Paging :-

→ It is a non-contiguous memory allocation technique. It divides the ~~page~~ physical memory into fixed sized blocks called "frames" & logical memory into "pages"

* the frames & pages are of same size, so results are logical page perfectly fits into one physical block.



→ the problem of external fragmentation is solved

+ swapping is easy b/w pages/frames

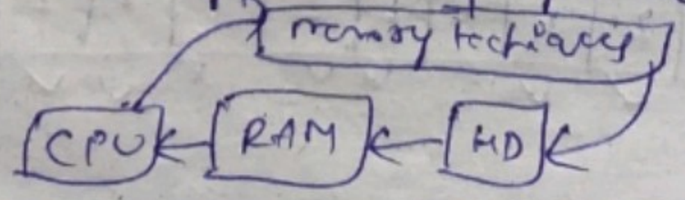
→ Internal fragmentation may occur

→ It uses page table in which information of frames & pages are stored.

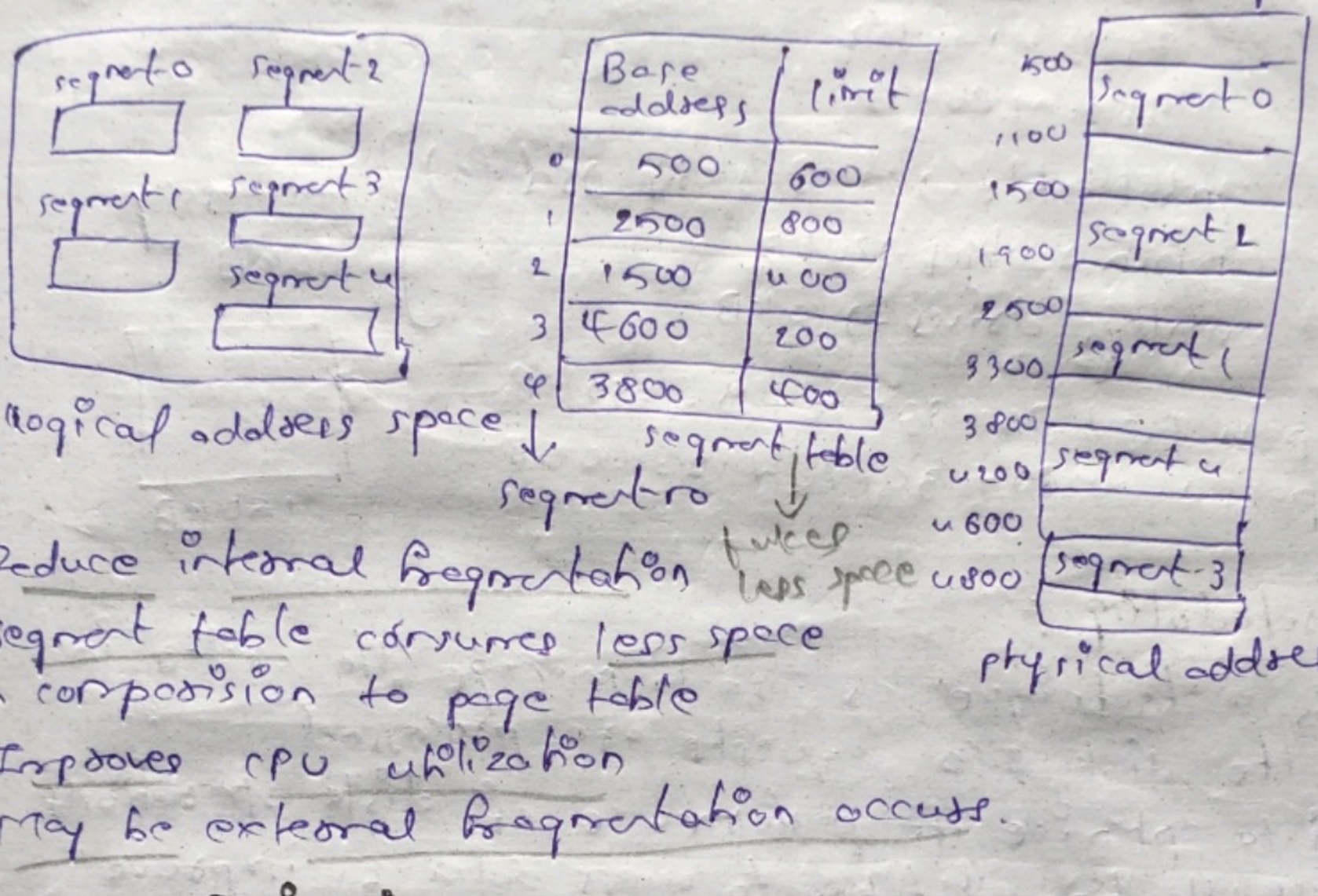
→ page size = frame size

→ the processor will generate "logical address"

→ the hard disk / memory will generate physical address



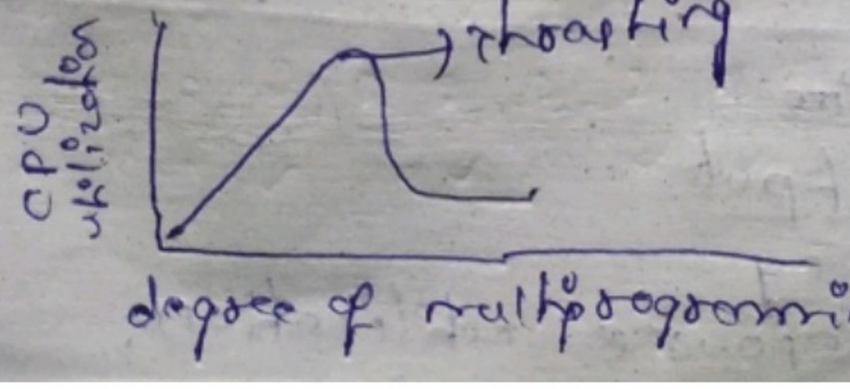
2) segmentation:
→ It is also a non-contiguous memory allocation technique
→ It divides a process into many variable sized areas of memory called "segments"
→ segments holds different types of data
→ segment number & offset is used for referring memory



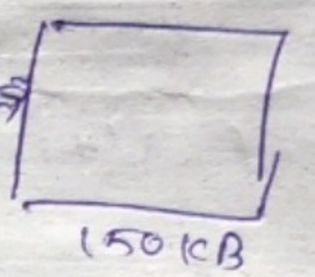
→ Reduce internal fragmentation
→ segment table consumes less space in comparision to page table
→ Improves CPU utilization
→ May be external fragmentation occurs.

3) Demand Paging:
→ It is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU
→ the OS loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start

4) thrashing:
→ It occurs when a system's high memory demand & low physical memory capacity cause it to spend a large amount of time rotating.
→ page b/w main memory & secondary storage



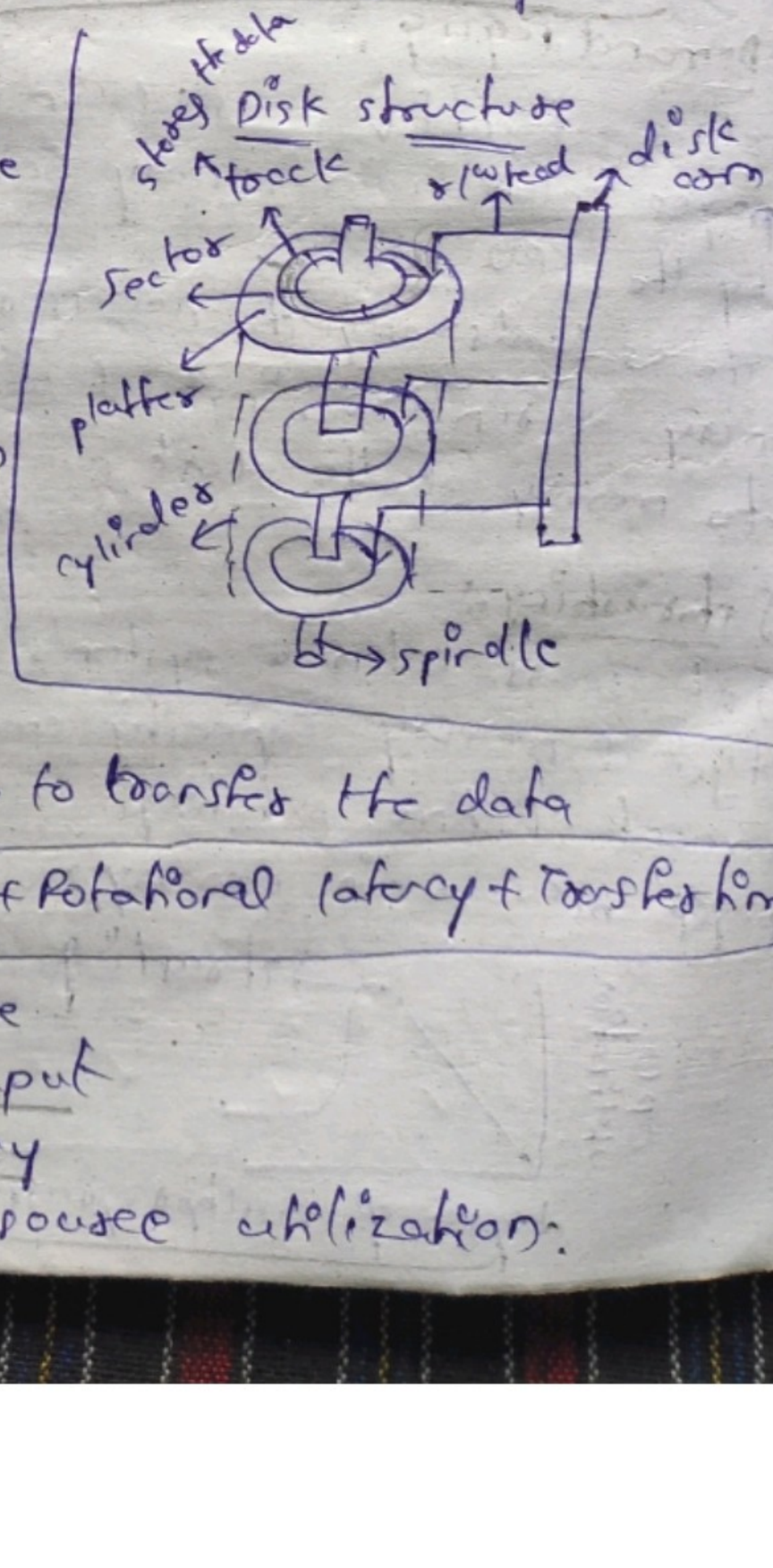
Overlays :
→ os keep in memory only those instructions & data that are needed at any times, when other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed
Ex : Available main memory = 150KB
Total size code = 200KB
Pass 1 → 70KB
Pass 2 → 80KB
symbol table → 30KB (stores OS data) } necessary 60KB
common routines → 20KB
overlays driver → 10KB
for pass 1 → 70+30+20+10 = 130KB
for pass 2 → 80+30+20+10 = 140KB



Disk scheduling :
→ It is a technique, os use to manage the order in which the disk I/O requests are processed.
→ used to optimize the performance of disk operations
→ Reduce the time it takes to access data & improve overall system efficiency

1) Seek time : Time taken to locate the disk arm to a specified track where the data is to be read/write

2) Rotational latency : Time taken by the desired sector of the disk to rotate into a position so that it can be access the r/w heads.



3) Transfer time : It is the time to transfer the data

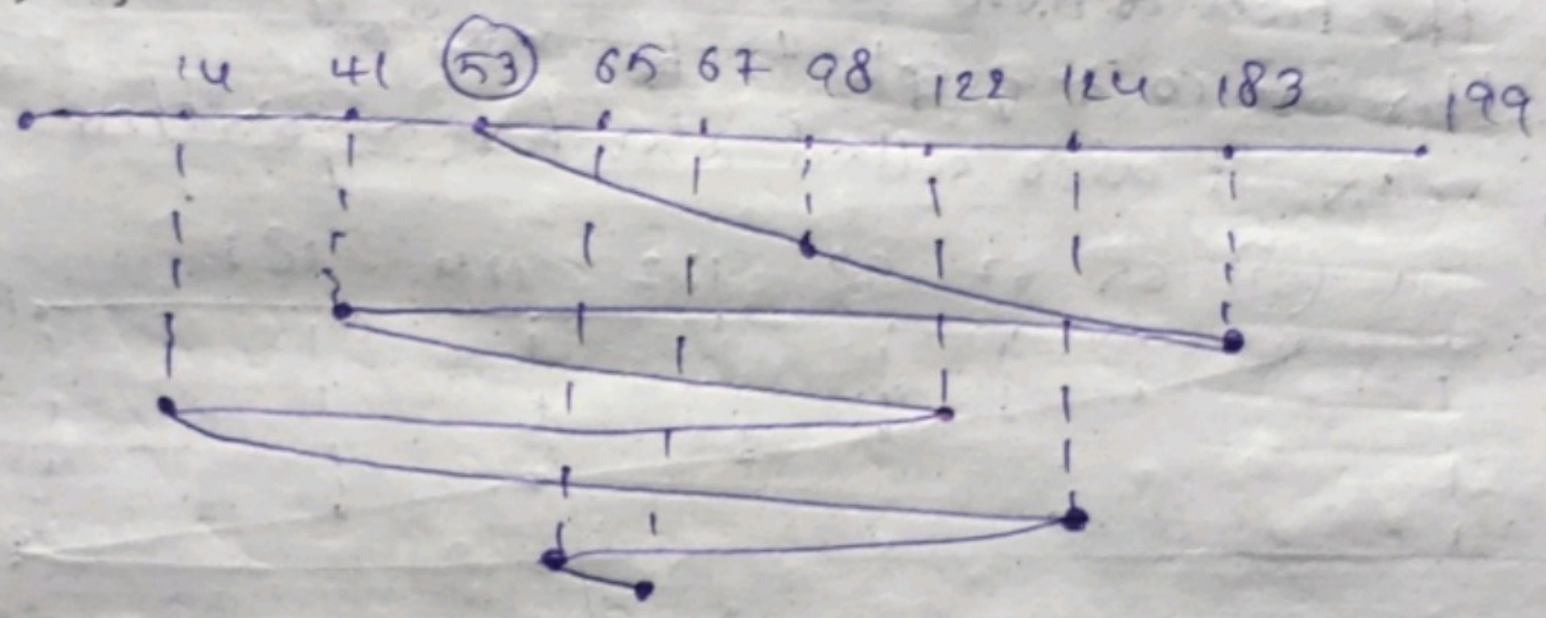
Disk Access Time = seek time + Rotational latency + Transfer time

Goals : 1) minimize seek time
2) maximize throughput
3) minimize latency
4) Efficiency in resource utilization.

1) FCFS: this entertain requests in the order they arrive in the disk queue
→ simple, easy to understand & implement
→ it doesn't cause starvation to any request
→ it results in increased seek time so it is inefficient

Ex: 1 consider a disk queue with requests for I/O to blocks or cylinders 98, 183, 41, 122, 14, 124, 65, 67. the FCFS algorithm is used the head is initially at cylinder 53. the cylinders are numbered from 0 to 199. the total head movement incurred while servicing these requests is

Sol)



Total no. of head movements =

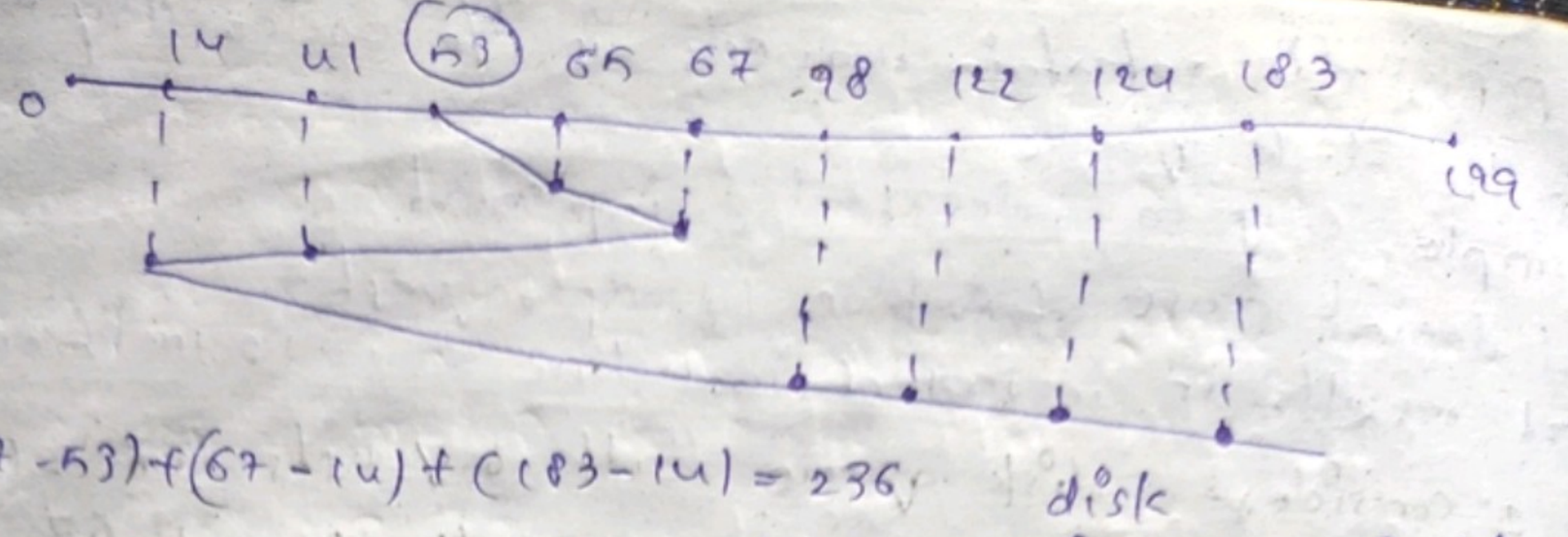
(183-53) + (183-41) + (122-41) + (122-14) + (124-14) + (124-65) + (67-65) = 632

2) SSTF (Shortest Seek Time First): the algorithm services that request that next which requires least no. of Head movements from its current position regardless of the direction.

→ Reduce total seek time compared to FCFS
→ provides increased throughput
→ overhead to find out the closest request
→ the requests which are far from the head might starve.

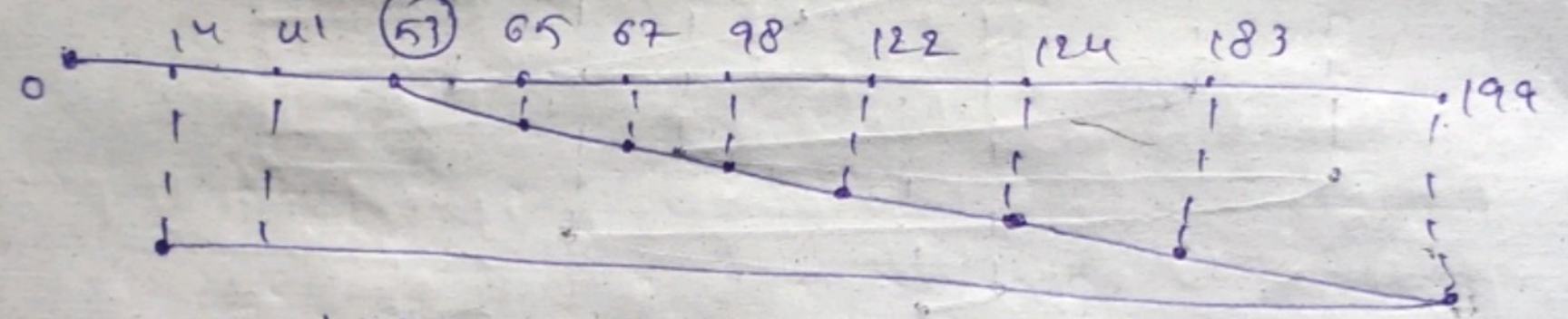
Ex: 1

Ex:



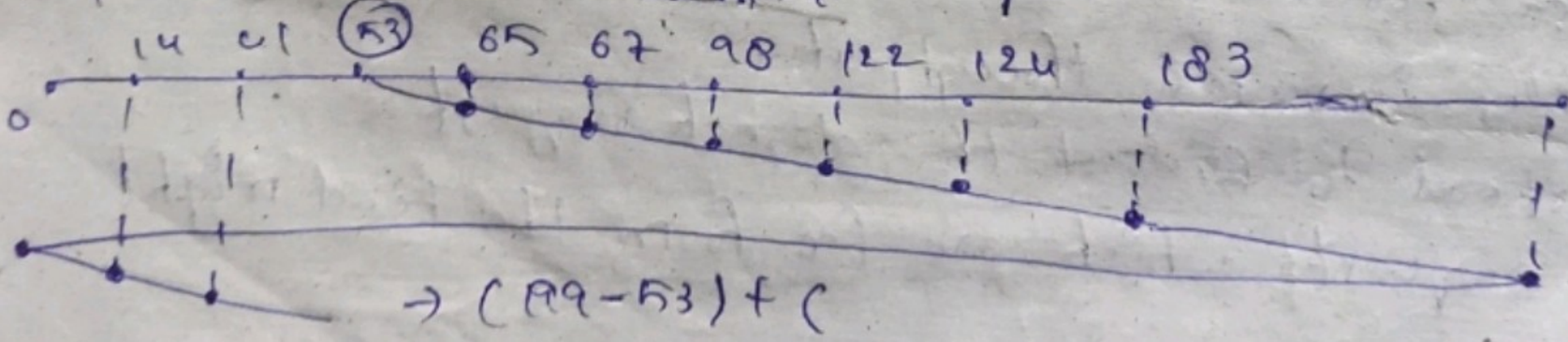
→ (67-53)+(67-14)+(183-14) = 236.

3) SCAN : It scans all the cylinders of the ↑back & forth
→ Head starts from one end of the disk, move towards the other end services all the reverse in b/w
→ No starvation
→ Causes the head to move till the end of the disk even if the there are no request to be serviced.
→ ~~the~~ SCAN is also called elevator algorithm



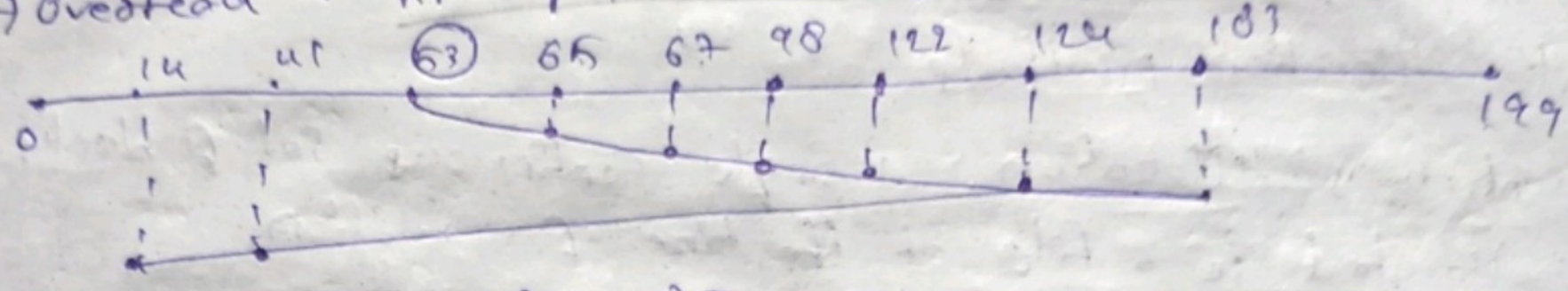
→ (199-53)+(199-14).

4) C-SCAN (Circular Scan) :-
→ Head starts from one end, move towards other end service all the requests in b/w
→ After reaching other end head reverse its direction returns to the starting & without servicing any request in b/w Same process repeats
→ provides uniform waiting time, better response time
→ takes more head movement compared to scan



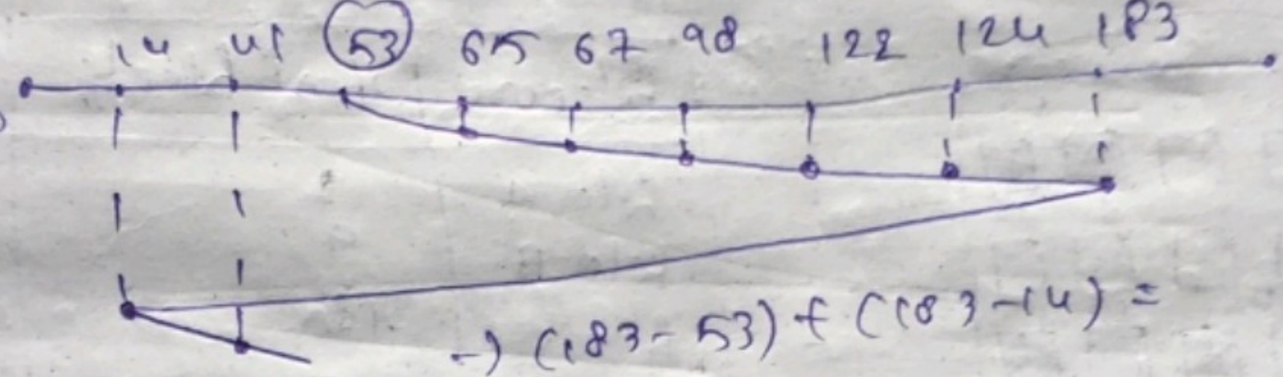
→ (199-53)+(

5) LOOK : Head starts from one end, & move towards last request service all request in b/w
→ Head reverse its direction returns to first request service all requests in b/w
→ provides better performance than SCAN
→ No starvation.
→ Overhead of finding the last request



→ (183-53) + (183-14) =

6) C-LOOK =



→ (183-53) + (183-14) =

File system Management:
→ It allows us to store, organize & manage files & directories or a storage device
→ File: A file is a collection of related information that is recorded on secondary storage (or) file is a collection of logically related entities
→ the name of file is divided into 2 parts filename & its extension.

File Attributes
1) Name
2) Type
3) size
4) creation date
5) Author
6) Last modified
7) protection

File Types:
1) Executable file - .exe, .com, .bin
2) object file - .obj, .o
3) source code file - c, Java, pas etc
4) Text file - .txt, .doc
5) word processor file - .docx, .rtf, .wp etc
6) Archive file - .arc, .zip, .tar
7) Multimedia file - .mpeg, .mov, .jpeg
8) print/view files - .gif, .pdf

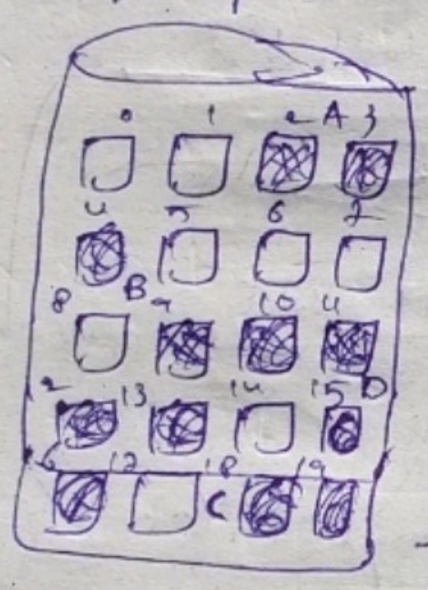
File operations

1) read 2) write 3) open 4) close 5) create 6) Append
7) Truncate 8) Delete

Directory: Collection of files is a directory
→ It contains information about the files, including attributes, location, etc

File Allocation methods:

1) Continuous Allocation: A single continuous set of blocks is allocated to a file at the time of creation.



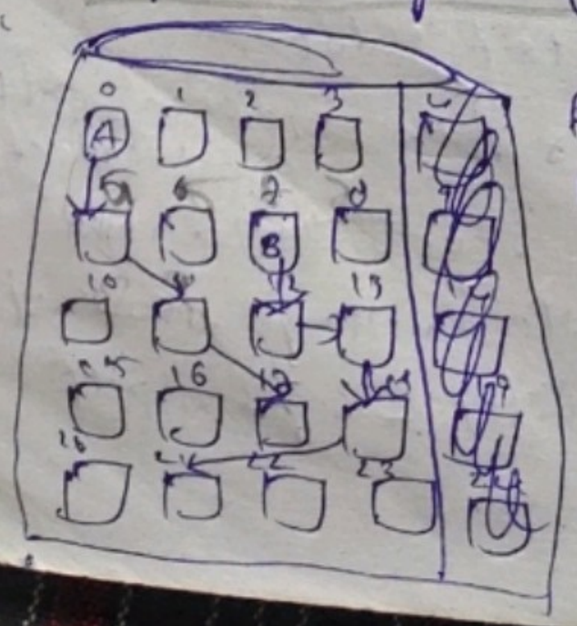
File Allocation table

Filename	start block	length
File A	2	3
File B	9	5
File C	18	2
File D	15	2

→ external fragmentation will occur
→ It is necessary to declare size of the file at the time of creation

2) Linked Allocation (Non-contiguous Allocation):
→ Allocation can be done in an individual basis
→ Each block contains a pointer to the next block
→ Any free block can be added to the existing blocks
→ There is no external fragmentation because only one block at a time is needed
→ There can be internal fragmentation exists
→ Instead of maintaining the pointers in every disk block
→ It supports only sequential access of files.

Ex:



File A :→ 0, 4, 9, 14
File B → 6, 10, 11, 15, 17

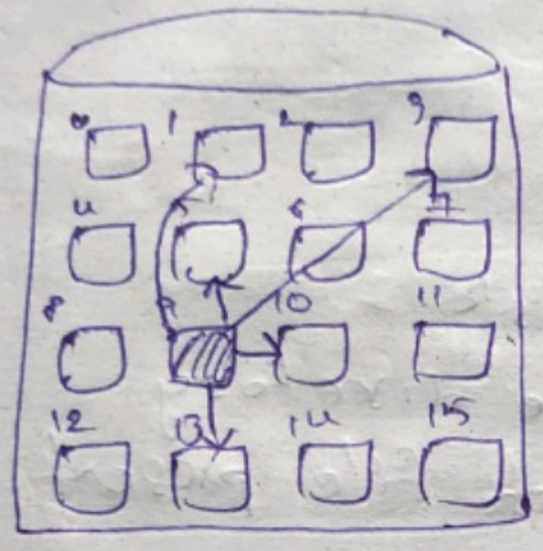
3) Indexed Allocation :-

→ In file allocation table containg a seperate one level index for each file

→ the index has one entry block allocated to the file

→ eliminates external fragmentation

→ supports both sequential & direct access methods



File-B : index block – 9 {1, 3, 10, 13, 15

File Access methods :-

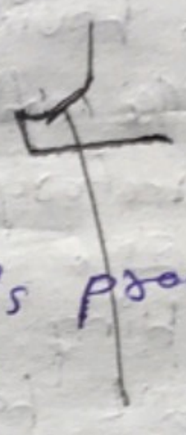
1) Sequential Access :- Information in the file is processed in order, one record after the other

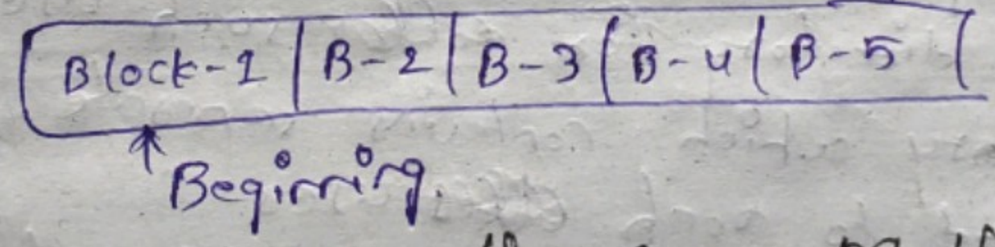
Ex :- Editor, compiler.

read next :- reads the next position of the file, automatically.

advances a file pointer

write next :- append to the end of the file.



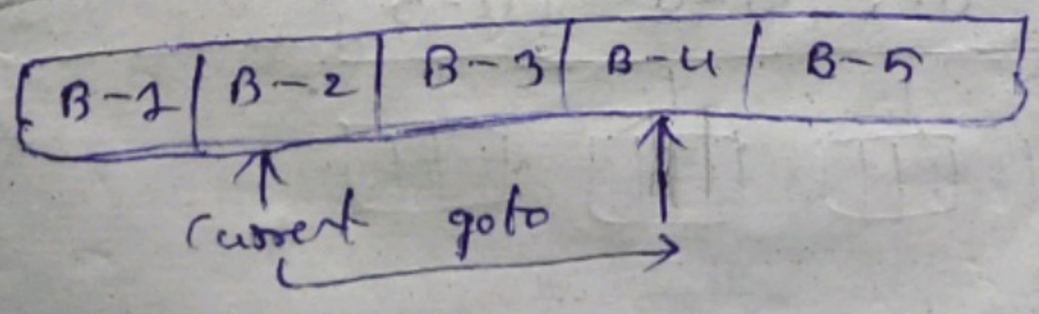


2) Direct Access (Relative Access Method) :-

→ It allows the ~~particular~~ program to read & write in the particular order

→ decreases average access time

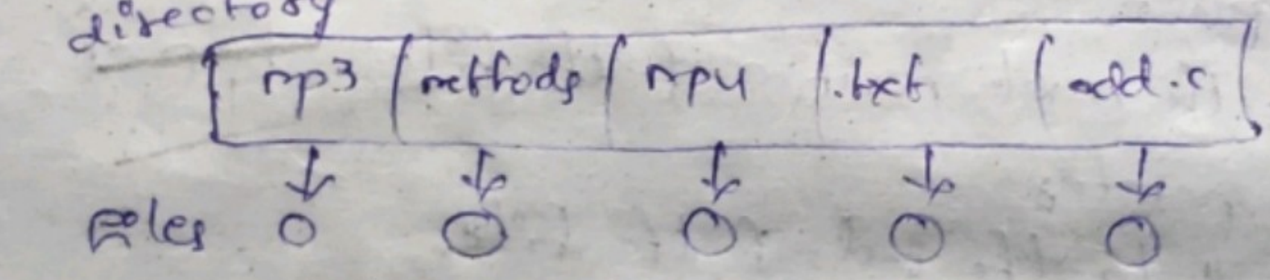
→ Highest, storage overhead, complex implementation



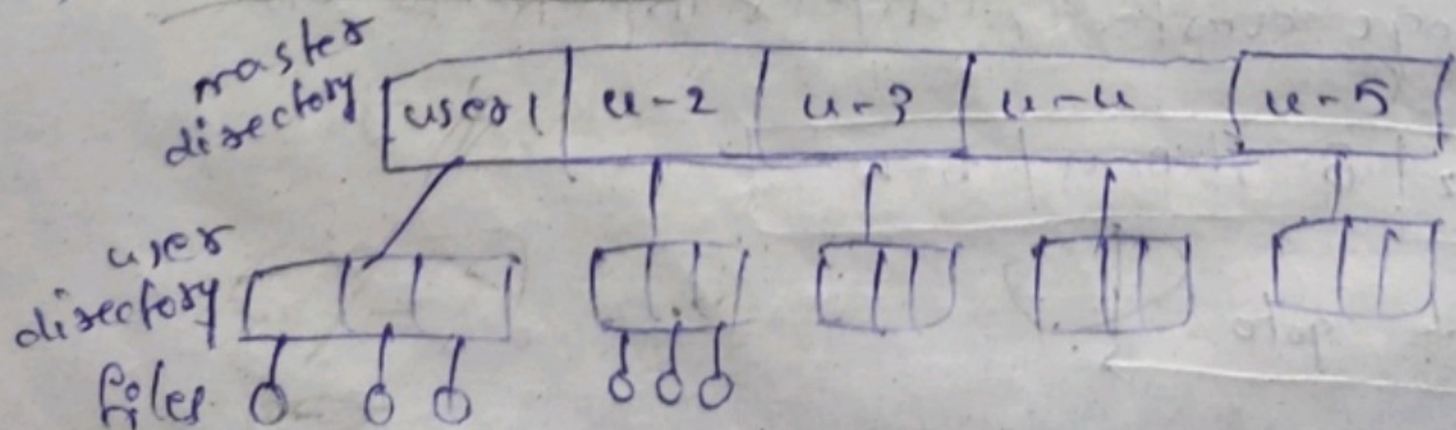
3) Indexed sequential Access:
→ It builts on the top of the sequential Access Method
→ Construct index for the file, contains a pointer to various blocks
→ To find a record in file, we first search the index, by the help of pointer we access file directly
→ efficient searching
→ Additional storage is required

Directory structure

1) single level directory:
→ It is having all the files into the disk
→ the entire system is having only one directory
→ the directory contains one entry for each file present on the file system
→ searching faster
→ we can't have two files with same name
→ searching file may take more time if it is big

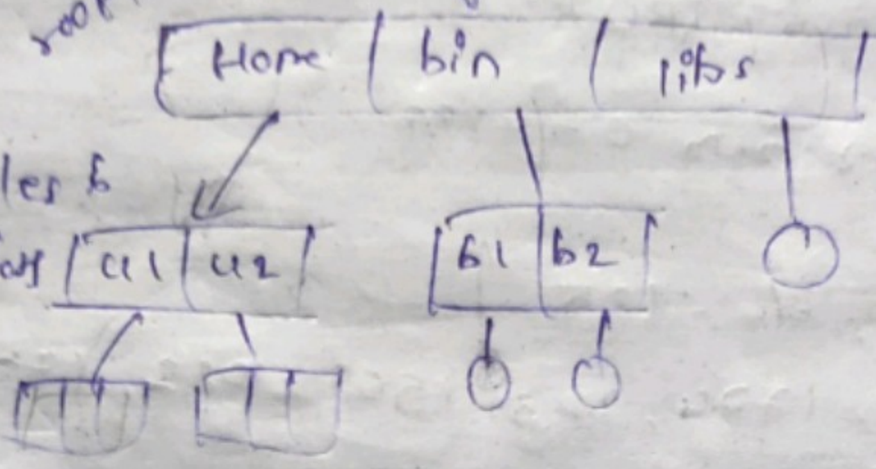


2) Two-level directory:
→ Creates separate directory for such user
→ one master directory which contains separate directories dedicated to each users
→ each file has a path file like c:/user1/files/bin/add.exe
→ Searching is efficient
→ Diff. user can have same file names
→ some kind of file can be grouped into single directory



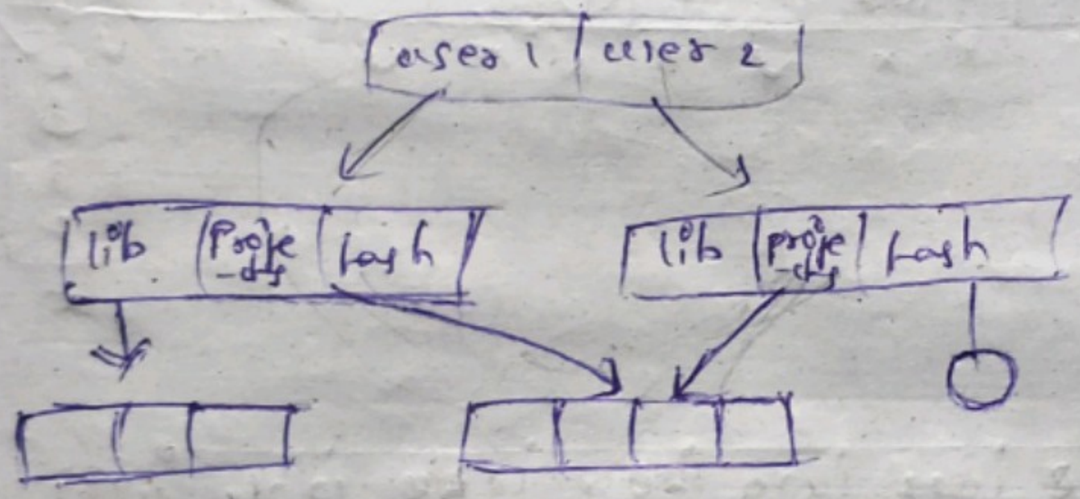
3) Tree structured Directory :-

→ Any directory entry can be a file, sub directory
→ similar kind of files can be grouped into single directory
→ Each user has its own directory
→ only administrator can have the access of root directory



4) Acyclic Graph Directory :-

→ Two or more directory entry can point to the same file / sub directory
→ Can have multiple paths for a same file



Monitors

→ It is an advanced process synchronization technique
→ It enable threads to mutual exclusion & wait() for a given condition to become true
→ It is a prg'ing language construct & abstract datatype
→ It has a shared variable & collection of procedure executing on the shared variable

Syntax :-

```
monitor
{
//shared variable declaration
data variables
procedure P1() {..}
    u       u
initialization code () {...}
},
```

→ Mutual exclusion is automatic
→ less difficult to implement
→ overcome timing.

TS-ECET-2023

Ex-1: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

frame size = 3

page faults = 15 using LRU?

Ex-2: disk drive cylinder from 0 - 4999

Now head is → 143

previous request is = 125

The queue = 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

use SCAN?

1A)

		3	3	3	①	1	1	②	2	②	2	2	⑥	6	6	①	1	1
	2	2	2	②	2	2	⑥	6	6	6	③	3	3	③	3	3	3	③
1	1	1	④	4	4	⑤	5	5	①	1	1	⑦	7	7	②	2	②	2
x	x	x	x	✓	x	x	x	x	x	✓	x	x	x	✓	x	x	✓	✓

page = 15 faults

⑥
3
2

x

2A)

0 86 130 125 143 913 948 1022 1470 1509 1750 1774 4999