date
5/02/24

Process management and synchronization

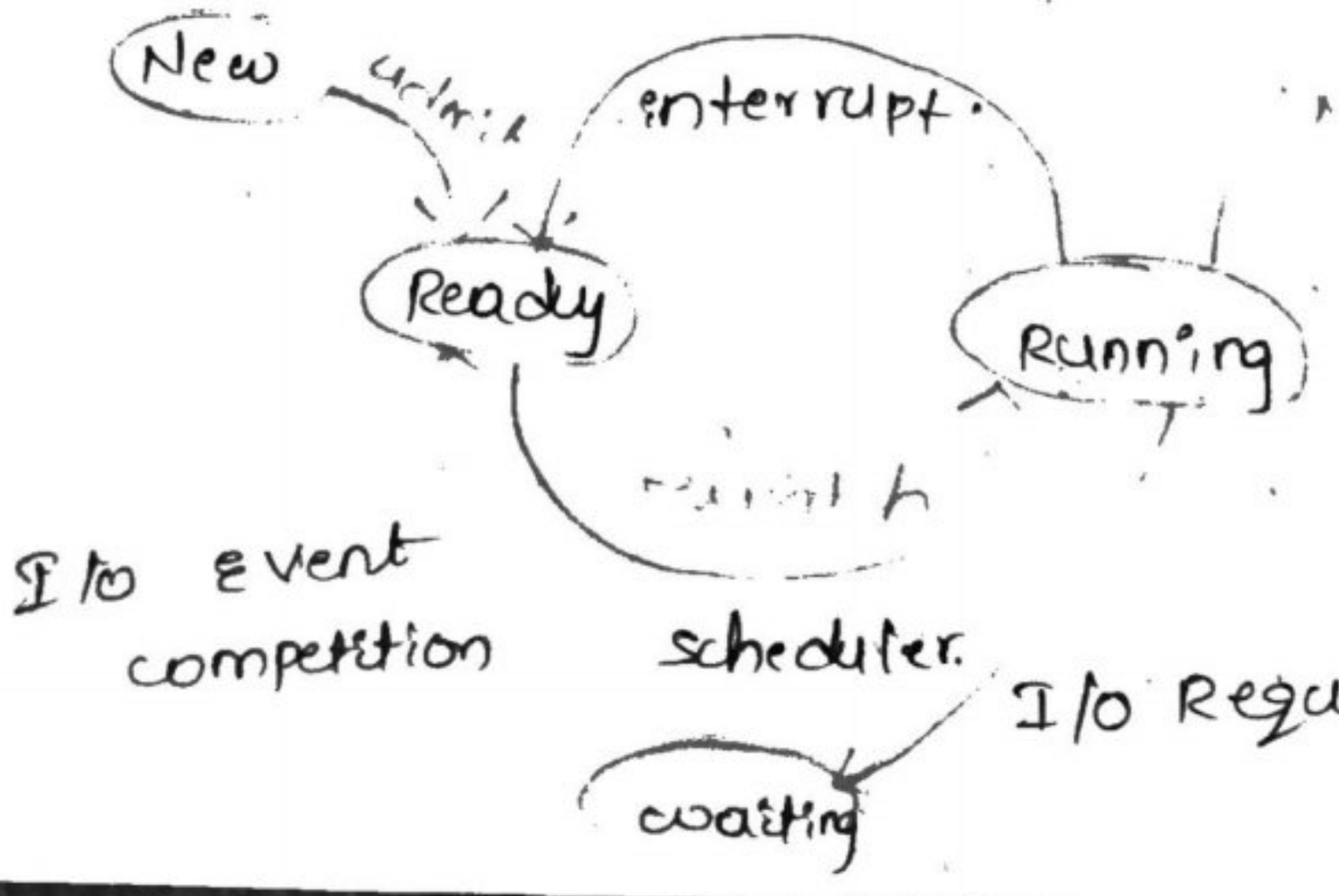
Process	program
1. process is a collection of machine instructions.	1. program is a collection of normal instructions.
2. process is a excuti -on of entire application using processor. (using more than one program).	2. program is a excution of unit of application using single cpu (or) multi processor.
3. process is used different operation functions are :- i) program counter ii) Register iii) stack. iv) Local & global Variab -les. v) Heap algorithm (or) A process is a program in excuetion. A process is more than program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of program counter and content of the processor's register. A process generally includes the process stack, which contains	Ex:- import java.lang.*; class car { int car cost; string car name; void set Data() { int c cost = 1,50,000; String c name = "AUDi"; } void display() { System.out.println("The car cost is" + ccost); S.O.P("the car name is" + c name); } Public static void main(string args[]) {

process	program

temporary data (such as	Car c = new Car();
	c.setData();
	c.display();
	}
	}

	output:- The car cost is 1,50,000 The car name is Audi.
* process supported object code	* program supported source code.
* process considered as dynamic.	* program is always considered as static.
* dynamic means can be alter.	* static means can not be alter.
* process stored in main memory.	* program stored in secondary memory.
* process supports the assembly languages. (8085, 8086)	* program supports the high level languages. (C, C++, Java, ... etc).

Process state Diagram :-



process state :-

- process state diagram
- process has 5 states they are :-

1. New / Born / create state
2. Run
3. Running
4. Terminated
5. Wait / Block.

create state :-

*It accepts different information from private areas. create state is responsible for sending new state information to other state using admit operation.

Run state :-

It accepts different processes from create state. It arranges the information into queue from run state is also responsible for sending any process information from ready queue to running state using dispatch function using for excuetion purpose.

Running state :-

It is under CPU control. It is used properly excuete the process that is fetched from the CPU to main memory.

Wait/Block state :-

If any process do not recieve input information from I/o devices to running state, then the corresponding processes are sent to the wait/block with the help of suspend function.

Terminated State :-

* It accepts successfully excuted process information.

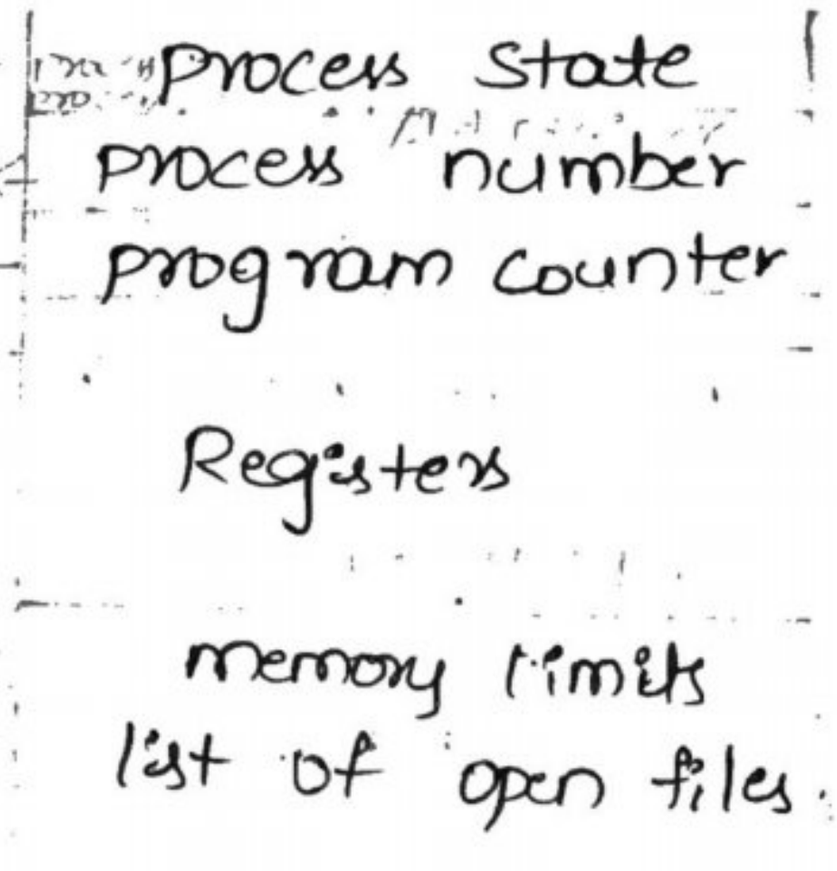
Process Control block :-

* Each and every running process consists of information.

* Every process running information must be stored under a control unit i.e., process control unit technically called "PCB"

* PCB has different partitions. They are process states: It maintains different states such as run/running/terminated. It consists of two other partitions. They are :-

1) parent process.
2) child process.



* process hierarchy is used to prioritize different processess.

process Id is used to allocate different numbers to each process.

* PC is used to prioritize different processes.

process ID is used to allocate different numbers to each process.

* PC is used to fetch the address of the next instruction

* Register recieves process information from PC. It stores instruction temporarily.

* Every process requires I/O information from I/O

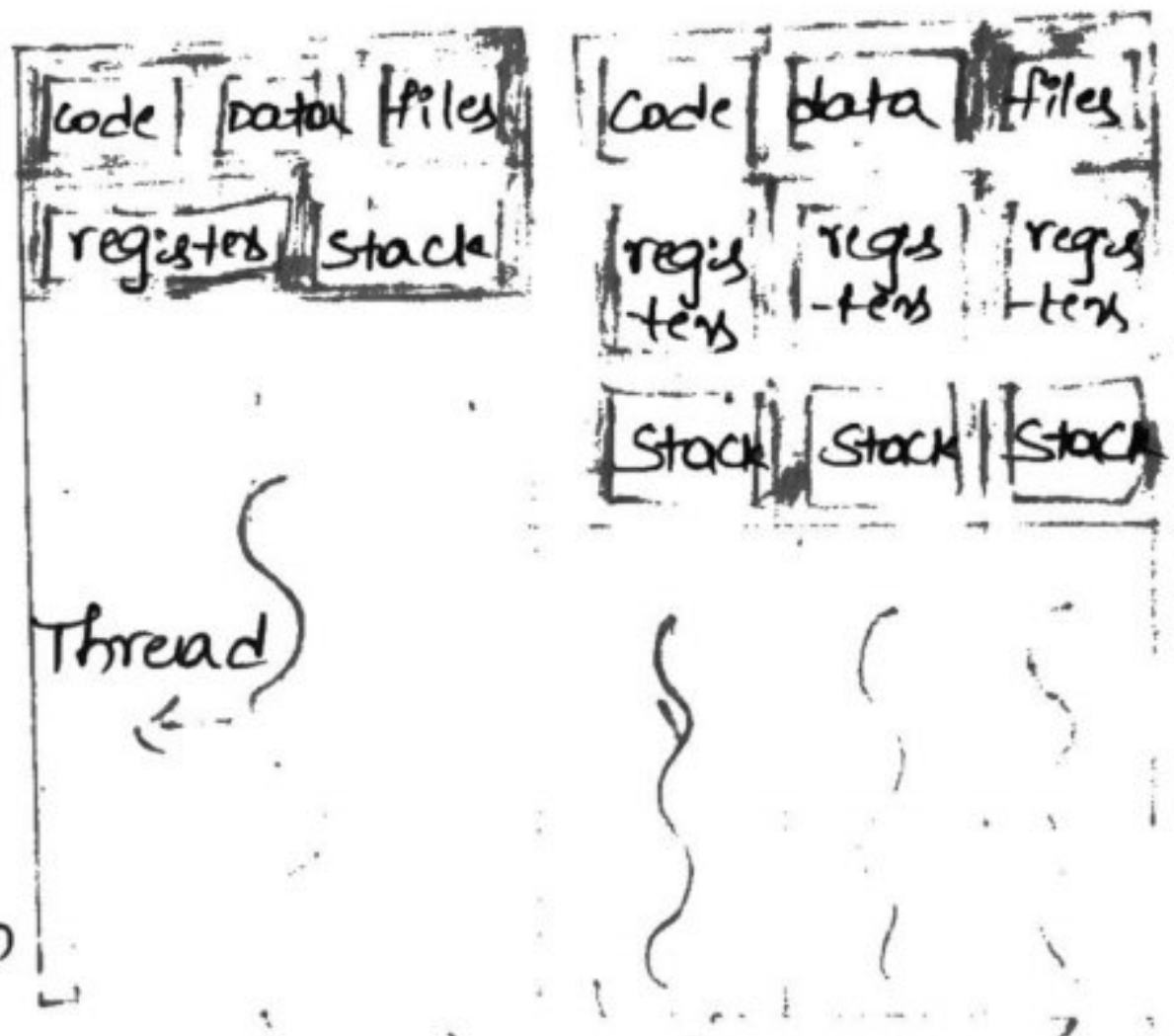
devices.

* Memory pointer stores successfully executed information under main memory sometimes memory is flooded, then we use <u>swap out</u> operation

Threads :-

what is a thread?

* A thread is a path of Execution within a process. A process can contain Multiple threads.



Why Multithreading?

A thread is also known as light weight process.

The idea is to achieve parallelism by dividing a process into multiple tabs can be different threads. Ms word uses Multiple threads: on thread to format the text, another thread to process inputs, e.t.c.

<u>Advantages of thread over process</u> :-

1. Responsiveness :- If the process is divided into multiple threads, if one thread complete its execution, then its output can be immediately return.

2. Faster context switch :- context switch time between threads is lower compared to the process context switch. process context switching requires More Overhead from the cpu.

3. Effective utilization of Multiprocessor system :- If we have multiple threads in a single process, then we schedule multiple thready on Multiple processor, This will makes process excuetion faster.

4. Resource sharing :- Resources like code, data and files can be shared among all the threads within a process.

Note :- stack and registers can't be shared among all the threads.

Each thread has its own stack and register

5. Communication :- Communication between multiple threads is easier, as the thread shares common address spaces. while in process we have to follow some specific communication technique for communication between two process.

many to many model :-

User threads :-

The

* Thread management done by user-level threads library three primary thread libraries.

* POSIX * pthread - P threads the threads extension of the POSIX standard may be provided as either a user or kernel-level library.

* win 32 thread - The win 32 thread library is a kernal level library available on windows system.

* Java threads - The java thread API allows threads to be created and managed direct in java program.

Multi threading Mode :-

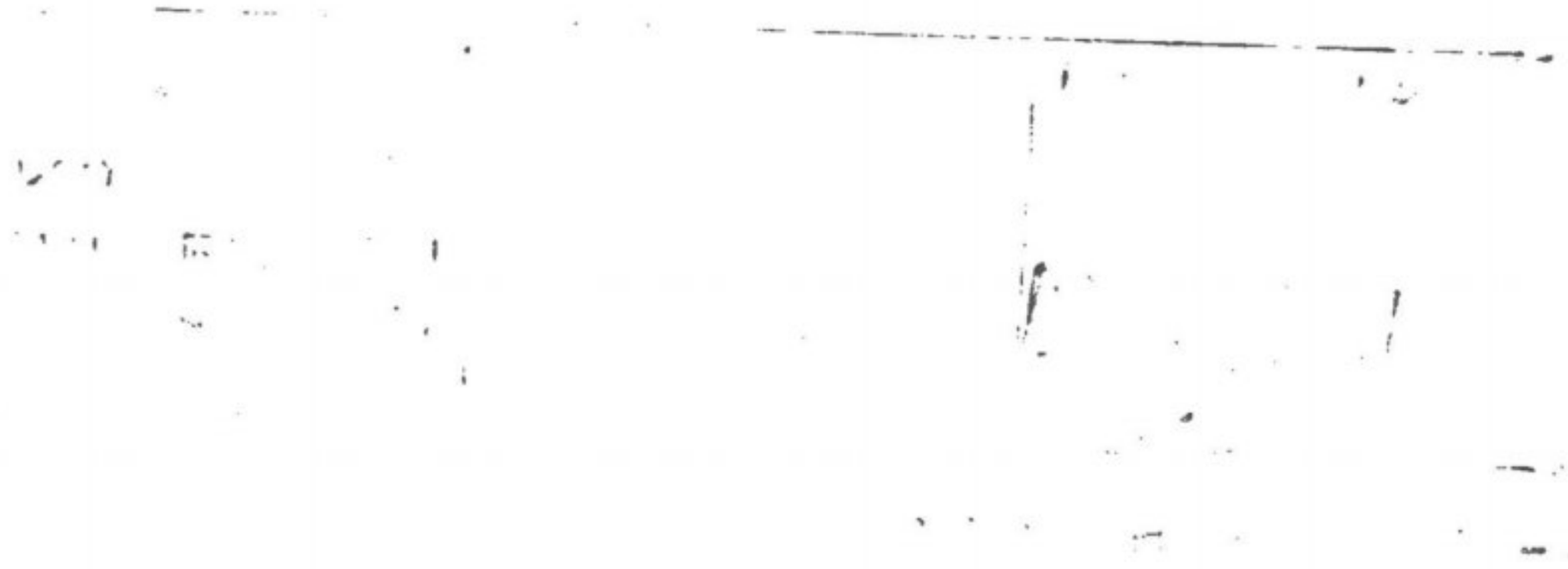
* many-to-one

* one to one

* many to many.

Many to many model :-

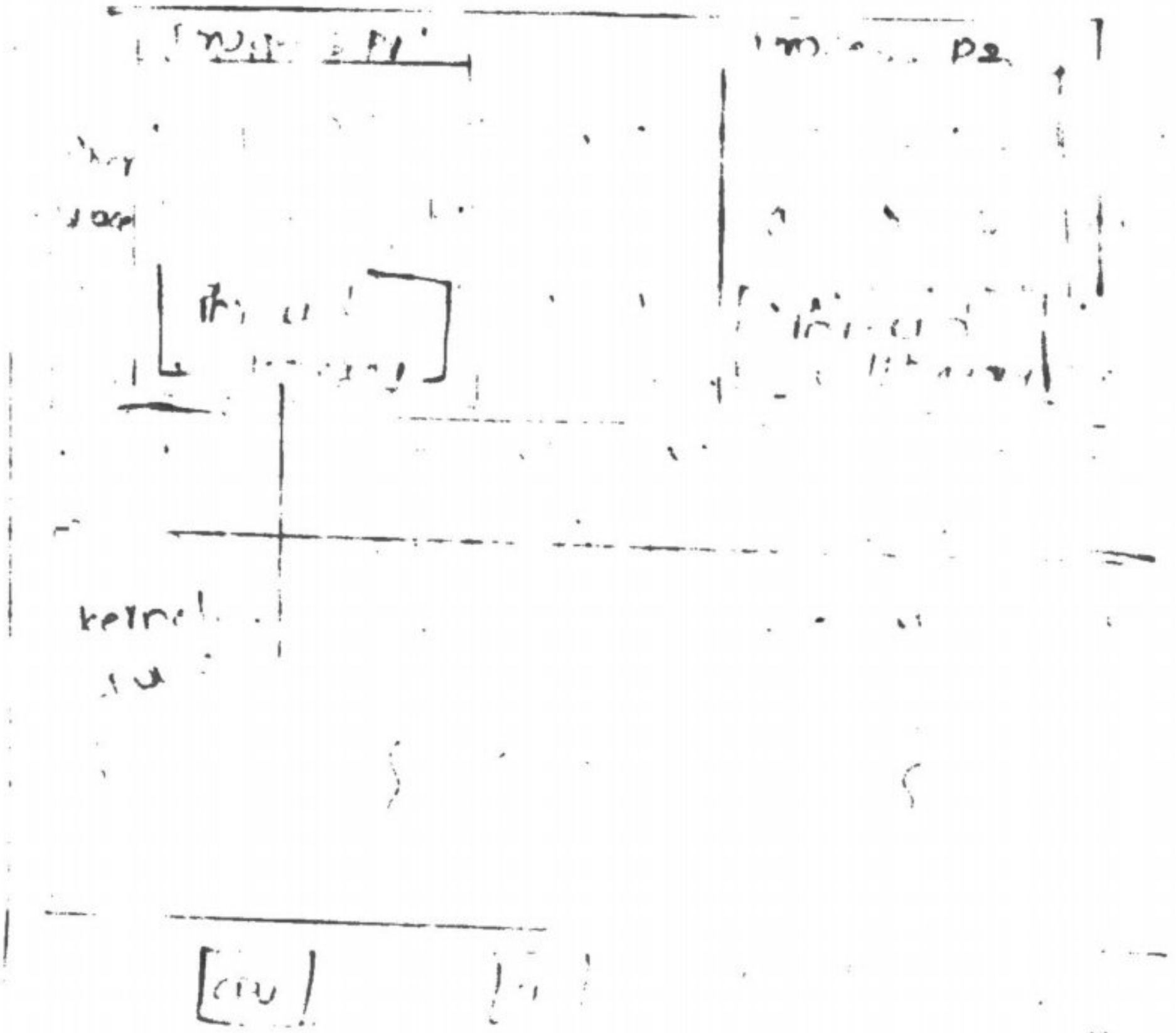
The many-to-many model multiplexes any number of users threads. The following diagram shows the many-to-many threading model where 6 user level thread are multiplexing with 6 kernal level threads in this model, developers can create as many user threads as necessary and the corresponding kernal threads can run in parallel on a multi processor machine. This model provides the best accuracy on concurrency and when a thread performe a blocking system, call the kernal can schedule another thread for execution.



Many to one Model

Many-to-one model maps many user level threads to one kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. only one thread can access the kernal at a time, so multiple threads are unable to run

in parallel on multi processors. If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the kernel threads use the many to one relationship models.



one-to-one model :-

There is one to one relationship of user level threads to the kernel level thread. This model provides more concurrency than the many to one model. It also allows another thread to run when a threads to execute in parallel on microprocessors. Disadvantage of this model is that creating user thread requires the corresponding kernal thread.

os/2 windows NT and windows 2000 use one to one relationship model.

* scheduling is of two types :-

1) Preemptive scheduling :-

It is a cpu scheduling technique that works by dividing time slots of cpu to a given process.

2) Non-preemptive scheduling :-

it is a cpu scheduling technique the process takes the resource and holds it till the process gets terminated or is pushed to the waiting state.

Scheduling Queues :-

* All processes upon entering into the system, are stored in the Job Queue. process in the ready state are placed in Ready Queue. Process waiting for a device to become available are Placed in Device Queues.

* A new process is initially put in the Ready Queue. It waits in a ready Queue until it is selected for execution.

* once the process is assigned to CPU, one of the following events may occur.

* The process could issue an I/O request, and then be placed in the I/O queue.

* The process could create a new sub process and waits for its termination.

* The process could be removed forcibly from the CPU, as a result of interrupt, and put back in the ready queue.

Schedulers :-

Three types of schedulers are present :

1. Long term scheduler :- The long term scheduler or job scheduler selects processes from this pool and loads them into memory for execution.

2. Short term scheduler :- The short term scheduler, or CPU scheduler selects from among the processes that are ready to execute and allocates the CPU one of them.

3. Middle term scheduler :- This scheduler removes the processes from memory and thus reduces the degree of multiprogramming. At some later time the process can be reintroduced into the memory and its execution can be continued where it left off. This is called swapping. The process is swapped out, and it is later swapped in, by the medium term scheduler.

context switch :-

- switching the cpu to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.
- When a context switch occurs, the kernal saves the context of the old process in the PCB and loads the saved context of the new process scheduled to run.
- context-switch time is pure overhead, because the system does no useful work while switching.
- Its speed varies from machine, depending on the memory speed, the number of registers that must be copied and the existence of special instructions.

what is cpu scheduling ?

cpu scheduling is a process of determining which process will own cpu for excuction while another process is on hold. The main task of cpu scheduling is to make sure that whenever the cpu remains idle, the os atleast select one of the processes available in the ready queue for excuetion. The selection process will be carried out by the cpu Scheduler. It selects the one of the processes in memory that are ready for excuetion.

[cpu scheduling]

[pre-emptive] | [Non-pre-emptive]

pre-emptive scheduling :-

In pre-emptive scheduling the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-pre-emptive scheduling :-

In this type of scheduling method, the cpu has been allocated to a specific process. The process that keeps the cpu busy will release the cpu either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like pre-emptive scheduling.

When scheduling is pre-emptive or Non-pre-emptive ?

To determine if scheduling is pre-emptive or non-pre-emptive, consider these four parameters :-

1) process switches from the running to the waiting state.

2) specific process switches from the running state to ready state.

3) specific process switches from the waiting state to ready state.

4) process finished its execution and terminated

only conditions 1 and 4 apply the scheduling is called non-pre-emptive.

Dispatcher :-

The dispatcher is the module that carries any information from run to running state. this function involves the following

1. switching context.
2. switching to user mode.
3. Jumping to the proper location in the user program to restart that program.

Scheduling Criteria :-

The criteria include the following

cpu utilization : It is the process of reducing the ideal time of the cpu during process execution.

cpu Throughput : It is the process of calculating of number of processors successfully executed under the processor. cpu throughout range from 0-100.

- Turn around time : It is the process to find out difference between arrival time and finishing time.

Formulas :-

Turn around time (TA) = finishing time - arrival time

$$\text{Total turn around time} = \frac{(P1 * TA) + (P2 * TA) + (P3 * TA) + \dots + (Pn * TA)}{\text{No. of processor}}$$

- waiting time :- It is the process to find out difference between starting time and arrival time.

Formulas :-

waiting Time (WT) = Starting time - Arrival time

Total waiting Time = $\frac{(P1 * WT) + (P2 * WT) + (P3 * WT) + \ldots + (Pn * WT)}{\text{No. of processors.}}$

Formula for Response time:-

Response Time (RT) = $\frac{\text{First Response time} - \text{arrival time}}{}$

Total response time = $\frac{(P1 * RT) + (P2 * RT) + (P3 * RT) + \ldots + (Pn * RT)}{\text{NO. of processors.}}$

Response Time: It is the process to find out difference between response time and arrival time.

scheduling algorithms :-

Different scheduling algorithms are :-

1. FCFS Algorithm
2. SJF (shortest job first).
3. Algorithm. priority Algorithm
4. Round Robin Algorithm:

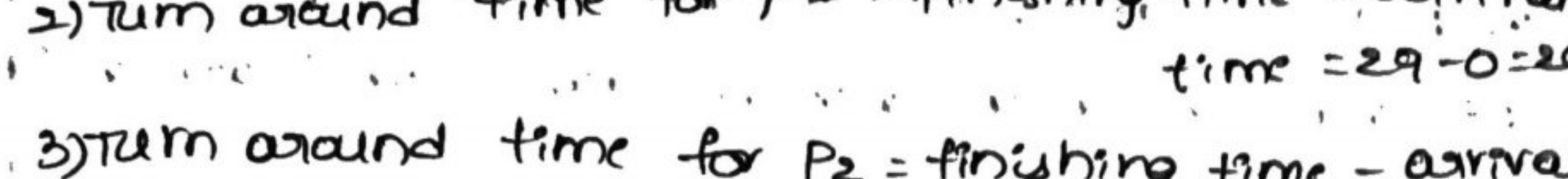
1. FCFS Algorithm :-

1. Start the process
2. Declare the array
3. Declare the number of processes
4. Input the processes along with burst time.
5. for each and every process calculate turn around time and waiting time and total turnaround time.
6. Display the results.

Example :-

Process	Burst time
P0	5
P1	24
P2	16
P3	10
P4	3

* First, prepare a Gantt chart for given problem.



* Calculate turnaround for each process :-

1) Turn around time for P_0 = finishing time - arrival time
= 5-0=5

2) Turn around time for P1 = finishing time - arrival time = 29-0=29.

3) Turn around time for P_2 = finishing time - arrival time = 45-0=45

4) Turn around time for P_3 = finishing time - arrival time = 55-0=55

5) Turn around time for P_4 = finishing time - arrival time = 58-0=58.

Total Turn around time = $\frac{(P_0 * TA) + (P1 * TA) + (P2 * TA) + (P3 * TA) + (P4 * TA)}{\text{Number of processors}}$

$= \frac{5+29+45+55+58}{5}$

$= 38.4$ milli sec

* Calculate waiting time for each process :-

1) waiting time for P_0 = Starting time - arrival time
= 0-0=0

2) waiting time for P1 = Starting time - arrival time
= 5-0=5

3) waiting time for P_2 = Starting time - arrival time
= 29-0=29

4) waiting time for P_3 = Starting time - arrival time
= 45-0=45

5) waiting time for P_4 = Starting time - arrival time
= 55-0=55.

problem :-

+3) 58.

ival time
= 5-0=5
arrival
29-0=29.
- arrival
15-0=45
me - arrival
-0=55
e - arrival
-0=58.

TA) + (P2 * TA)
(P4 * TA)
processes

process :-

arrival time

. arrival time

- arrival time

- arrival time

time - arrival time

5.

• Total waiting time $= \frac{(P_0 * WT) + (P_1 * WT) + (P_2 * WT) + (P_3 * WT) + (P_4 * WT)}{\text{Number of processors.}}$

* calculate the Response time for each process :-

1) Response time for P_0 = first response time - arrival time
= 0 - 0 = 0

2) Response time for P_1 = first response time - arrival time
= 5 - 0 = 5

3) Response time for P_2 = C.T - AT = 29 - 0 = 29

4) Response time for P_3 = C.T - AT = 45 - 0 = 45

5) Response time for P_4 = C.T - AT = 55 - 0 = 55

• Total Response time $= \frac{(P_0 * RT) + (P_1 * RT) + (P_2 * RT) + (P_3 * RT) + (P_4 * RT)}{\text{Number of processors.}}$

$= \frac{(0 + 5 + 29 + 45 + 55)}{5} = 26.4$ milli sec

SJF Algorithm :-

* SJF Stands for shortest Job first.
* It is a non-pre-emptive scheduling algorithm.
* Non-pre-emptive means any process successfully excueted under the processor without any time duration.

Main Algorithm for SJF :-

1. Start the process.
2. Declare the array.
3. Declare the Number of processors.
4. Input the processes along with the Burst time.
5. Sort the Burst time in increasing order.
6. Calculate or to find Out turn around time, waiting time and response time (using some mathematical formulas are :-

Turn around time = finishing time - arrival time.

waiting time = Starting time − Arrival time.

Response time = first response time − arrival time.

* Finally to calculate total average turn around time, total average waiting time and total average response time.

1). Total average turn around time :-

$$= \frac{(P1 * TA) + (P2 * TA) + (P3 * TA) + \dots + (Pn * TA)}{\text{No. of processors.}}$$

2) Total average waiting time :-

$$= \frac{(P1 * WT) + (P2 * WT) + (P3 * WT) + \dots + (Pn * WT)}{\text{No. of processors.}}$$

3) Total average response time :-

$$= \frac{(P1 * RT) + (P2 * RT) + (P3 * RT) + \dots + (Pn * RT)}{\text{No. of processors.}}$$

* Display the resultant value and stop the processors.

Ex :-

Process	Burst Time
P_1	5
P_2	24
P_3	16
P_4	10
1st ← P_5	(3) → least

: First we have to prepare a Gantt chart.

According to SJF to select the least Burst time and also select the processes and response of execution it is a non-pre-emptive algorithm

P_5	P_1	P_4	P_3	P_2
0	3	8	18	34
0+3	3+5	8+10	18+16	34+24

* To find out turn around time for each and every process :-

Turn around time P_1 = finishing time - Arrival time

$= 8 - 0 = 8$

Turn around time $P_2 = 58 - 0 = 58$

Turn around time $P_3 = 34 - 0 = 34$

Turn around time for $P_4 = 18 - 0 = 18$.

Turn around time for $P_5 = 13 - 0 = 13$.

∴ Total average turn around time :-

$$= \frac{(P_1 * TA) + (P_2 * TA) + (P_3 * TA) + (P_4 * TA) + (P_5 * TA)}{\text{No. of processors}}$$

$$= \frac{8 + 58 + 34 + 18 + 3}{5} = 24.2 \text{ milli sec}$$

* To find out waiting time for each and every process :-

waiting time for P_1 = starting time - arrival time

$= 3 - 0 = 3$

waiting time for $P_2 = ST - AT = 34 - 0 = 34$

waiting time for $P_3 = ST - AT = 18 - 0 = 18$

waiting time for $P_4 = ST - AT = 8 - 0 = 8$

waiting time for $P_5 = ST - AT = 3 - 0 = 3$

Total : Average waiting time :-

$$= \frac{(P1 * WT) + (P_2 * WT) + (P_3 * WT) + (P_4 * WT) + (P_5 * WT)}{\text{No. of processors}}$$

$$= \frac{3 + 34 + 18 + 8 + 3}{5}$$

$= 12.6$ milli sec

* To find out Response time for each & every process :-

Response time for P_1 = First response time - arrival time

$= 3 - 0 = 3$

Response time for $P_2 = 34 - 0 = 34$

Response time for P_3 : 18 - 0 : 18

Response time for P_4 : 8 - 0 : 8

Response time for P_5 : 0 - 0 0

Total average Response time :-

$$\frac{(P_1 * RT) + (P_2 * RT) + (P_3 * RT) + (P_4 * RT) + (P_5 * RT)}{\text{No: of processes.}}$$

$$= \frac{2 + 34 + 18 + 8 + 0}{6\ 5} = \frac{63}{5} = 12.6 \text{ milli sec}$$

Priority Algorithm :-

* It is a non-pre-emptive scheduling algorithm
* Sometimes priority algorithm consider as a pre-emptive algorithm.
* Any process successfully excueted under the processor within the time duration.
* Non-pre-emptive means any process success fully Excueted under the processor without any time duration.

Algorithm :-

1. Start the process.
2. Declare the array.
3. Declare the number of processes.
4. Input the processes, along with their priori
5. Sort the priority in increasing order.
6. To find out the turn around time, waiting time and Response time (using mathematical processes are :-

1. Turn around time = finishing time - arrival tim
2. Waiting time : waiting time - Arrival time
3. Response time : Response time - Arrival time

Finally we calculate TATAT, TAWT, TART

1. Total Average turn around time :-

$$= \frac{(P_1 * TA) + (P_2 * TA) + (P_3 * TA) + (P_4 * TA) \ldots + (P_n * TA)}{\text{No. of processors.}}$$

2. Total Average waiting time :-

$$= \frac{(P_1 * WT) + (P_2 * WT) + (P_3 * WT) \ldots + (P_n * PWT)}{\text{No. of processors.}}$$

3. Total Average Response time :-

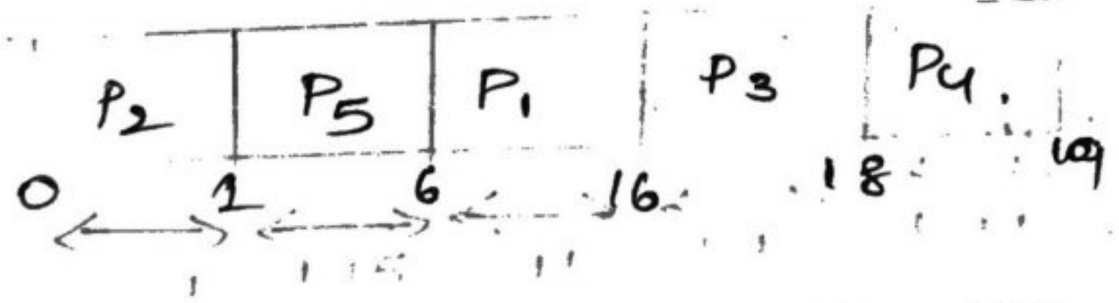
$$= \frac{(P_1 * RT) + (P_2 * RT) + (P_3 * RT) \ldots + (P_n * RT)}{\text{No. of processors.}}$$

* Finally display the value and stop the processors.

Ex :-

process	Burst Time	Priority.
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

First we have to prepare a Gantt chart :-



Turn around time $P_1 = FT - AT = 16 - 0 = 16$

Turn around time $P_2 = FT - AT = 1 - 0 = 1$

Turn around time $P_3 = FT - AT = 18 - 0 = 18$

Turn around time $P_4 = 19 - 0 = 19$

Turn around time $P_5 = 6 - 0 = 6$.

total Turn around time $= \frac{16 + 1 + 18 + 19 + 6}{5} = \frac{60}{5} = 12$ millisec.

waiting time P_1 = Starting time - Arrival time $= 6 - 0 = 6$

waiting time $P_2 = ST - AT = 0 - 0 = 0$

waiting time $P_3 = 16 - 0 = 16$

waiting time $P_4 = 18 - 0 = 18$

waiting time $P_5 = 1 - 0 = 1$

Total Average waiting time $= \frac{16 + 0 + 18 + 16}{5} = \frac{41}{5} = 8.2$ milli sec //

Response time for $P_1 = F.R.T - A.T = 6 - 0 = 6$

Response time for $P_2 = 0 - 0 = 0$

Response time for $P_3 = 16 - 0 = 16$

Response time for $P_4 = 18 - 0 = 18$

Response time for $P_5 = 1 - 0 = 1$

∴ Total Average Response time :-

$$\frac{(P_0 * RT) + (P_1 * RT) + (P_2 * RT) + (P_3 * RT) + (P_4 * RT) + (P_5 * RT)}{5}$$

$$= \frac{41}{5} = 8.2 \text{ milli sec} //$$

Round Robbin algorithm :-

* It is a pre-emptive algorithm.
* pre-emptive means time, Quantum (or) time slice of the excuetion.
* Time slice means each and every process allocated to time = duration.
* Main algorithm :-

1. start the process.
2. Declare the array.
3. Declare the Number of processors.
4. Input the process along with their Burst time and time slice.
5. If Burst time less than Eqeal to time slice then process successfully excueted under the processors.
6. If Burst time greater than time slide then any process not successfully excueted under the Processors.

7. To find (or) to calculate turn around time, waiting time and reponse time by using different Mathematical formulas give :-

1) Turn around time = finishing time - Arrival time.

2) waiting time = starting time - Arrival time.

3) Reponse time = first response time - Arrival time

8. Finally to calculate total average turn around time, TAWT and TART

9. Display the resultants.

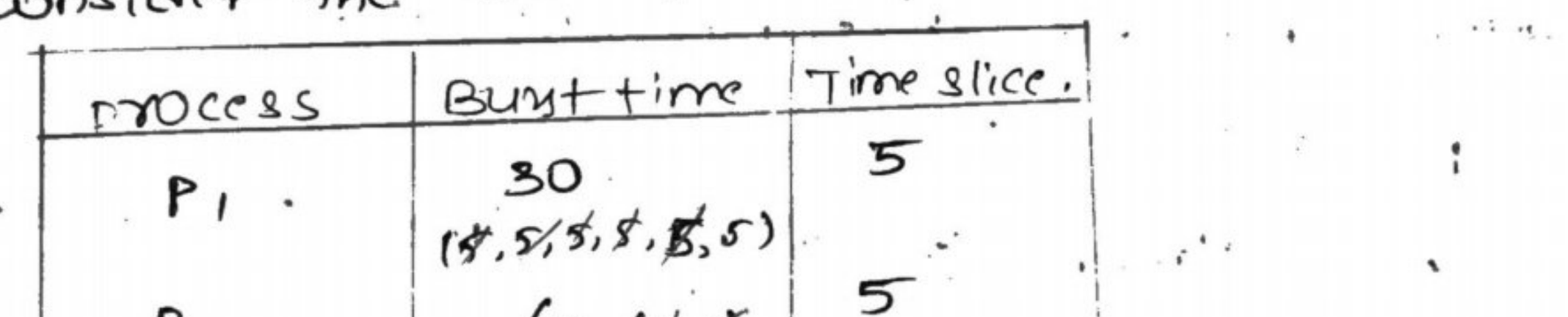
10. Stop the process.

Consider the Example :-

process	Burst time	Time slice.
P_1	30 (5, 5, 5, 5, 5, 5)	5
P_2	6 (5, 1)	5
P_3	8 (5, 3)	5

First we have to prepare a Gantt-chart.

for given process.



To find out turn around time for each & Every proces :-

Turn around time P_1 = FT - AT = 44 - 0 = 44.

Turn around time P_2 = FT - AT = 21 - 0 = 21

Turn around time P_3 = FT - AT = 24 - 0 = 24.

Total Average turn around time :-

$$= \frac{44 + 21 + 24}{3} = \frac{89}{3} = 29.6 \text{ milli sec.}$$

* To find out waiting time for each & every process :-

waiting time P_1 : Starting time - Arrival time

$= 0 - 0 + 15 - 5 + 24 - 20 = 14$

waiting time P_2 : Starting time - Arrival time

$= 5 - 0 + 20 - 10 = 15$

waiting time P_3 : Starting time - Arrival time

$= 10 - 0 + 21 - 15 = 16$

Total average waiting time :-

$= \frac{14 + 15 + 16}{3} = \frac{45}{3} = 15$ milli sec.

* To find out response time for each & every process :-

Response time P_1 = first response time - Arrival time

$= 0 - 0 = 0$

Response time P_2 = F.T - AT $= 5 - 0 = 5$

Response time P_3 = F.T - AT $= 10 - 0 = 10$

Total average response time :-

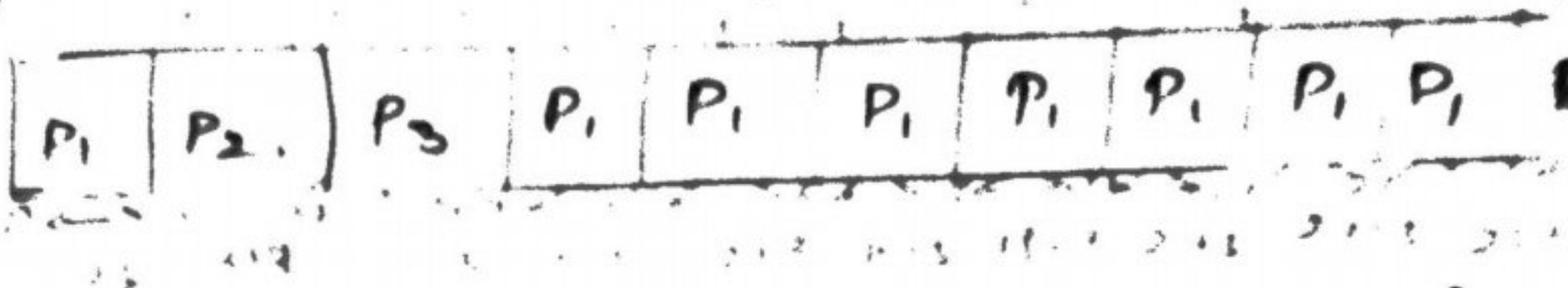
$= \frac{0 + 5 + 10}{3} = \frac{15}{3} = 5$ milli sec //.

* According to pre-emptive algorithm W.T and Response time are different.

(2) Consider another example.

Process	Burst time	Slice time
P_1	24	3
P_2	3	3
P_3	3	3

first we have to perform gantt chart for the processes.



To find out turn around time for each & every process :-

Turn around time P_1 = Finishing time - Arrival time
$= 30 - 0 = 30$

Turn around time P_2 = F.T - A.T $= 6 - 0 = 6$

Turn around time P_3 = F.T - A.T $= 9 - 0 = 9$

Total average turn around time :-

$= \frac{30+6+9}{3} = 15$ milli Sec.

To find out waiting time for each & every process :-

waiting time P_1 = Starting time - Arrival time
$= 0 - 0 + 9 - 3 = 6$

waiting time P_2 = S.T - A.T $= 3 - 0 = 3$.

waiting time P_3 = S.T - A.T $= 6 - 0 = 6$.

Total Average W.T = $\frac{6+3+6}{3} = 5$ milli Sec.

To find out Response time for each & Every Process :-

Response time P_1 = FRT - AT $= 0 - 0 = 0$

Response time P_2 = FRT - AT $= 3 - 0 = 3$

Response time P_3 = FRT - AT $= 6 - 0 = 6$.

Total average response time :-

$= \frac{9}{2} = 3$ milli sec.

(3) consider another example :-

Process	Burst time	slice time
P_1	30	4
P_2	8	4
P_3	8	4

P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1	P_1

To find out turn around time for each & every process :-

Turn around time P_1 = F.T − A.T = 46 − 0 = 46

Turn around time P_2 = F.T − A.T = 20 − 0 = 20

Turn around time P_3 = F.T − A.T = 24 − 0 = 24

Total Average turn around time :-

$$= \frac{46+20+24}{3} = \frac{90}{3} = 30 \text{ milli sec}$$

To find out waiting time for each & every process :-

waiting time P_1 = S.T − A.T = 0 + 0 + 12 − 4 + 24 − 16 = 8 + 8 = 16

waiting time P_2 = S.T − A.T = 4 − 0 + 16 − 4 = 4 + 12 = 16

waiting time P_3 = S.T − A.T = 8 − 0 + 20 − 12 = 8 + 8 = 16

Total Average waiting time :-

$$= \frac{16+16+16}{3} = \frac{48}{3} = 16 \text{ milliSec}$$

To find out Response time for each & even process :-

Response time P_1 = F.R.T − A.T = 0 − 0 = 0

Response time P_2 = F.R.T − A.T = 4 − 0 = 4

Response time P_3 = F.R.T − A.T = 8 − 0 = 8.

∴ Total Average Response time :-

$$= \frac{8+4+0}{3} = 4 \text{ milli Sec.}$$

critical section:

*critical section is a place.
*critical section supported to execute more than one process sucessfully under the processor.
*critical section supported to enter only one process at a time. and successfully completed first process. then allowed next process into critical section for the purpose of execution.
*critical section depends on three factors :-
i) Mutual Excuetion.
ii) progress
iii) Bound.

i) Mutual excuetion :-
* According to mutual excuetion property only allows one process into critical section for the purpose of excuetion. and wait for some next process from time for excuetion.

ii) Progress :-
* Progress intially maintained '0' Progress of the critical section.
* According to progress to wish any process into critical section for excuetion.
*one process successfully completed under the Critical section allows next process into critical section for excuetion.
* parallelly Excueted more than one process successfully under the critical section.

iii) Bound :-
* Bound property provides a limited (or) some time durations to the each & every process.
* Based on the time limit any process enter into critical section for the purpose of excuetion.
* successfully excueted more than one process

under the critical section based on bounded time (or) bounded process.

Peterson's solution :-

* It is a software approch.
* pertenson's solution is implemented with the help of using different conditions (or) criterias are

i) critical Section.
two Different
ii) Variables < flag
                TURN.

iii) Required two different processors (more than one process).

process names are :-

1) process P1
2) P2.

flag variable maintains two '0' (true) and '1' (false).

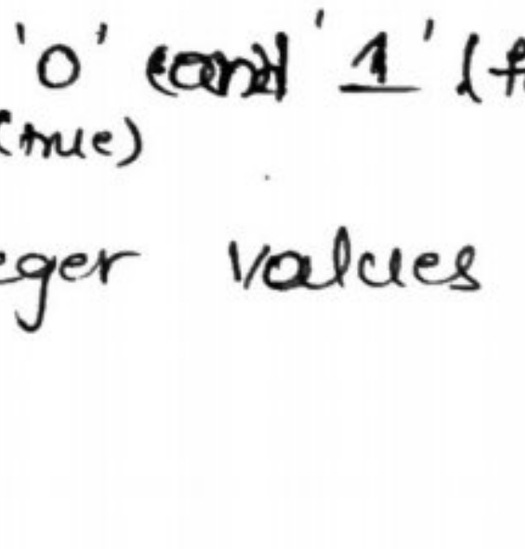
Turn has maintains two integer values are '0' and '1'.

procedure for P1 :-

1. Start the process P1.
2. Declare the variable flag (0) = True.
                        Turn (1) = false.
3. Set the flag(1) and turn = 1
4. P1 satisfies the above condition then enter p1 into critical section for execution.
5. when flag(0) = false then enter P2 remainder Section into critical section.
6. stop process.

Implementation code for P1 :-

```
void P1( )
{
while (true)
{
```



```
flag[0] = True;
turn = 1;
while (flag(1) && turn == 1) do nothing
 <P1 enter into critical section >
 Flag[0] = False;
}<Enter P2. remainder section into critical section>
}
```

Procedure for "P2"

1. start the process P2.
2. Declare the variable flag (1) = True.
   Turn(0) = false.
3. set the flag (0) and turn==0. enter.
4. P2 satisfies the above condition then, P2 into critical section for execution.
5. when flag(1) = false then enter P1 remainder Section into critical section.
6. stop process.

Implementation code for P2 :-

```
void P2()
{
while (true)
{
flag[1] = true;
 turn = 0;
 while(Flag[0] && turn == 0) do nothing.
  <P2 enter into critical section >
  Flag[1] = False;
 <Enter P1 remainder section into critical
                                  section>.
}
}
```

Interrupts :-

* Interrupt is a hardware technique.
* Interrupt means to discuss normal flow of execution temporarily under the critical section.
* Interrupt provides two different conditions to the user.
  - Enable condition
  - Disable condition.
* Interrupt technique is implemented using minimum two processor (P_1, P_2)
* Interrupt technique is also required critical section.
* According to interrupt technique P1 and P2 process successfully excuted under the critical section using disable and enable conditions.
* Intially disable condition applied on critical section for Entry of p1 process. into the critical section.
* Enable condition is used to entry of p2 process into remainder section into critical section.

Example Code :-

Procedure for P1 :-

```
Void P1 ()
{
while (true)
{
< Disable condition >
< P1 enter into critical section >
  < Enable condition >
  < P2 remainder section >
      enter into critical section >
```

}

}

Procedure for P_2 :-

```
void P2()
{
while(true)
{
<Disable condition>
<P2 enter into critical section>
  <Enable condition>
<P1 remainder section enter into critical
                                section>
}
}
```

Test set technique :-

* It is a hardware technique.
* Test set approch is always maintained boolean type.
* Boolean type means to maintaining two integer values are :- '0' and '1'.
* Test set approch is always maintain '0' value.
* properly implementing of test set approch required to take any one integer variable.
* Integer variable intially assigned to "ϕ". equal
* when integer value returns '1' at that time consider has true .(P1 process first section successfully excueted under C.S)
* when integer value return '0' at that time consider has false (P2 process is successfully excueted under remainder section)

```
program for test set
Boolean Testset (int i)
{
If (i==1) then
{
i=1 ;
return true ;
}
else
{
i=0 ;
return false ;
}
}
```

Exchange Method :-

* Exchange Method initially required two variable that are :-

1) Register
2) Memory (or) main memory.

* to exchanging of any information from register to main memory (or) main memory to register required another variable is "Temp"

* to properly exchanges from register to main memory and main memory to register any process information.

example :-

```
void exchange (int register, int main memory).
{
  int temp ;
  temp = main memory ;
  register = main memory ;
```

temp = register

}

producer consumer problem :-

* It is a hardware technique.
* producer consumer problem is implemented with the help of using different conditions are :-

1) wait (assigned to zero / n)
2) signal (assigned to one / s)
3) produce () 4) append () / add ()
5) Buffer (b)
   queue
6) in (Buffer Items). (6.1)
7) in >= 1 8) take () [8.1 > r = b [out]]
9) consume ().

* produce method takes a number of items.
* Append / add is used to adding of number of item into buffer queue.
* Buffer queue assign to 'b' type variable.
* Buffer queue is maintained number of items into one order.
* Buffer queue supported to more Number of items.
* Buffer items arranged into array type.
* Buffer items are represented in "In" type.
* total Buffer items are represented in the form of In = b [in].
* consumer retriving of any item values from the buffer queue initially to maintain intial condition In >= 1.
* Above condition satisfies retriving of the buffer queue using take() method.

* Retriving items are arranged into the term of V = b[out]
* Finally recieving from the buffer queue using consumer method.

Example :-

program :-

```
Producer - consumer structure
{
int n=0;
in s=1;
int in, out;
int b;
int w, v;
};
```

Procedure for producer :-

```
Void producer()
{
 produce();
 append();
  w1 = b[in];
  in--;
}
}
```

→ while (true){

procedure for consumer :-

```
 Void consumer()
{
while (true){
If (in ≥ 1) then
Take()
V = b[out]
out++;
consume()
}
}
```

Semaphores :-

* semaphore is a hardware technique.
* semaphore is maintained integer values are 0 to 9 (sometimes semaphore is called Non-Negative value).
* semaphore supported two operation values :
  1. wait 2. signal.
* wait operation implemented. 'P' operation().
Sometimes wait operation specified another names are decrease(), down(), sleep().
* signal operation implemented as 'V' operation()
Sometimes signal operation specified another names are, increase(), up(), wake-up().
* semaphore is implemented with the help of using different conditions are :-
  1. Ready queue.
  2. Counting semaphore.
  3. Binary semaphore.
  4. critical section.
  5. Minimum two processes (or) More than one process.
* semaphore is again divided into two types :-
  1. Binary semaphore.
  2. Counting semaphore.

1. Binary semaphore :-

* Binary semaphore shortly denoted as "BS".
=> Binary semaphore supported two values are "0" & "1".
=> Binary semaphore is implemented two modes are : 1. wait 2. Signal.

Implementation of wait operation :-

```
void semaphore (Binary semaphore BS)
{
void wait (Binary Semaphore BS)
{
while (true)
{
If (Bs value ==1) then
 Bs Value =0
< process enter into critical section >
else
BS value = 1
< process enter into ready queue and block
    the process >
}
}
}.
```

Implementation of signal operation :-

```
void signal (Binary Semaphore BS)
{
while (true)
{
If (Bs queue == Empty) then
{
 Bs queue = 0
process enter into C.S &
<process successfully excueted>
else
 BS Queue = 1
```

<Remove old process from ready queue &
insert new process into Ready queue>

}
}
}.

Counting Semaphores :-

*counting semaphore is a hardware technique.
*counting semaphore supported two conditional variables are :-
1) ~~Critical section~~ counting semaphore wait.
2) counting semaphore signal.
*counting semaphore is implemented based on count -ing variable and queue.
*counting semaphore is mainly applied on processor and resources.

Implementation of wait operation :-

```
Structure semaphore (counting Semaphore cs).
{
  int count;
  Queue type Queue;
};
Void cs wait (counting Semaphore cs)
{
while (true)
{
cs count ++;
If (cs count < 0) then
{
<process enter into queue and block the
process>
}
}
```

To :implementation of signal operation :-

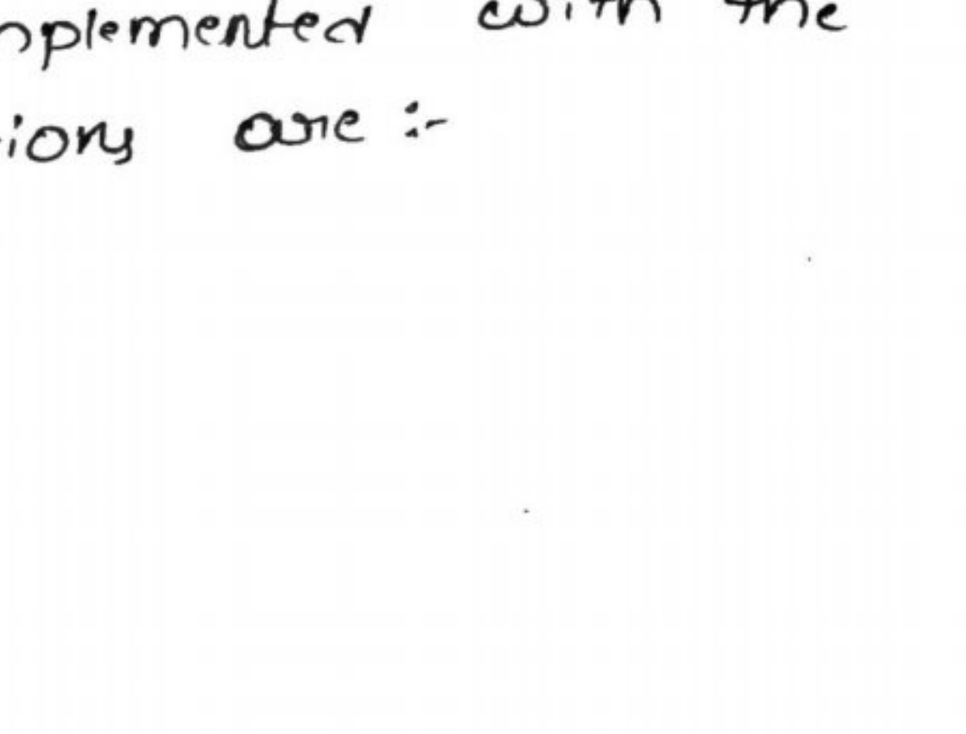
```
void cs signal (counting semaphore cs)
{
while (true)
{
cs count - - ;
If (cs  count ≤ 0) then
 {
 Remove old process from queue & insert new process
< process enter into queue and block the process's>
                                        into queue .
else
<proces  executed  successfully  under  the  CS>
 }
}
}
```

Monitor :-

* Monitor is a software technique.
* Monitor technique is implemented with the help of different conditions are :-

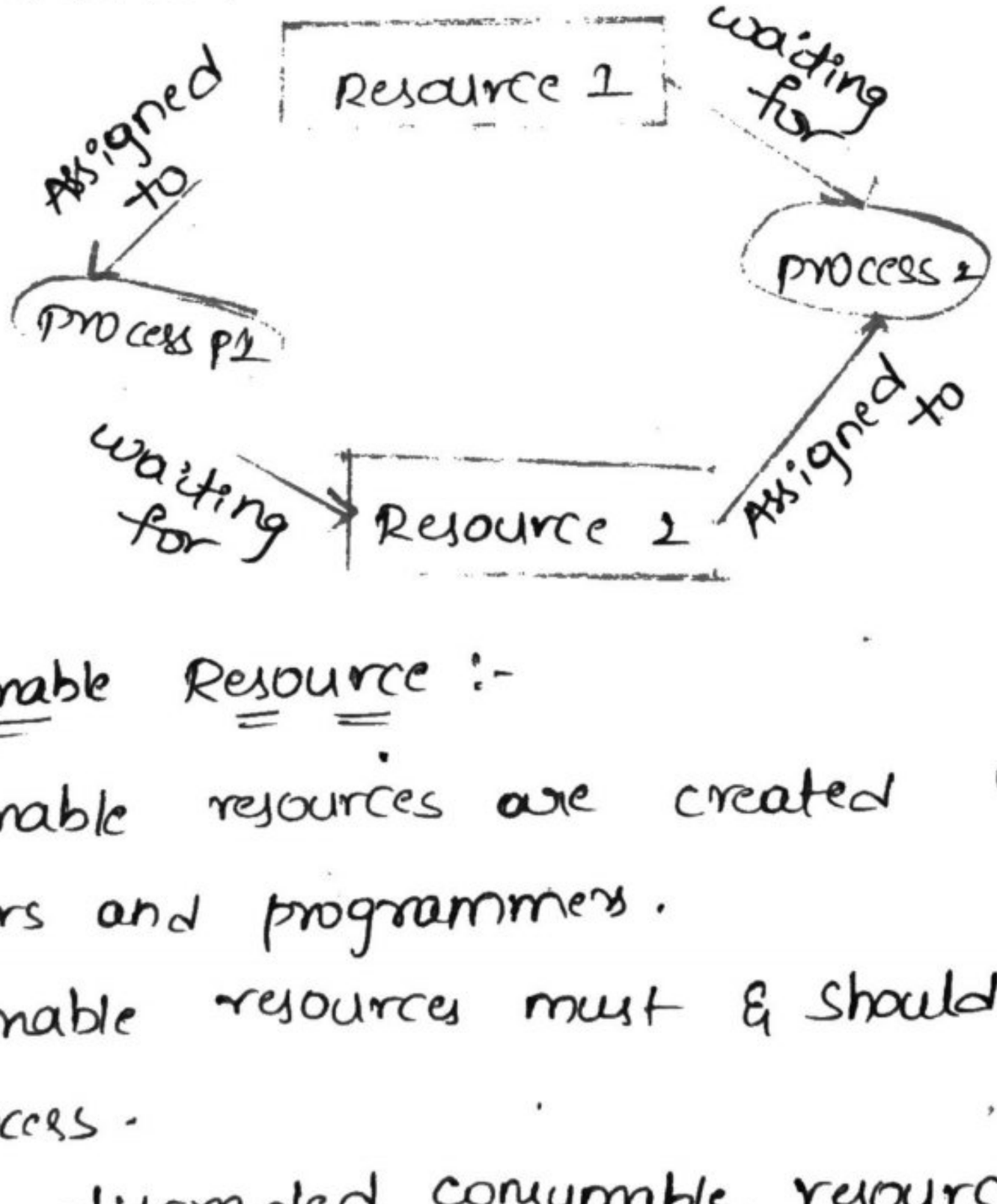
1) Ready queue.
2) Entry Exit
3) Local variables
4) procedures.
5) Intial Conditions.
6) Control wait() / conditional wait().
7) control signal () / conditional signal ().
8) Blocked queue.
9) Urgent queue.
10) Entry exit / out exit.

Monitor →



Dead Lock :-

* Dead lock is a Situation where a set of processes are blocked because each process is holding a resource and waiting for another resource required by some other process.
* Consider a example when two trains are coming toward each other on the same track and there is only one track none of the trains can move once they are in front of Each other.
* A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for the resources held by other (s)
* For example in the below diagram, process 1 is holding resource 1 and waiting for resource 2 which is acquired by the process 1 and process 2 is waiting for resource 1.



1. Consumable Resource :-

* consumable resources are created with the help of users and programmers.
* Consumable resources must & should needed to use for process.
* Finally discraded consumable resources from private area.
* Sometimes consumable resources overloaded into main memory or secondary memory.

2. Reusable Resources :-

* Reusable resources are created with help of using developers. Reusable resources shared by different users.

* Reusable resources are available in main (or) secondary memory.

Dead lock conditions / Dead lock preventions :-

* To properly maintaining of dead locks required to follow the 4 conditions are :-

1. Mutual Exclusion
2. Hold and wait.
3. No preemption
4. circular wait

* sometimes these 4 conditions are called dead lock preventions.

1. Mutual Exclusion :-

* Intially consider two processes one common resource. process names are p1 & p2 and common resource name is 'p'.

* p1 enters into critical section for the purpose of execution.

* p1 properly is used common resource & successfully execution under the critical section.

* process p1 execution time does not allow any other process into critical section.

* process p2 ~~execution time~~ does enter into critical section for purpose of execution.

* P2 properly is used common resource & successfully executed under critical section.

process P2 excueted time does not allow any other process into critical section.

conclusion :-

1. Mutual exclusion properly does not support more no. of process & resources.

2. To overcome this drawback we are using hold and wait condition.

2. Hold and wait :-

* According to Hold & wait, consider two proces P1, P2 & resources R1 & R2.

* Hold & wait is implemented based on any proces request waiting for one resource & hold another resources.

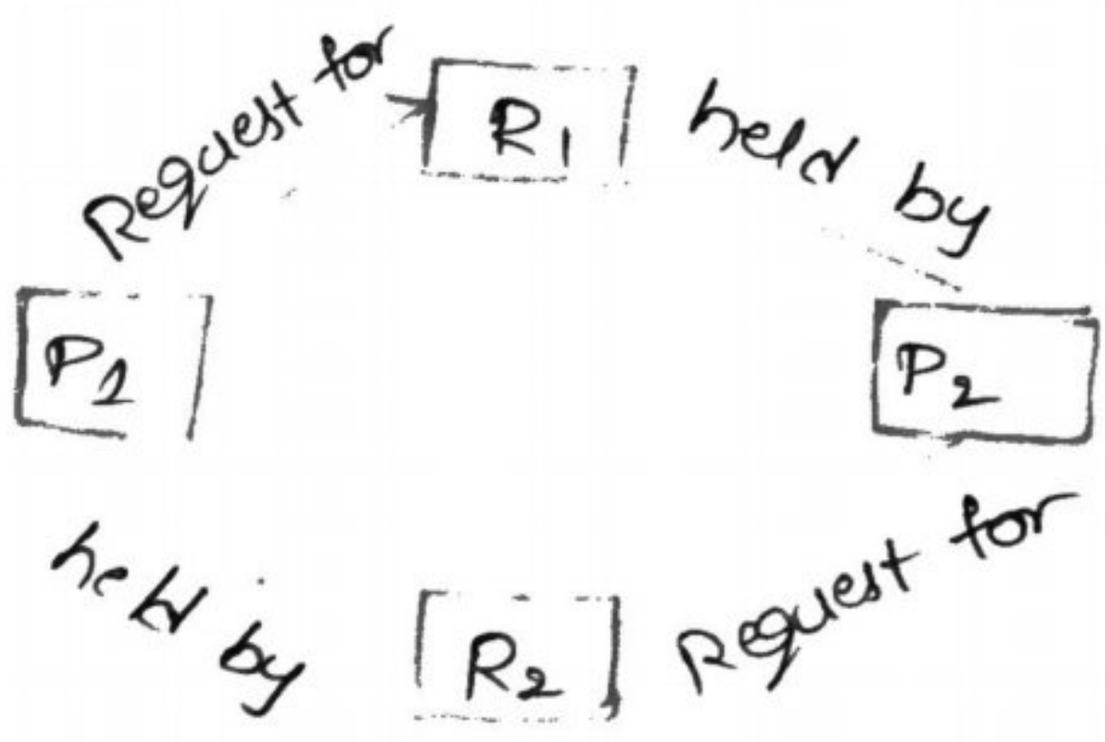
* P1 sends send a request to R1 resource and already held by R2 resource.

* Finally process P1 successfully completed under the critical section.

* P2 process Enter into the critical section for the purpose of excution (R1 & R2 resources required).

* P2 process Send a request to R2 resource and already held by R1 resource.

* Finally process P2 successfully completed under the critical section.



3. No preemption :-

* Intially consider more than one process & more than one resource.

* According to no preemption once enter any process into critical section must & should complete under critical section.

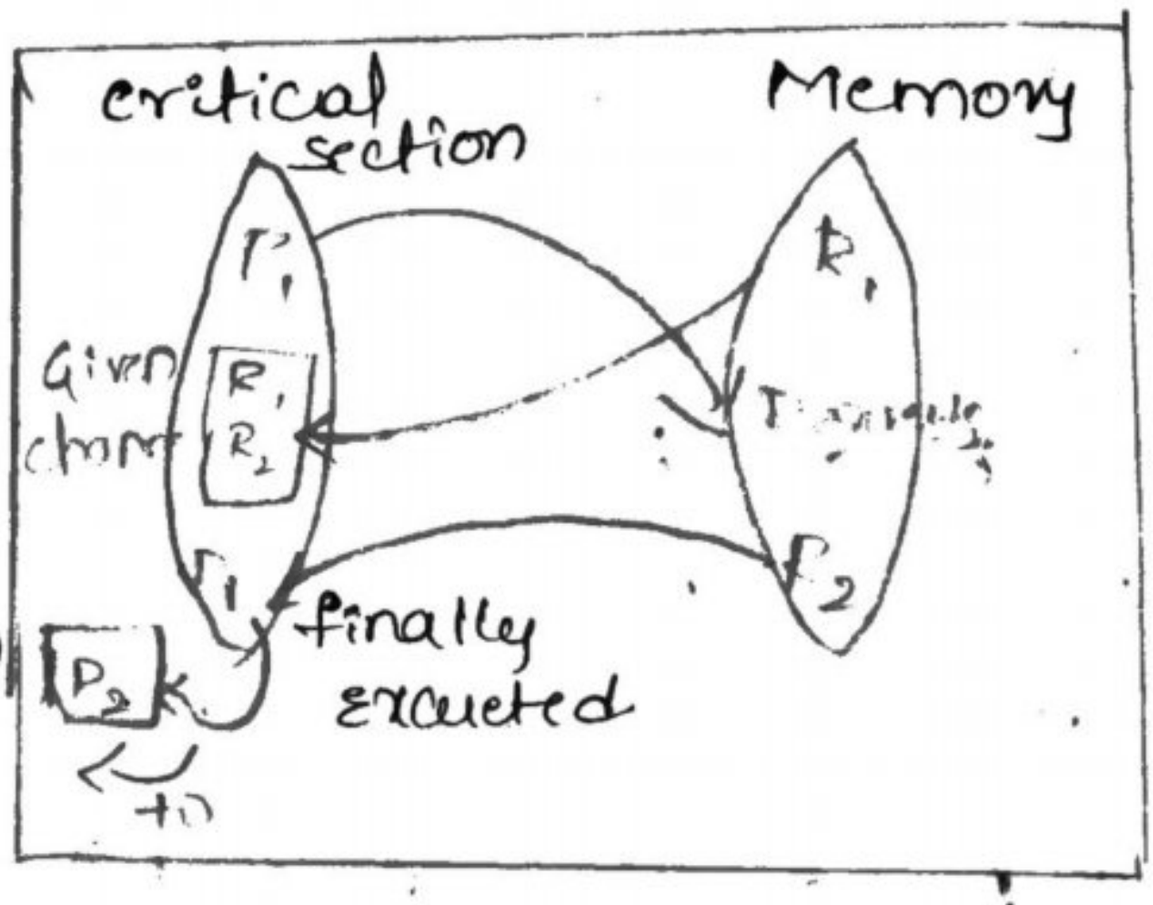
* Any process excuetion time does not allow any another process into critical section.

Example :-

* Consider two process P1, P2 and resource R1, R2.

Conclusion :-

* P1 excueted time does not allow P2 into critical section (process P2 waits for some time)



4. Circular wait :-

* According to circular wait consider more number of processors & More number of resources.

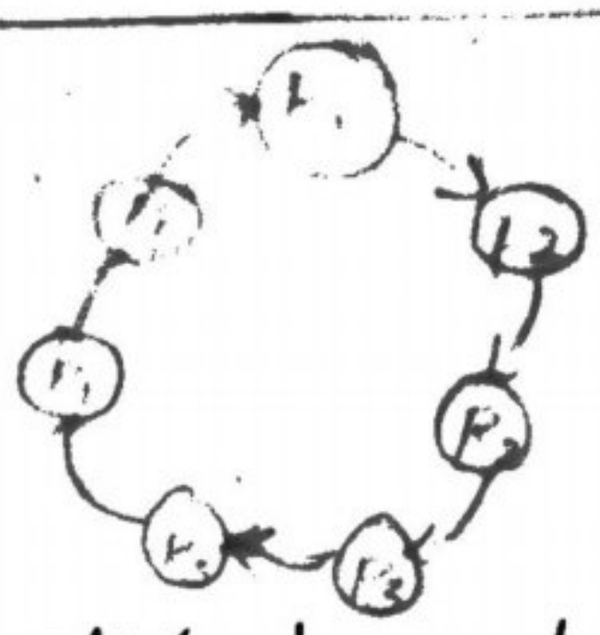
* Circular wait is implemented with the help of using two protocols are;

1. F(Ri) > F(Rj) 2. F(Rj) > F(Rj)

1. F(Ri) > F(Rj) :-

* process 'p' sends a request for resource Rj & instantly processor provides resource Ri (Resour Ri has maintained highest priority).

2. F(Rj) > F(Ri)



* process 'p' sends a request for resource Ri & instantly processor Provided resource Rj.
* Sometime, circular what is implemented based on circular queue.
* circular queue is always maintained starting and ending points are same.

Dead lock Avoidance :-

* In dead lock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause dead lock in the system.
* The state of the system will continuously be checked for safe and unsafe states.
* In order to avoid dead locks, the process must tell OS, the maximum number of resources a process can request to complete its execution.
* The simplest and most useful approch states that the process should declare the maximum number of resources of each type it may ever need.
* The deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

* Banker's algorithm / safety algorithm / Dead lock avoidance algorithm :-

1. Initially consider a available / work and finish vector available / work represented as 'm' and finish represented as 'n'. finish [i] = 1...n and finish[i] = False ;

2. find an 'i' such that :-
   - Finish [i] : false ;
   - Need [i] <= work ;
   if 'i' exists go to step 3.
   else go to step 4.
3. Work = work + Allocation
   goto Step 2.
4. Finish [i] = true for all processes.

Request Resource Algorithm :-
1. Request [i] <= need [i]
   goto step 2 otherwise error condition.
2. Request [i] <= Available work
   goto step 3 otherwise wait for shorter/ longer time.
3. Available = Available - Request.
   Allocation = Allocation + Request
   Need [i] = Need [i] - Request.

Problem :-

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

and also A has 10 instances,
B has 5 instances.
C has 7 instances.

* First we need to find out the work vector.

work vector :-
A = A instance - Total allocation of 'A'
= 10 - (0+2+3+2+0) = 10 - 7 = 3.

Work vector :-

A = A instance – Total allocation of 'A'.

= 10 - (0+2+3+2+0) = 10-7 = 3.

B = B instance – Total allocation of 'B'.

= 5 - (1+1) = 5-2 = 3.

C = C instance – Total allocation values of 'C'.

= 7 - (2+1+2) = 7-5 = 2.

* work vector = [3 3 2]

*To find out the need vector using formula.

Need vector = Max allocation – Allocation.

Need vector = $\begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$.

* Needed to check for P_0 condition is : Need [i] <= work.

* for P_0 : [7 4 3] <= [3 3 2] (Not safe)

* for P_1 : [1 2 2] <= [3 3 2] (safe)

since P1 are work safe we proceed as :-

work = work + Allocation.

for P1 : work = work + Allocation.

= [3 3 2] + [2 0 0]

= [5 3 2].

* Needed to check for P_2 : Need [i] <= work for

P_2 : [6 0 0] <= [5 3 2] (Not safe).

* Needed to check for P_3 : Need [i] <= work.

For P_3 : [0 1 1] <= [5 3 2] (safe).

Since P_3 is safe we proceed Oy.

For P_3 : work = work + Allocation.

= [5 3 2] + [2 1 1]

= [7 4 3].

* Needed to check for P_4: Need [i] <= work.

For P_4: [4 3 1] <= [7 4 3] (safe)

Since P_4 is safe we proceed ons:-

For P_4: work = work + Allocation = [7 4 3] + [0 0 2]

= [7 4 5]

* Again do the same process for the 'Not safe' processes.

Need to checle for 'P_0': Need [i] <= work

[7 4 3] <= [7 4 5] (safe).

Since P_0 : worle = work + Allocation

= [7 4 5] + [0 1 0]

= [7 5 5]

* Need to check for P_2 : Need [1] < = work

[6 0 0] < = [7 5 5] (safe)

Since 'P_2 is safe we proceed as :-

For P_2 : work = work + allocation = [7 5 5] + [3 0 2]

= [10 5 7]

safe order : P_1 P_3 P_4 P_0 P_2.

4) ~~Con~~sider a dead locle.

Dead Lock prevention Algorithm :-
Detection

Intially consider work on Available vector..

is represented as "m" finish Vector is represented by 'n'.

1. Finish [i] = 1 . . . n
2. Finish [i] = False.

allocation [i] ≠ 0

3. Find of i such that request [i] = worle.

If i exist goto step 4, else goto Step 5.

4. work = Work + Allocation.

5. Finish [i] = False for all processes.

Problem :-

Process	Request			Allocation		
	A	B	C	A	B	C
P_0	0	0	0	0	1	0
P_1	2	0	2	2	0	0
P_2	0	0	0	3	0	3
P_3	1	0	0	2	1	1
P_4	0	0	2	0	0	2

and also A has 7 instancey
B has 2 instances.
C has 6 instances

Sol:- First we need to find out the work vector.

Work vector :-

A = A instancey - Total Allocation of 'A'.

= 7 - 7 = 0

B = B instances - Total Allocation of B

= 2 - (1+1) = 2 - 2 = 0

C = C instances - Total Allocations of C.

= 6 - (3 + 1 + 2) = 6 - 6 = 0.

∴ Work Vector = [0, 0, 0]

* Needed to check for P_0 condition is :
Need [i] <= work.

* To find out work Vector : P_0 :-

[0 0 0] ≤ [0 0 0]

Above condition satisfies P_0 safe.

To find out for P_0.

work = work + allocation for P_0.

$= [0\ 0\ 0] + [0\ 1\ 0]$

$= [0\ 1\ 0]$

Need to apply new work on 'p'.

P_1 request [i] ≤ work.

$[2\ 0\ 2] \leq [0\ 0\ 0]$

Above condition not satisfies it is unsafe,

check for P_2.

P_2 request [i] ≤ work.

$[0\ 0\ 0] \leq [0\ 0\ 0]$

To find out for P_2

work = Work + Allocation.

work = [0 0 0] + [3 0 3]

Work = [3 0 3]

check for P_3

P_3 request [i] ≤ work.

$[1\ 0\ 0] \leq [0\ 0\ 0]$

To find out for P_3

Work = Work + Allocation

It is not satisfies P_3. so, it is unsafe.

check for P_4 :-

P_4 request [i] ≤ work.

$[0\ 0\ 2] \leq [0\ 0\ 0]$

It not satisfies P_4. So, it is unsafe.

Part - II

Memory Management Strategies

1. Introduction
2. Swapping
3. Contiguous memory allocation.
4. Pagging
5. Segmentation.

Introduction :-

* Memory is divided into two types. They are :-

1. Primary memory : It is also called as Semiconductors memory.

2. Secondary memory :- It is called as magnetic memory.

* primary memory maitains base values.
* secondary memory maintains limit values.
* Memory management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

Basic concepts :-

Logical Address :-

* It is the address generated by CPU while a program is running is referred to as logical address.
* The logical address is virtual as it does not Exist physically, hence it is also called as virtual address.

Relocation :-

* Relocation is the process of assigning load addresses for position-dependent code and data of a program and adjusting the code &

data to reflect the assigned addresses.

Relocation Register :-

* Relocation register scheme is used to protect user processes from each other, and from changing operating-System code and data.
* Relocation Register contains value of smallest physical address.

physical Address :-

* physical Address identifies a physical location in a memory.
* MMU (memory-management unit) computes the physical address for the corresponding logical address.

Buffer Manager :-

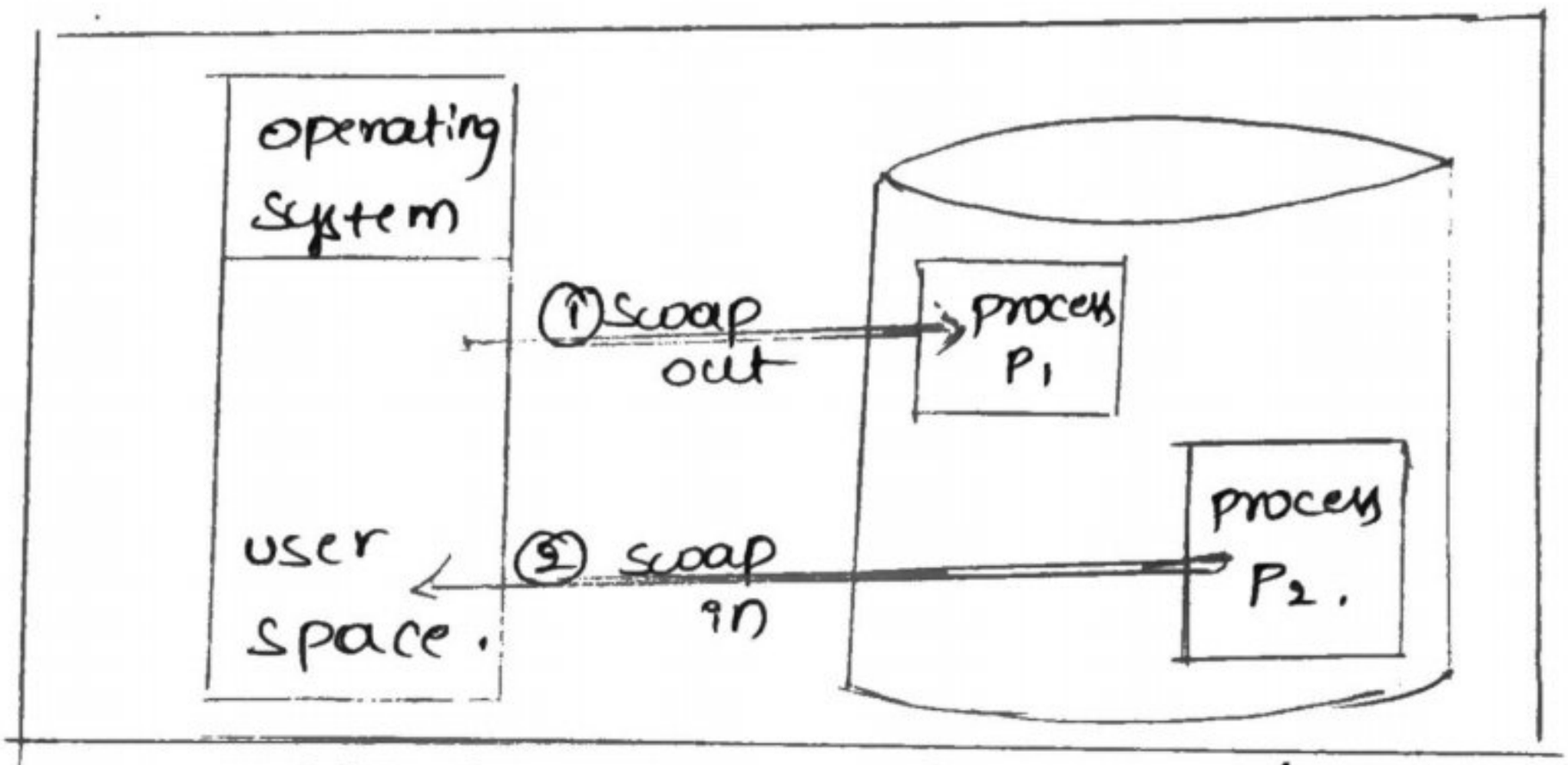
* Buffer manager is a software layer. software layer is appeared on main memory.
* According to buffer manager available memory is divided into equal number of partition. Each & every partition is called page or frame.
* Sometimes main memory gets overloaded at the time needed to transfer some of the pages from main memory to secondary memory.
* Transformation is done based on page replacement algorithms.
* once found any page fault must and should transfer corresponding page from main memory to secondary memory with the help of swapping.
* Sometimes bringing required pages from secondary memory to main memory using swapping.

swapping :-

* swapping is mechanism in which a process can be swapped temporarily out of main memory to secondary storage and make that memory available to other processes which is known as swap-out operation.

* At some later time, the system swaps back the process from, the secondary storage to main memory which is known as swap-in operation.

* Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason swapping is also known as a technique for memory compaction.



Swapping of two processes using a disk as a backing stone.

Contiguous Memory Allocation :-

* In the contiguous memory allocation, both the operating system and the user must reside in the main memory.

* The main memory is divided into two portions one portion is for the operating and other is for the user program.

* In the contiguous memory allocation when any user process request for the memory a single

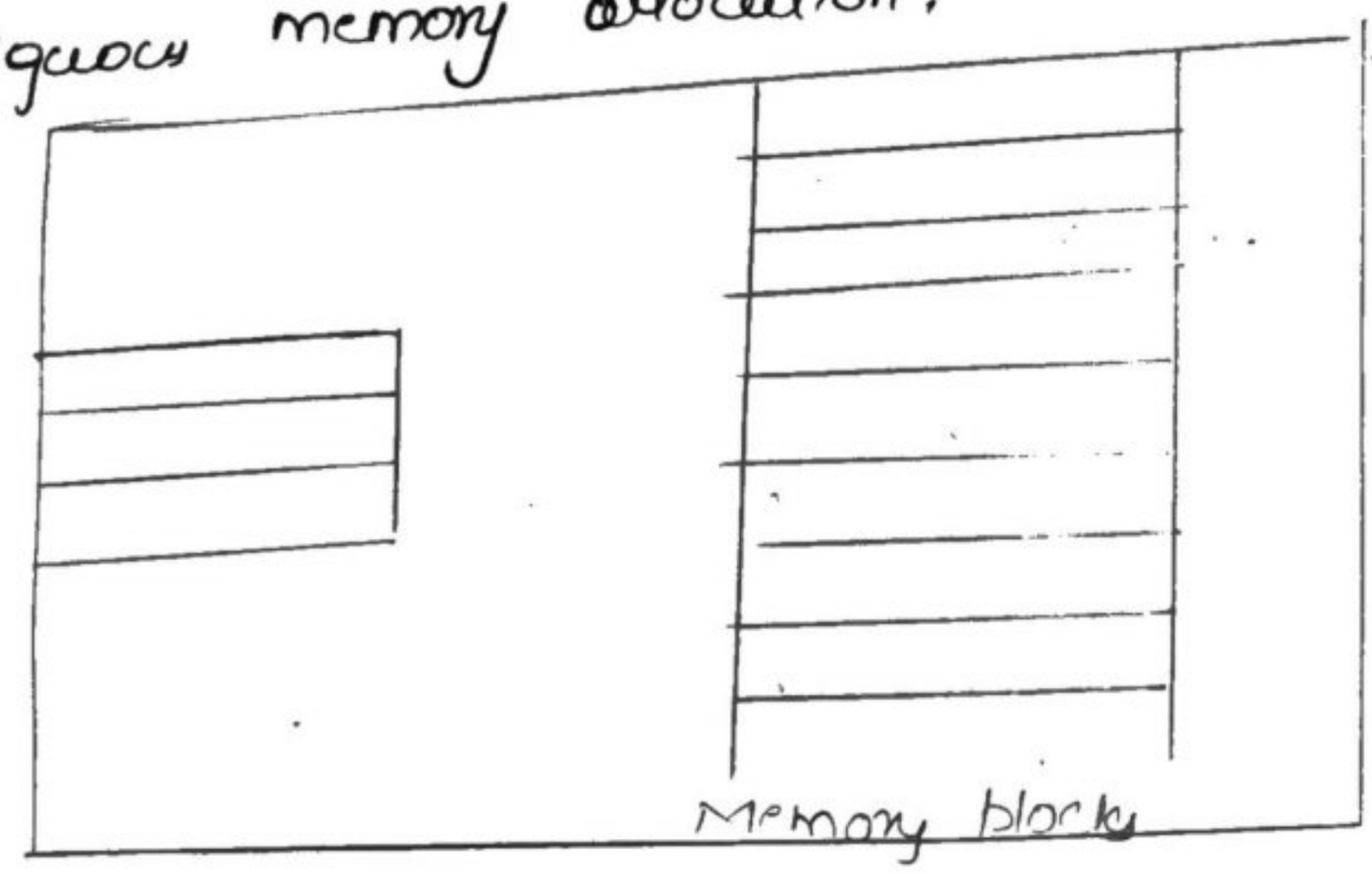
section of the contiguous memory block is given to that process according to its need.

* we can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.

* A single process is allocated in that fixed sized single partition.

* But this will increase the degree of multiprogramming means more than one process in the main memory that bounds the number of fixed partition done in memory.

* Internal fragmentation increases because of the contiguous memory allocation.



Contiguous Memory allocation.

*Fixed sized partition :- In the fixed sized partition the system divides memory into fixed size partition and here entire partition is allowed to a process and if there is some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

* Advantage: Management or Book keeping is easy.

* Disadvantage: Internal fragmentation.

* Variable Size partition: In the variable size

partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

* Advantage: There is no internal fragmentation.

Memory Allocation Algorithms :-

* In Partition Allocation, when there is more than one partition freely available to accomodate a process's request, a partition must be selected.

* To choose a particular partition, a partition allocation method is needed. A partition allocation Method is considered better if it avoids internal fragmentation.

* When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

* Memory Allocation algorithm is divided into 3 types. They are :-

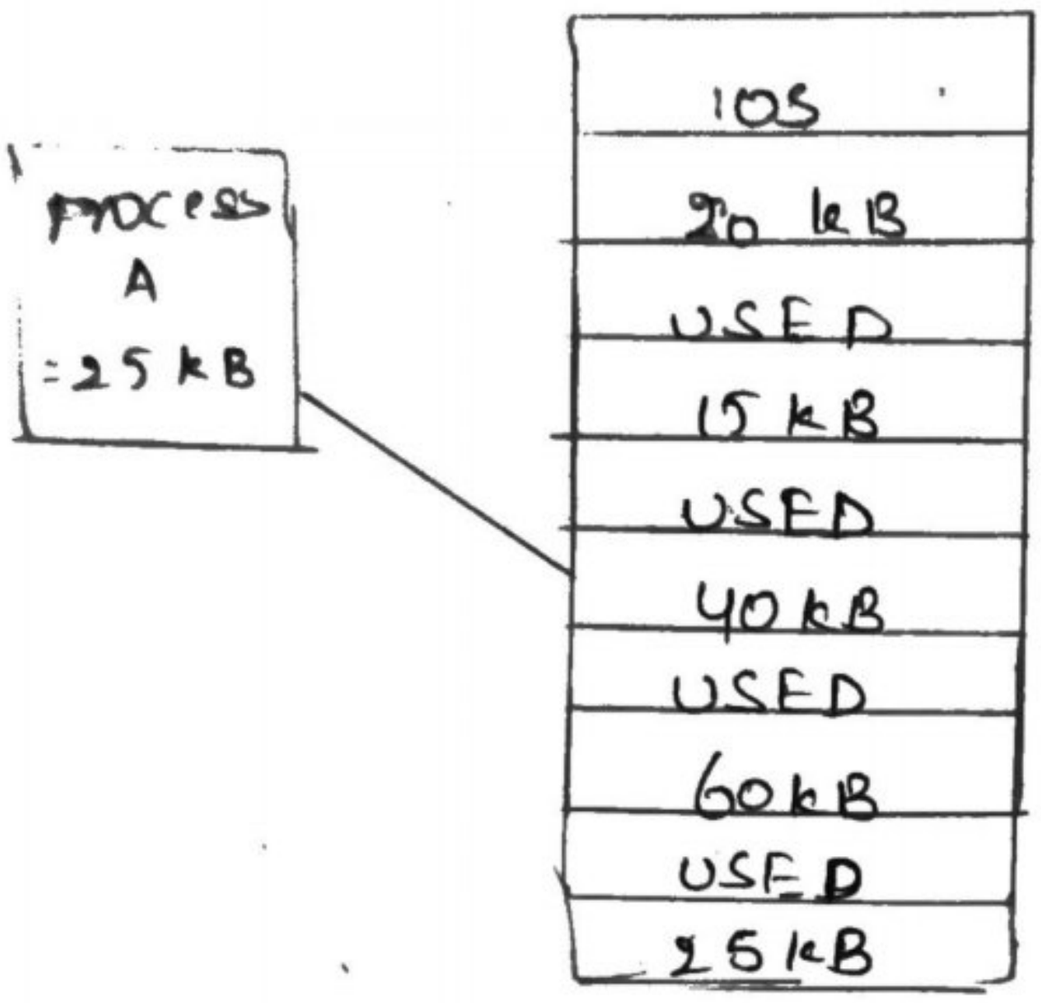
1. First fit Algorithm.
2. Best fit algorithm.
3. Worst fit algorithm.

First Fit Algorithm :-

* According to first fit, Algorithm, it scans memory. from the beginning and choose the first available block that is large enough.

* In the first fit, the partition is allocation which is the first sufficent block from the top

of main memory.



HOLE = 40 - 25 = 15 kB

Advantages of first fit Algorithm :-

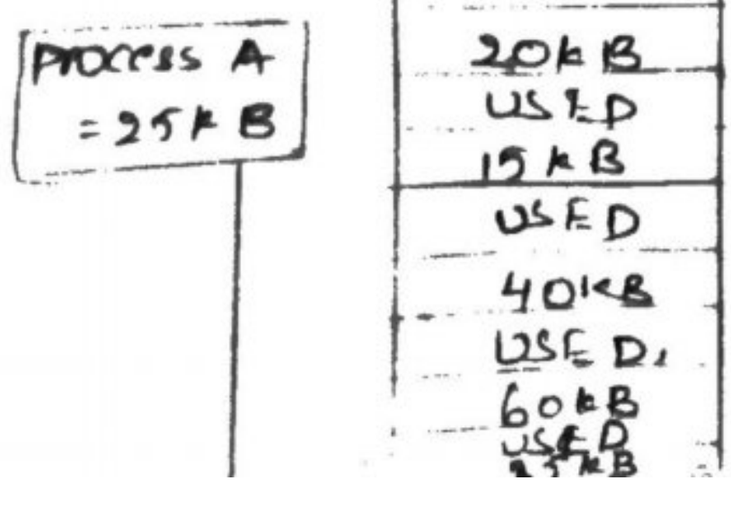
* First fit algorithm minimized cost and time duration.

Drawbacks of first fit algorithm :-

* Internal fragmentation appears.
* Internal fragmentation means process size should maintain lesser size value compared with partition size value.

Best Fit Algorithm :-

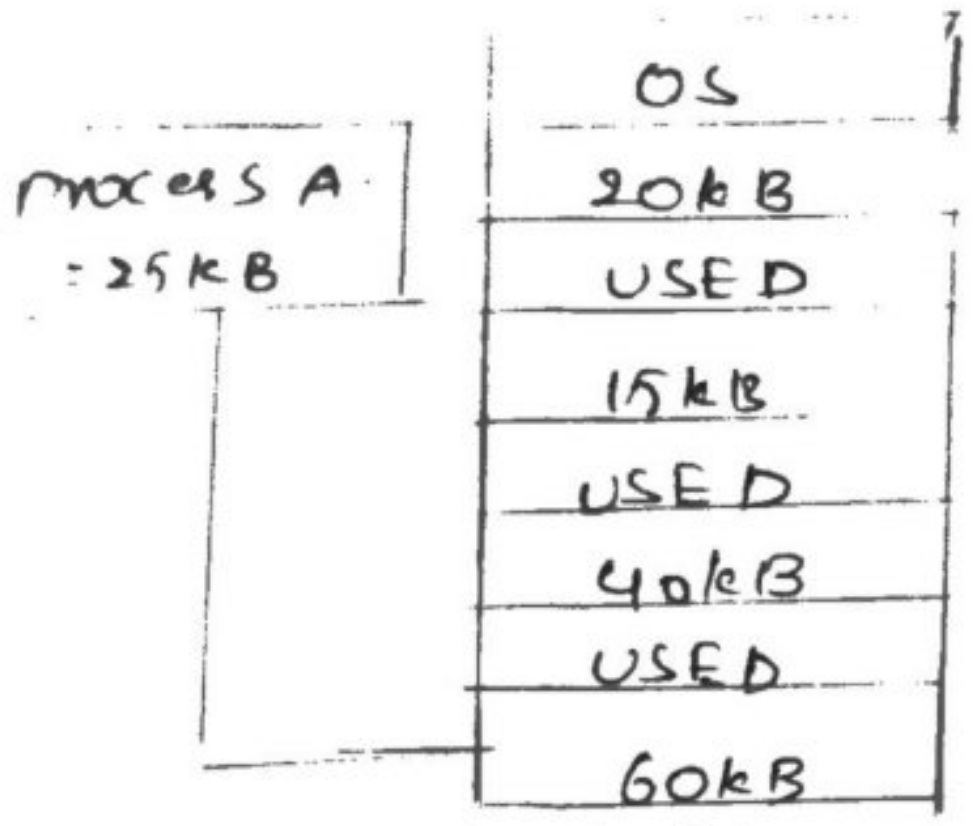
* Best fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.
* It searches the Entire list of holes to find the smallest hole whose size is greater than or Equal to size of process.



Hole = 25 - 25 = 0 kB.

Worst Fit Algorithm :-

* Worst fit allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.
* It is opposite to the best-fit algorithm it searches the entire list of hole to find the largest hole and allocate it to process.



Advantage :-

* The main advantage of worst fit algorithm is processing of more than one process or job is done.

Disadvantage :-

* It has internal fragmentation.
* More time is required.
* It is costlier.

Conclusion :-

*To overcome the drawback (internal fragmentation) we follow other alternative algorithms :-

* paging
* segmentation.
* combination of paging & segmentation.

Compaction :- It is the process of collecting all free spaces of main memory and allocation of free space information) control of [illegible] main memory

Paging :-

→ In computer operating system, paging is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory.

* In this scheme, the operating system retrieves data from secondary storage in same-size blocks called pages
* paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
* This scheme permits the physical address space of Process to be non-contiguous.

- Logical Address or Virtual address :- An address generated by the CPU.
- Logical Address space or virtual address space : the set of all logical address generated by a Program
- physical Address : An address actually available on memory unit.
- physical Address space : the set of all physical addresses corresponding to the logical addresses.
- The physical Address space is conceptually divided into a number of fixed-size blocks, called frames.
- The logical address space is also splitted into fixed-size blocks called pages.
- Page size = frame size.

Address generated by CPU is divided into :-

* Page number (p) : Number of bits required to represent the pages in logical Address space or Page number.

* Page Offset (d) : Number of bits required to represent particular word in a page or page Size of logical Address space or word number of a page or page offset.

Physical Address is divided into :-

* Frame number (f) :- Number of bits required to represent the frame of physical Address space or frame number.

* Frame Offset (d) : Number of bits required to represent particular word in a frame or frame Size of physical Address space or word number of frame or frame offset.

* The hardware implementation of page table can be done by using dedicated registers.
* But the usage of registers for the page table is satisfactory only if page table is small.
* If page table contain large number of entries then we can use TLB : a special, small fast look up hardware cache.

Paging :-

* In computer operating system, paging is a memory management scheme by which a computer stores and retrives data from secondary storage for use in main memory.
* In this scheme, the operating system retrives data from secondary storage in same-size blocks called pages.
* paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
* This scheme permits the physical address space of process to be non-contiguous.

- Logical Address or virtual address :- An address generated by the CPU.
- Logical Address space or virtual address space : The set of all logical address generated by a program.
- physical Address : An address actually available on memory unit.
- physical Address space : the set of all physical addresses corresponding to the logical addresses
- The physical Address space is conceptually divided into a number of fixed-size blocks, called frames.
- The logical address space is also splitted into fixed-size blocks called 'pages.
- page size = frame size.

Address generated by CPU is divided into :-

* page number (p) : Number of bits required to represent the pages in logical address space or page number.

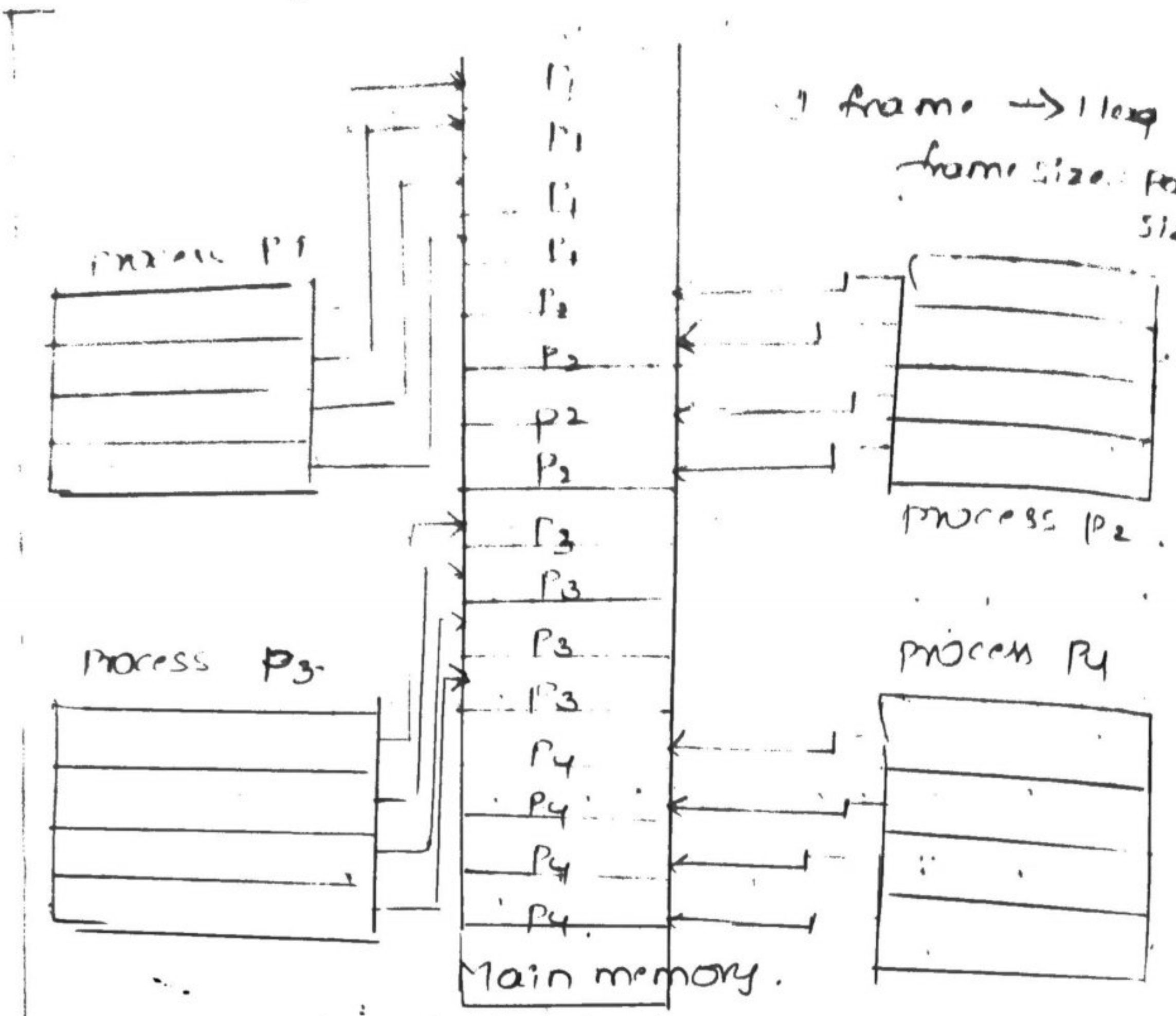
* page offset (d) : Number of bits required to represent particular word in a page or page size of logical Address space or word number of a page or page offset.

Physical Address is divided into :-

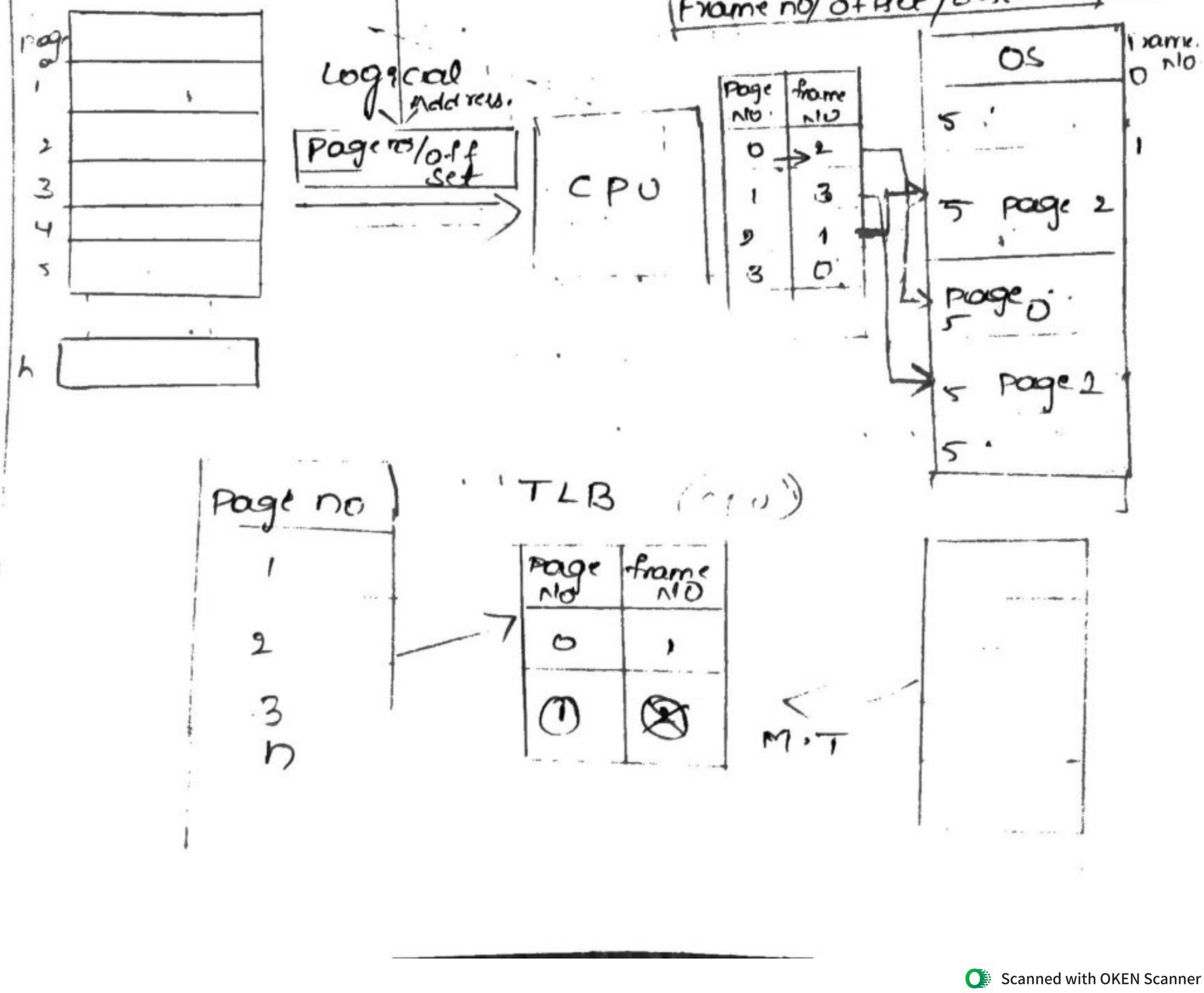
* Frame number (f) :- Number of bits required to represent the frame of physical Address space or frame number.

* Frame offset (d) : Number of bits required to represent particular word in a frame or frame size of physical Address space or word number of frame or frame offset.

* The hardware implementation of page table can be done by using dedicated registers.
* But the usage of registers for the page table is satisfactory only if page table is small.
* If page table contain large number of entries then we can use TLB ; a special, small fast look up hardware cache.



Pagging technique :- (structure) :-



conclusion :-

Ragging technique suffering with internal fragmentation to overcome this drawback we are using "segmentation technique".

Page Replacement Algorithm :-

*Page replacement algorithms are divided into :-

1) FIFO (First in first out).
2) LRU (Least Recently used).
3) Optimal
4) LFU (Least frequency used)
5) MRU (Most Recently used)
6) clock technique.

*Page replacement algorithms are used to find out page found and hard page from the main memory and replace the corresponding place from main memory.

* It also support to insert new page into replacement page of the main memory.

FIFO :-

*FIFO Standy for first in first out.

*It is a page replacement algorithm

Algorithm for FIFO :-

1. Start the process
2. Declare the variables.
3. Read the no. of pages and no. of frame.
4. Allocate the pages to frame.
5. If no. of available pages > no. of frames then replace the pages of memory based on fifo

6. Display the result

7. Stop the process.

Example 1 :- Consider page reference string 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 with 3 page frames.

Soln	2	3	2	1	5	2	4	5	3	2	5	2
	(2)	2	2	2	(5)	5	5	5	(3)	3	3	3
		(3)	3	3	3	(2)	2	2	2	(2)	(5)	5
			(1)	2	1	(4)	4	4	4	4	4	(2)
				PF1	PF2	PF3		PF4			PF5	PF6

*Total Number of page fault = 6

*Page Fault rate = number of page faults / number of pages in the string

$= 6/12 = 0.5$ //.

Example 2 :- Consider page reference string 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 5, 4, 5, 4, 2.

1	2	3	4	5	3	4	1	6
(1)	1	1	(4)	4	4	4	4	(6)
	(2)	2	2	(5)	5	5	5	5
		(3)	3	3	3	3	(1)	1
			PF1	PF2			PF3	PF4

7	8	7	8	9	5	4	5	4	2
6	6	6	6	(9)	9	9	9	9	(2)
7	7	7	7	7	(5)	5	5	5	5
1	(8)	8	8	8	8	(4)	4	4	4
PF5	PF6			PF7	PF8	PF9	PF10		PF10

Total Number of page fault = 10

page fault rate = number of page fault / number of page in string

10/14 : 0.53

LRU (Least Recently Used) :-

-Algorithm :

1. Start the process
2. According to LRU algorithm we need to search or scan available main memory atleast once.
3. According to LRU algorithm the algorithm checks for : i. Recently used pages (RU)

   ii) Pre recently used page (PRU)

   iii) Least recently used page (LRU)
4. Once find least Recently used pages replace the least recently used pages.
5. Insert new pages in place of least recently used pages.
6. Display the result.
7. Stop the process.

Example :- 1 :- Consider page reference string 2, 3, 2, 1, 5, 2, 4, 5, 3, 2.

2	3	2	1	5	2	4	5	3	2	5	2
(2)	2	2	2	2	2	2	2	(3)	3	3	3
	(3)	3	3	(5)	5	5	5	5	5	5	5
			(1)	1	1	(4)	4	4	(2)	2	2
				PF1		PF2		PF3	PF4		

Total No. of page faults = 4

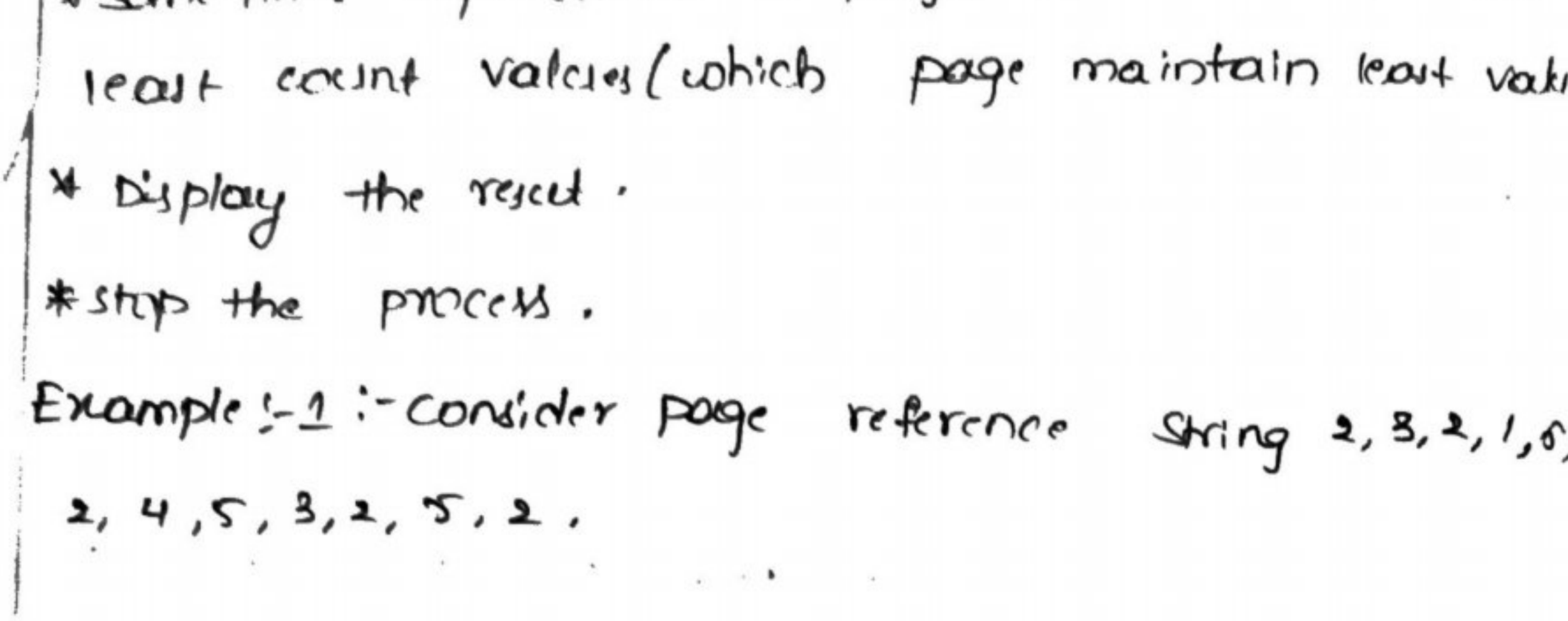
page fault rate = number of page fault / number of page in string

= 4/12 = 0.333 //..

optimal page Replacement Algorithm :-

* It is a page replacement algorithm.
* optimal algorithm entirely required a string (collection of pages).
* Available pages needed to allocate, available frames of memory.
* According to optimal algorithm needed to inserting new pages into frames of memory. At that time must and should require replacement of pages.
* Replacement of pages is done based on "least valeie".
* Sometimes Replacement of pages is done base on least count values (which page maintain least valig
* Display the reset.
* stop the process.

Example :- 1 :- Consider page reference string 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2.

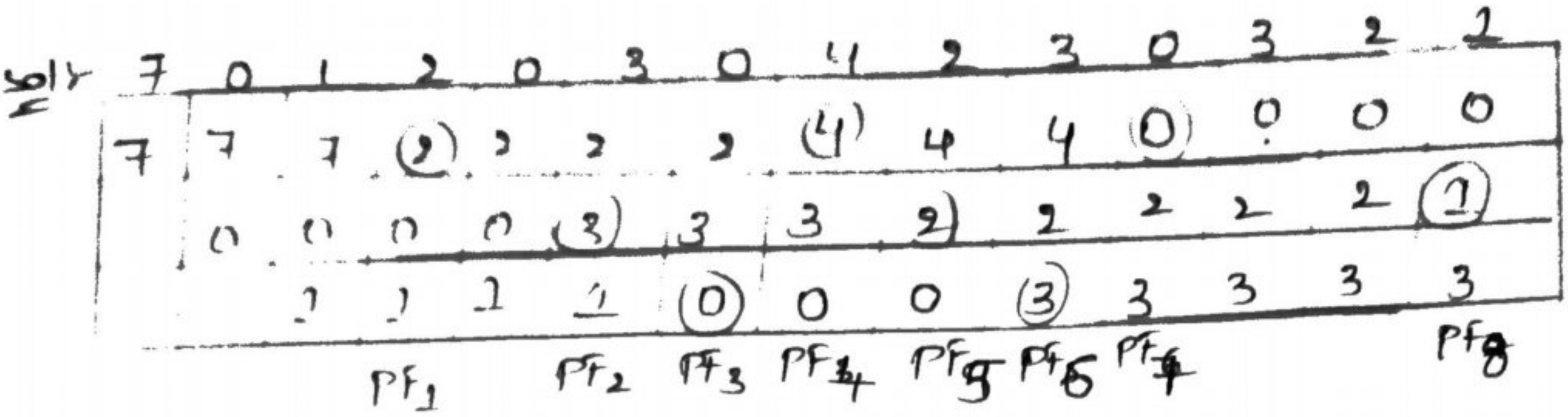


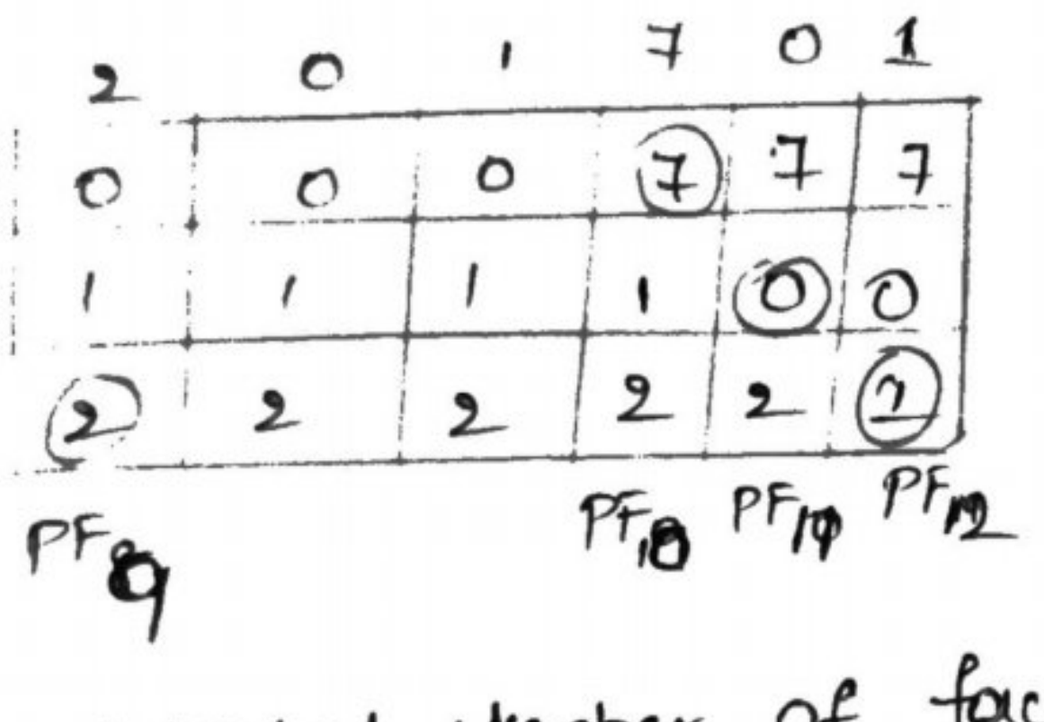
Total Number of page faults = 3

Page faalt rate = No. of page facult / Number of Pages in string

= 3/12 = 0.25

Example :2 consider a string 7,0,1,2,0,3,0,4,2,3,0,3,2
2,1,2,0,1,7,0,1



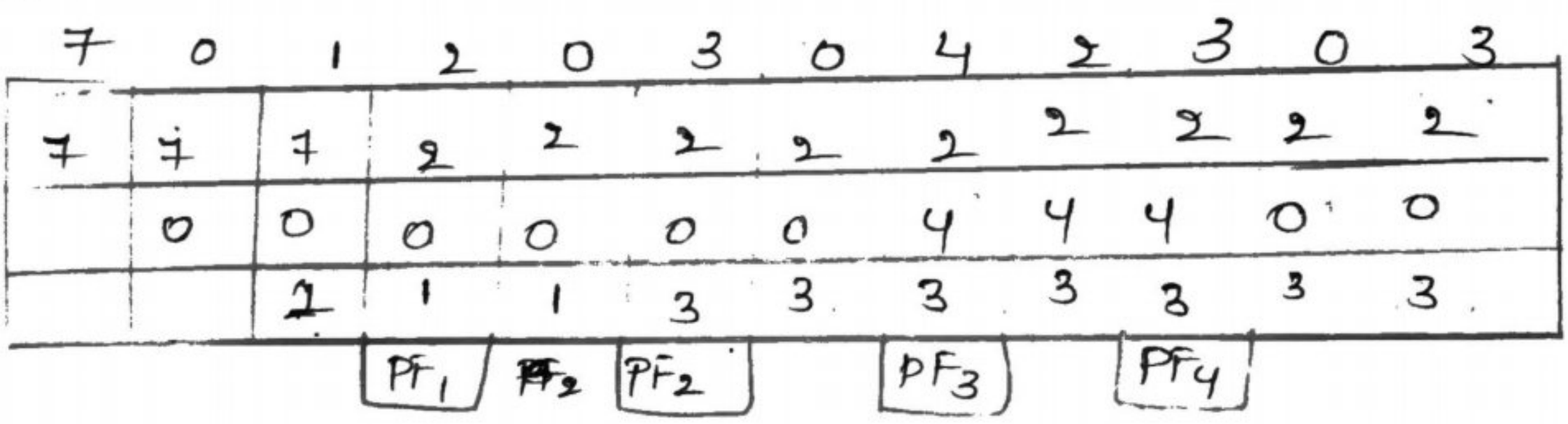


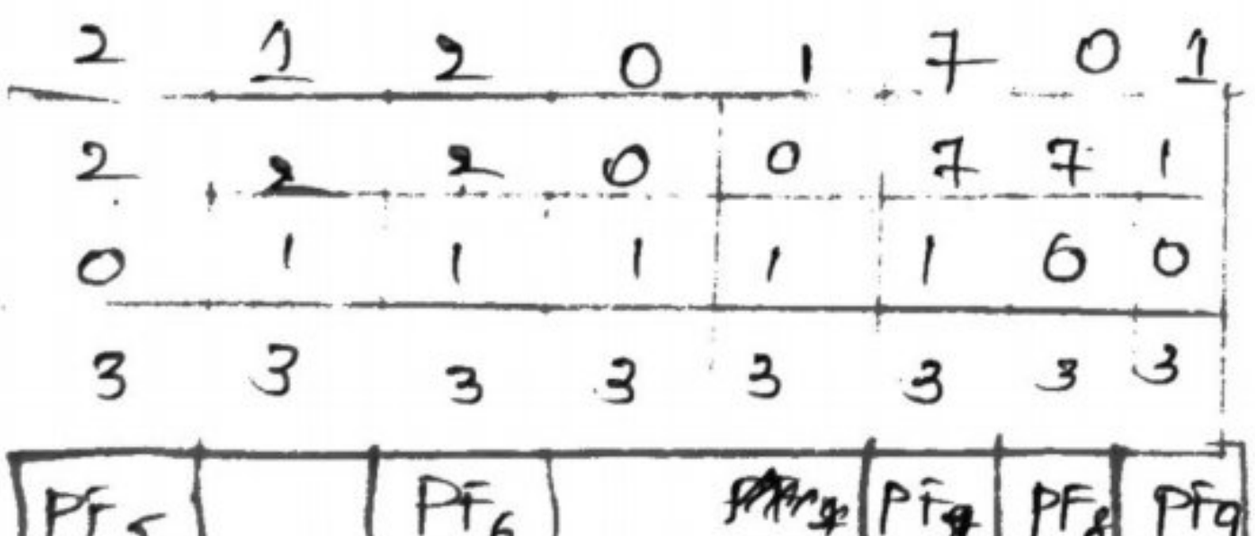
∴ Total Number of faults = 12.

Page fault rate = number of page faults / number of pages in string

$$= \frac{12}{20_{10}} = \frac{3}{5}$$

LRU :-





Total No. of pages = 9

Page fault rate : Number of page faults / Number of pages in string = 9/20

=

LFU (Least frequently used) :-

* LFU stands for least frequently used.
* It is a page replacement algorithm.
* Page replacement is used to find out hard page and fault page from mainmemory and corresponding place from the main memory.
* Insert new pages into removed replace pages of the main memory.
* page replacement algorithm is also used to finding of page fault rate.
* page fault rate is calculated by the help of using mathematical formula.

page fault rate = No. of page faults / Total No. of pages in given string.

LFU main Algorithm :-

1. Intially required string (Collection of pages).
2. start the process
3. Available pages needed to allocate available frames of memory.
4. According to LFU algorithm needed to allocate

of any new pages into frames of memory at that time required replacement of pages.

6. Replacement of pages is done based on different conditions :-

a. Replacement is done on "least count values".

b. Sometimes does not used recently used replacement pages.

c. Sometimes does not use cache memory pages (longer time frequently used pages).

7. Display the result.

8. Stop the process.

Example 1 :- Consider page reference string 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 with 3 page frames.

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	4	4	4	4	5	5
			1	5	5	5	5	3	3	3	3
				PF_1		PF_2		PF_3		PF_4	

∴ No. of page faults = 4

∴ Page fault rate = No. of page faults / Total No. of No. of pages in string

$= \frac{4}{12} = 0.33$ //.

Segmentation : technique :-

* Intially consider any process (or) user application.
* User application is divided into no. of partitions.
* Each and every partition assign to segment numbers.
* segment means collection of (or) grouping of pages.
* Each and every segment information required to send into processor for the purpose of execution.
* CPU response and assigns a some address value to the each & every segment (using

logical address).

*According to segmentation technique available logical address is divided into two Partition.

1. Segment Number
2. Offset / Displacement (d).

* Segmentation technique provides segment map table to the user.

*segment map table supports three attribule values

1. Segment Number
2. Base address
3. Limit value

*According to segmentation technique Physical address divided into two partition values are;-

1) Base address

2) off set / Displacement

*when both logical address offset & physical address offset values are Equal at that time to transfer segment information into Base address value of the memory.

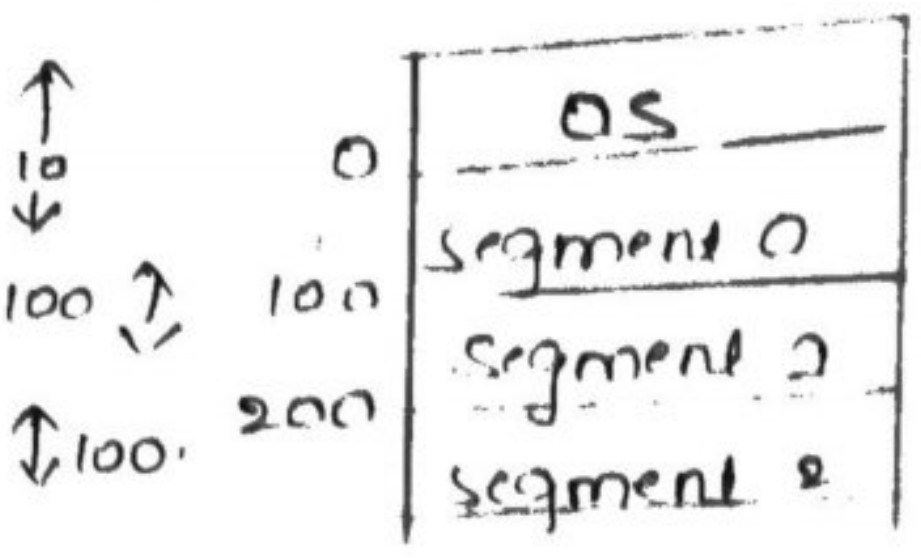
*sometimes segmentation technique suffering with External fragmentation.

segment table :-

segment number	Base Number	Limit value
0	0	100
1	100	100
2	100	100

Physical address.

Base address	offset/displacement



Difference Between segmentation and pagging technique.

Pagging	Segmentation
1. Intially require any process or job.	2. Intially require any process or job.
2. Available process is divided into number of pages.	2. Available process is divided into number of segments.
3. According to pagging technique logical address Contains page number and offset/displacement.	3. According to segment technique logical address contain segment number & offset/displacement.
4. Maintain page table	4. Maintain segment table.
5. It maintain 2 attribute values page number & frame value.	5. It maintain 3 attribute values segment number, limit and base values.
6. According to page technique physical address Contain frame number & offset/displacement.	6. According to segmentation technique physical address contain base value and displacement.
7. It is suffering with internal fragmentation.	7. It is suffering with external fragmentation.

Files :-

* A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

* In general a file is a sequence of bits, bytes lines or records whose meaning is defined by the files creator and user.

File structure :-

A file structure should be according to a required format that the operating system can understand.

* A file has a certain defined structure according to its type.

* A text file is a sequence of characters organised into lines.

* A source file is a sequence of procedures and functions.

* An object file is a sequence of bytes organized into blocks that are understandable by the machine.

* when operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File type :-

* File type refers to the ability of the operating system to distinguish different types of the file such as text files source files & binary files Etc. Many operating systems support many types of files

operating system like MS DOS and UNIX have the following types of files.

1. ordinary files

* these are the file that contain user information
* these may have text, database or executable program
* the user can apply various operations on such files like add, modify, delete or even remove the entire file.

2. Directory files :-

* these files contain list of file name and other information related to these files.

3. Special files :-

* these files are also known as device file.
* these files represented physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types :-

character special files :- data is handled character by character as in the case of terminals or printers.

Block special files - Data is handled in blocks as in the case of disks and tapes.

Operation on files :-

1. Search a file : Any user want to searching of any file is available or not available under directory.

2. Create a file : Any user want to creating of new file is done with help of using create() file operation. create of file is used to properly create a new file.

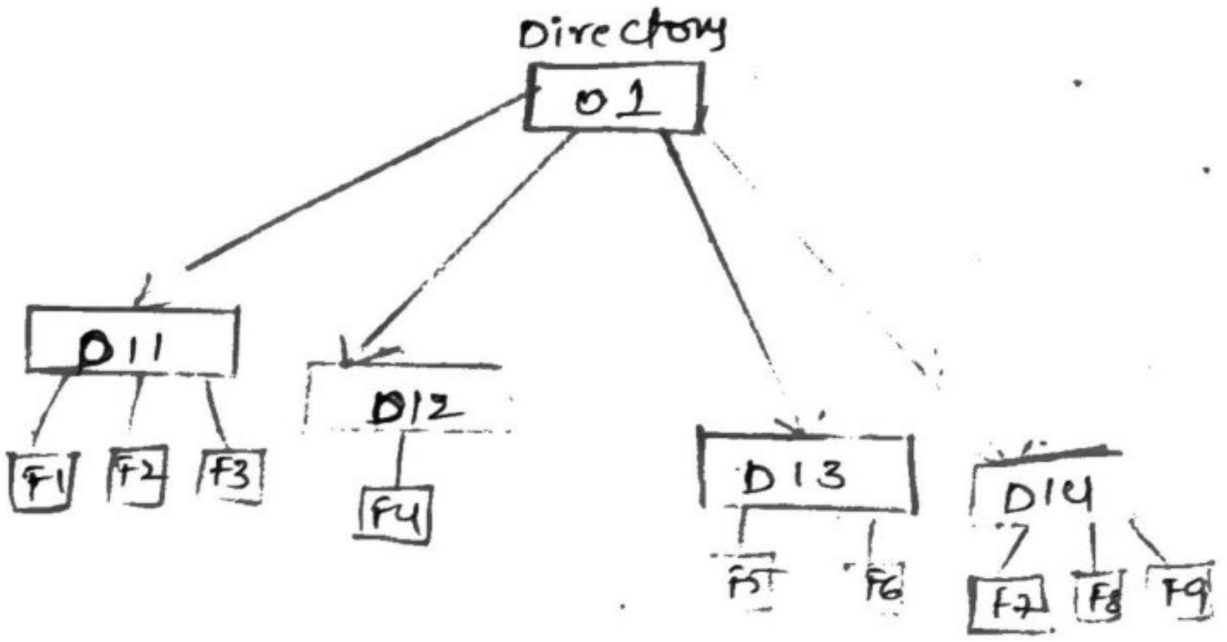
3. Delete a file : Any user want to deleting of particular file from directory.

4. Transfer a file : To transfer a file from one directory to another directory.

5. list a file / List a directory : To grouping of no. of files into single at that time we use list a file.

Directories :-

* A directory is the collection of the correlated files on the disk.
* In a directory, we can store the complete file attributes or some attributes of the file. A directory can be comprised of various files.
* with the help of the directory, we can maintain the information related to the files

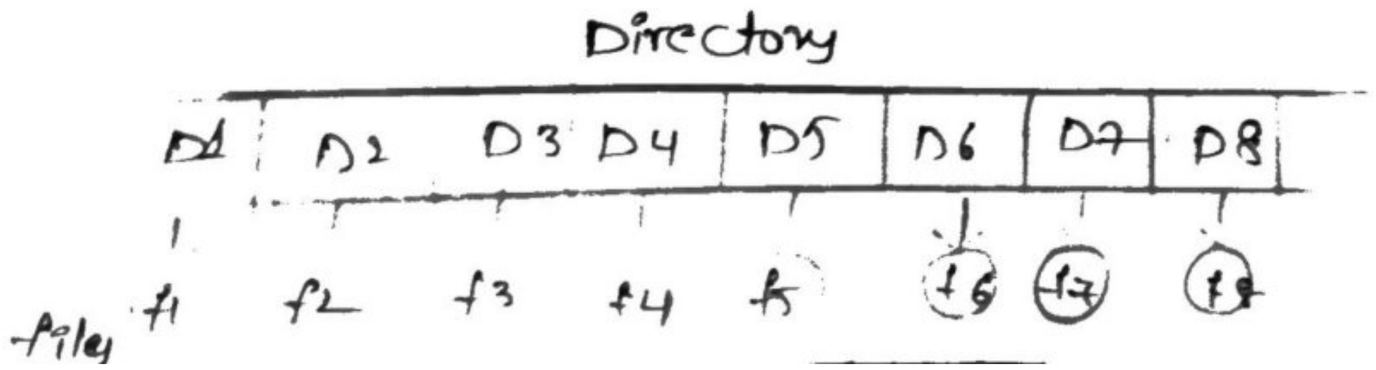


Single level Directory :-

1-Single-Level Directory is the Easiest directory Structure.

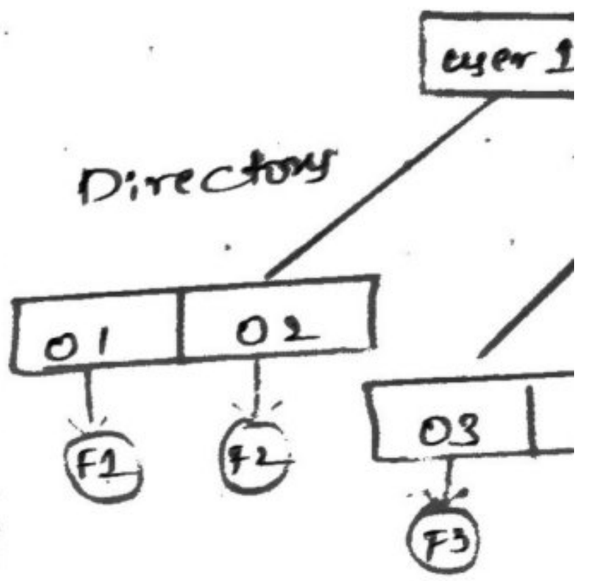
There is only one directory in a single-level directory, and that directory is called a root-directory.

* In a single-level directory, all the files are present in one directory that makes it easy to understand.
* In this, under the root directory, the user cannot create the subdirectories.



Two-level directory

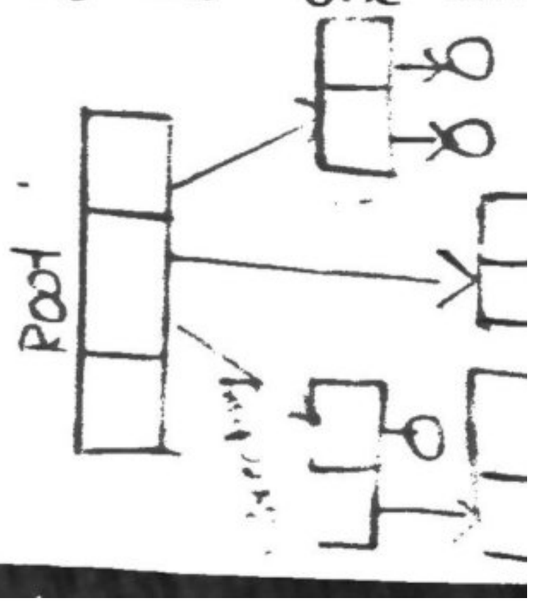
* Two-level directory structure, in this individual directory
* there is one ma directory that inclu every user.
* At the second level different directory without permition, other user's direc



files.

Hierarchial directory

A tree structured structure in which directory or a fil The tree-structured the two-level dire files into one dir



iping of
. coe case

e correlated

mplete file
e A directory

an maintain

F9

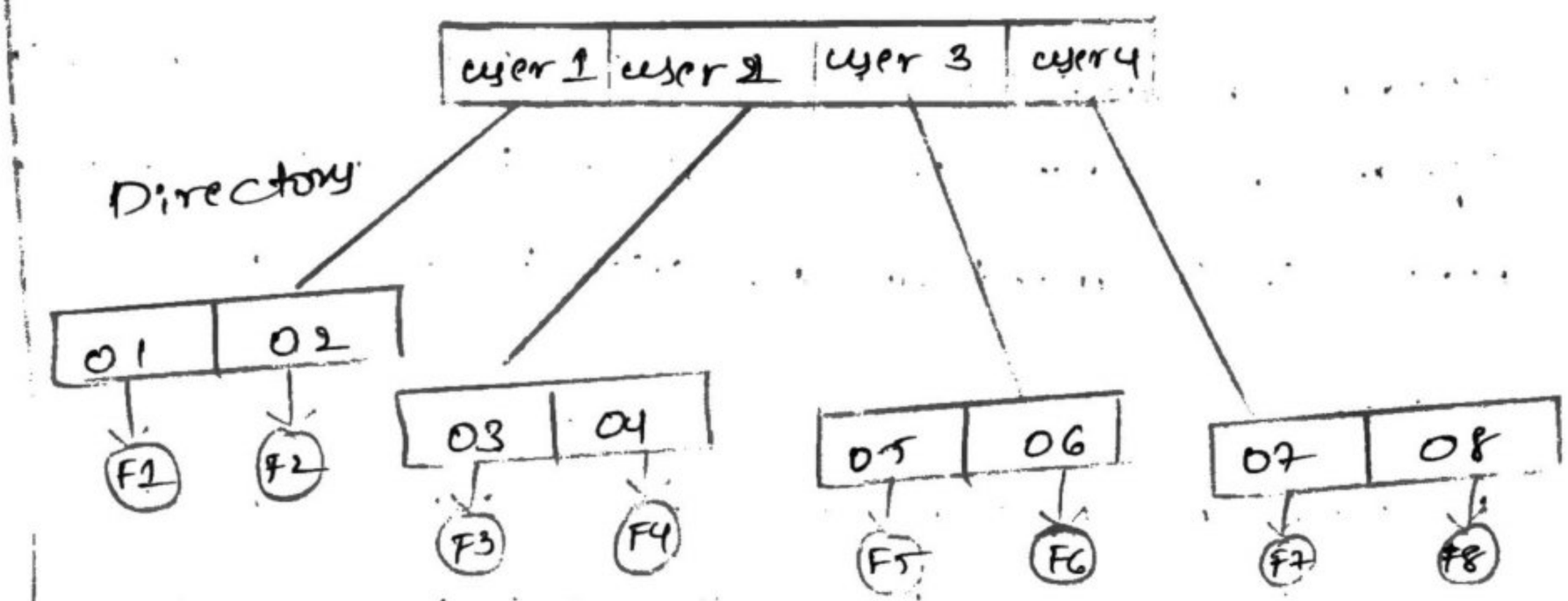
irectory structure.
ingle-level
lled a root-

files are present
to understand.
the user cannd-

D8
f8

TLDO-level directory :-

* TWO-level directory is another type of directory structure. In this it is possible to create an individual directory for each of the users.
* there is one master node in the two-level directory that include an individual directory for every user.
* At the second level of the directory, there is a different directory present for each of the users. without permission, no user can enter into the other user's directory.:

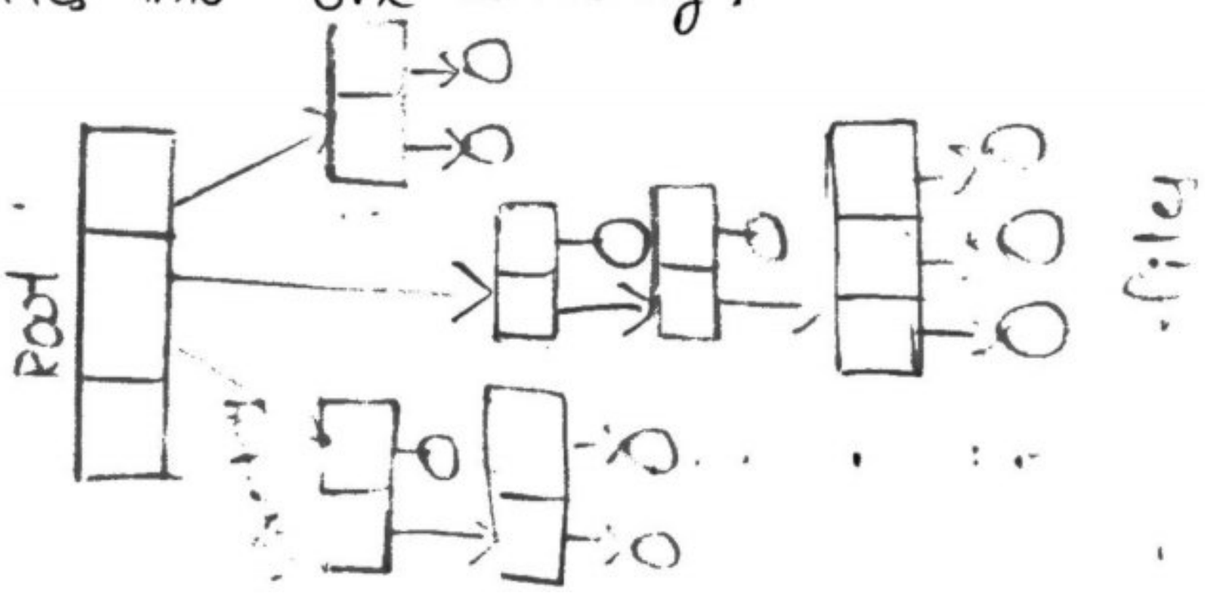


files.

Hierarchial directory :-

A tree structured directory is another type of directory structure in which the directory enter may be a sub-directory or a file.

The tree-structured directory reduces the limitations of the two-level directory. we can group the same type of files into one directory.



File System Implementation :-

we can implement file system by using two types of data structures.

(1) ON-disk Structures :-

* Generally they contain information about total number of disk blocks, free disk blocks, location of them and etc. Below given are different on-disk structures.

* Boot control Block :-

It is usually the first block of volume and it contains information needed to boot on operating system. In UNIX it is called boot block and in NTFS it is called as partition boot sector.

* Volume control Block :-

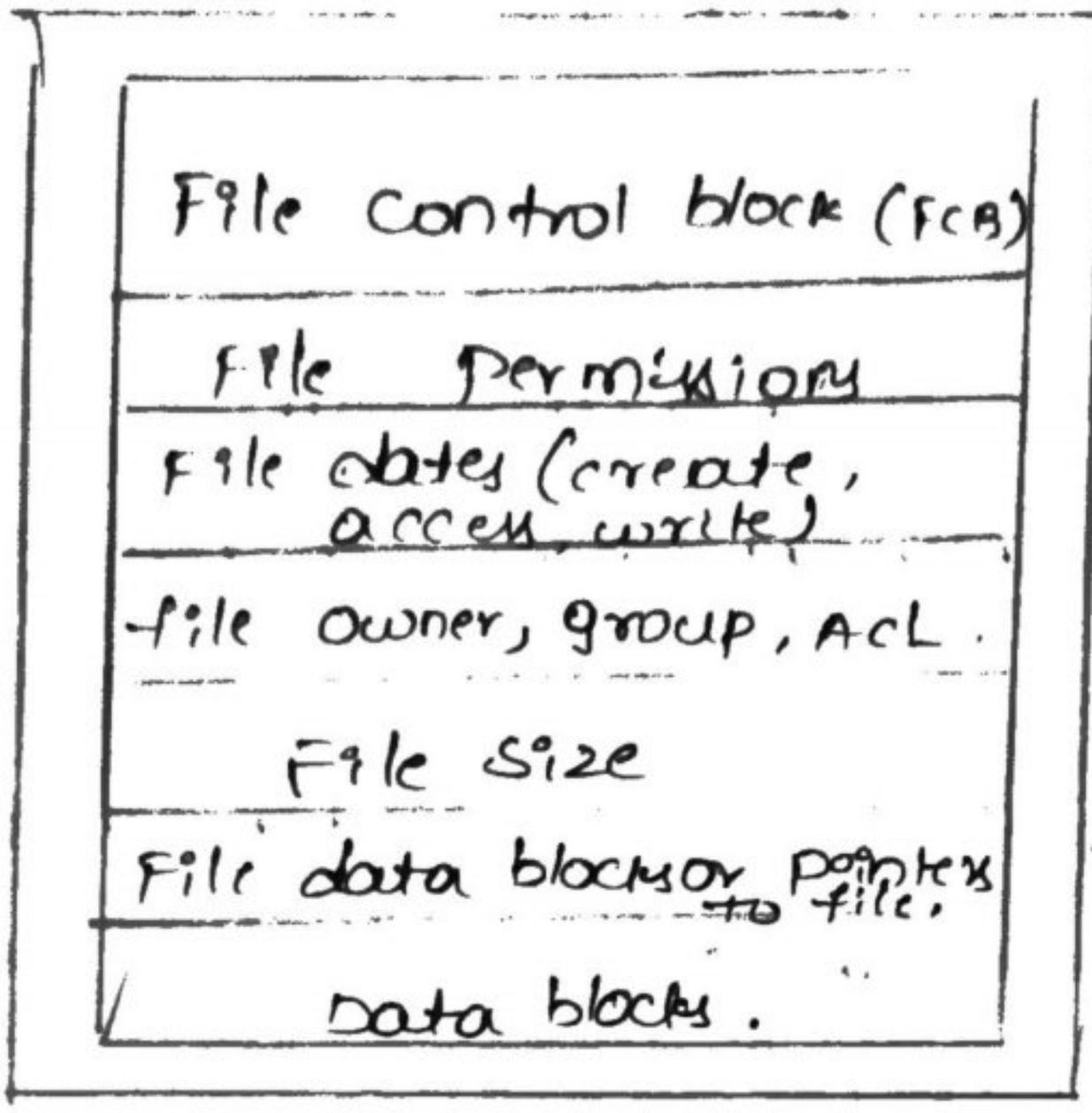
It has information about a particular partition ex: free block count, block size and block pointers etc. In UNIX it is called super block and in NTFS it is stored in master file table.

* Directory Structure :-

They store files names and associated inode numbers In UNIX, includes file name and associated file names and in NTFS, it is stored in master file table.

* per-file FCB :-

It contains details about files and it has a unique identifier number to allow association with directory entry In NTFS it is stored in master file table.



② In-Memory structures :-

They are maintained in main-memory and these are helpful for the file system management for coaching, several in-memory structures given below :-

1. Mount table :-

It contains information about each mounted volume.

2. Directory-structure cache :-

This cache holds the directory information of recently accessed directories.

3. System Wide Open-file table :-

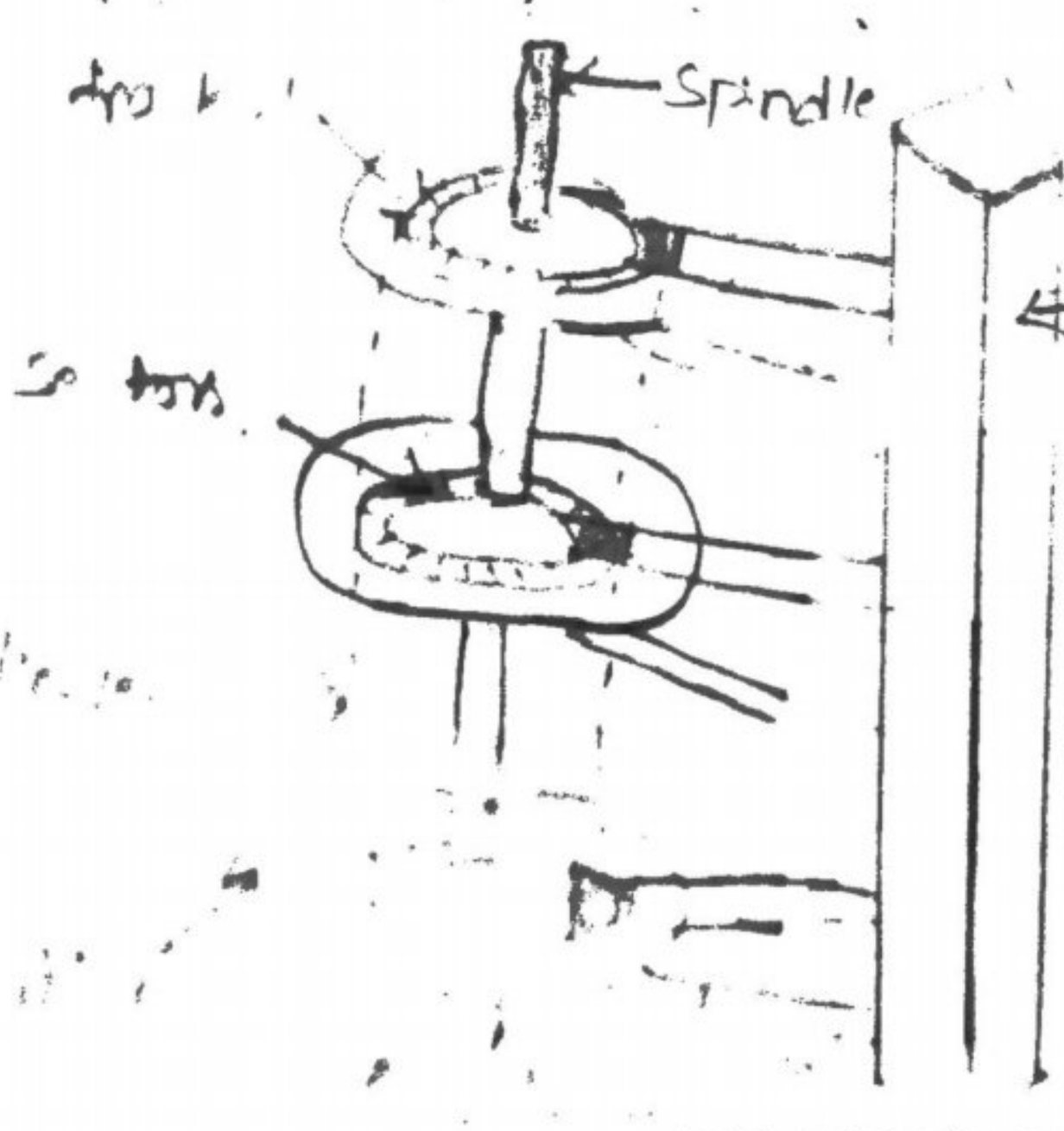
It contains the copy of FCB of each open file.

4. per-process open-file table :-

It contains information opened by that particular process and it maps with appropriate system wide open-file.

Part - II

* Overview of Disc Structure and Attachments :-
  * It maintains a spindle.
  * spindle is a cylindrically type.
  * Spindle is interconnected with no. of circular representation.
  * Each and every circular representation is called Floppy Disk (or) CD (compatible Disk) or DVD (Disk Vedio Device)
  * Each and every CD/DVD toy maintained a surface. Surface is filled with magnetic material (or) magnetic core :
  * Available CD/DVD is attached with read and write heads with the help of disk/actuator.
  * Read head is the process of retrieving/reading any information from the CD/DVD.
  * Write head is taken care of to properly sending of any information from private area into either CD/DVD
  * CD/DVD are rotated with help of motor.



Raid Structure :-

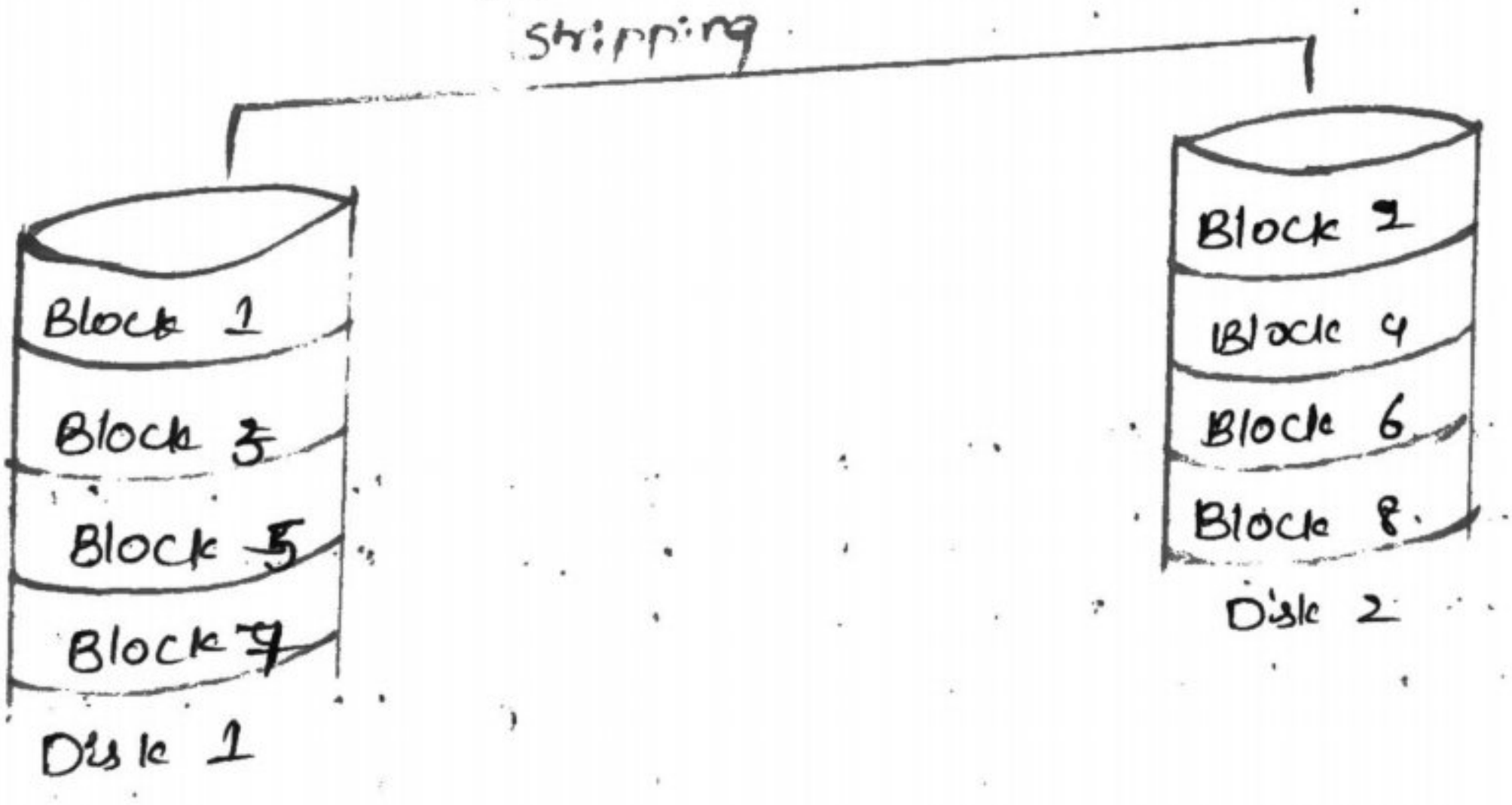
* Raid, or Redundant Arrays of Independent Disks is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both.

RAID Levels :-

Raid 0 RAID 3 RAID 6.
RAID 1 RAID 4
RAID 2 RAID 5

RAID 0 :-

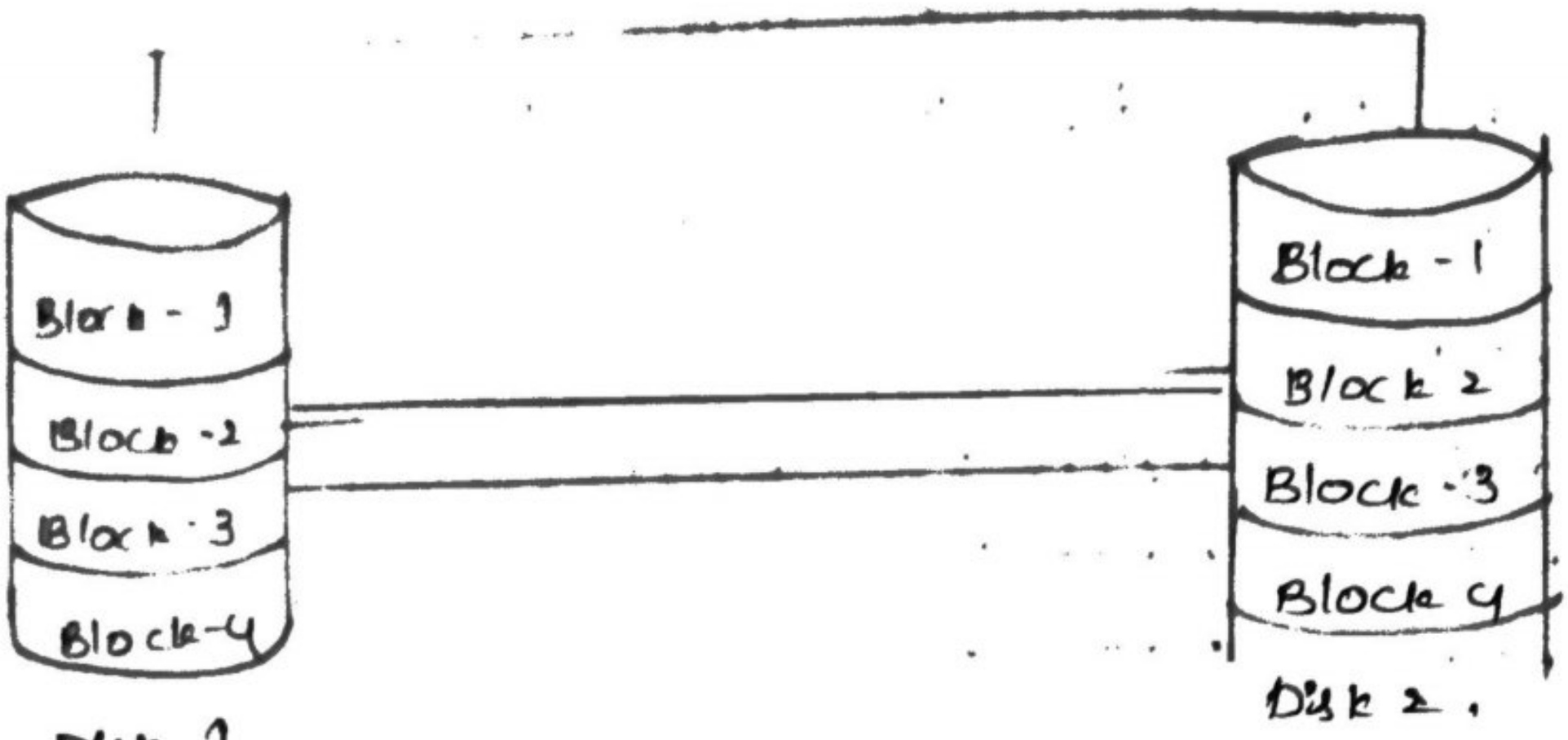
This configuration has stripping, but no redundancy, of data. It offers the best performance, but it does not provide facelt tolerance.



RAID 1 :-

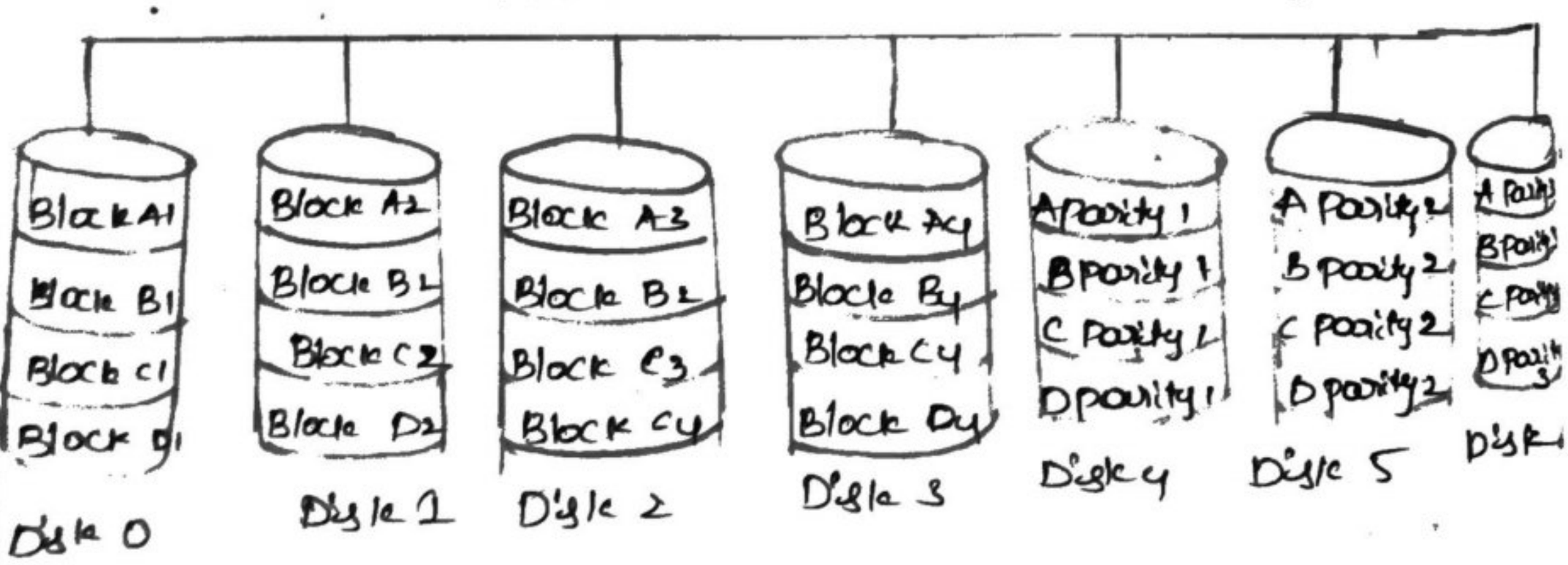
Also known as disk mirroring, this configuration consists of atleast two drives that duplicate the storage of data. there is no stripping. Read performance is improved since either disk can be read at the same time. write performance is the same as for single disk storage.

PAID 1
MIRRORING



RAID 2 :- The configuration uses stripping across disk with some disks storing error checking and correcting (ECE) information. Raid 2 also used a dedicated hamming code parity, a linear form of error correction code. RAID 2 has no advantage over RAID 3 and is no longer used.

RAID 2



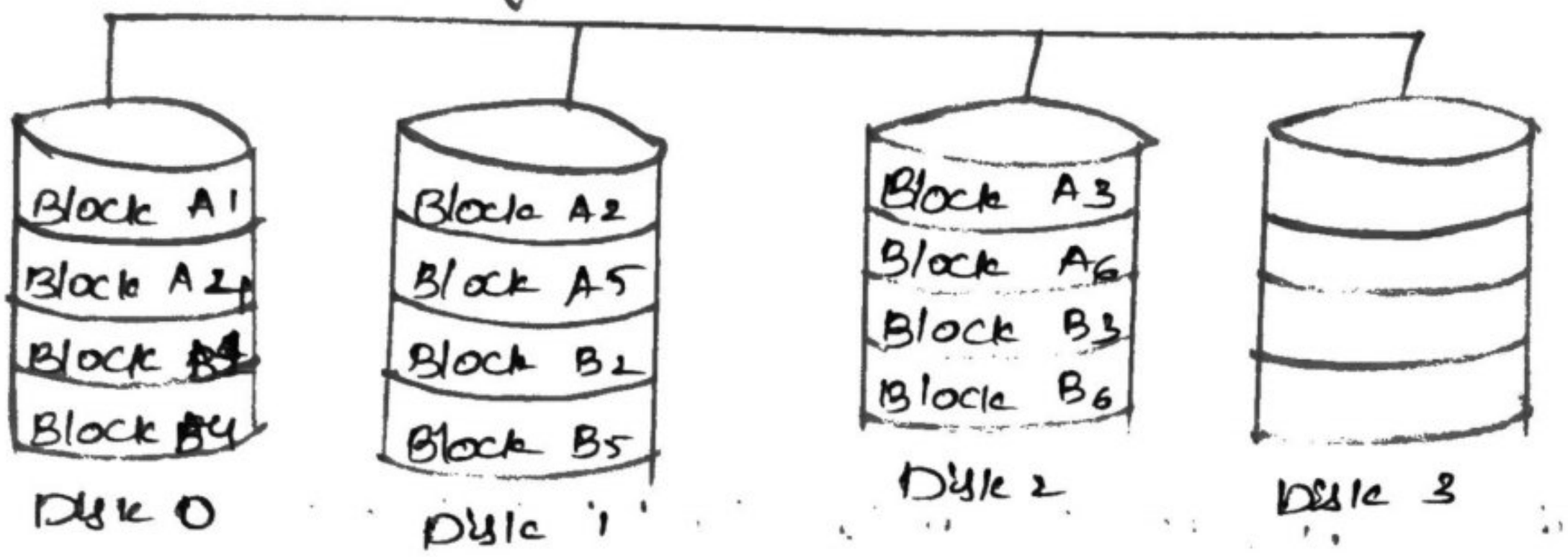
RAID-3 :-

This technique uses stripping and dedicates on drive to storing parity information. The embeded ECC information is used to detect errors. Data recovery is accomplished by calculating the exclusive information recorded on other drives since an I/O operation addresses all the drives at the same time, RAID 3

cannot overlap I/o. for this reason, RAID 3 is best to single-user systems with long record application.

RAID-3

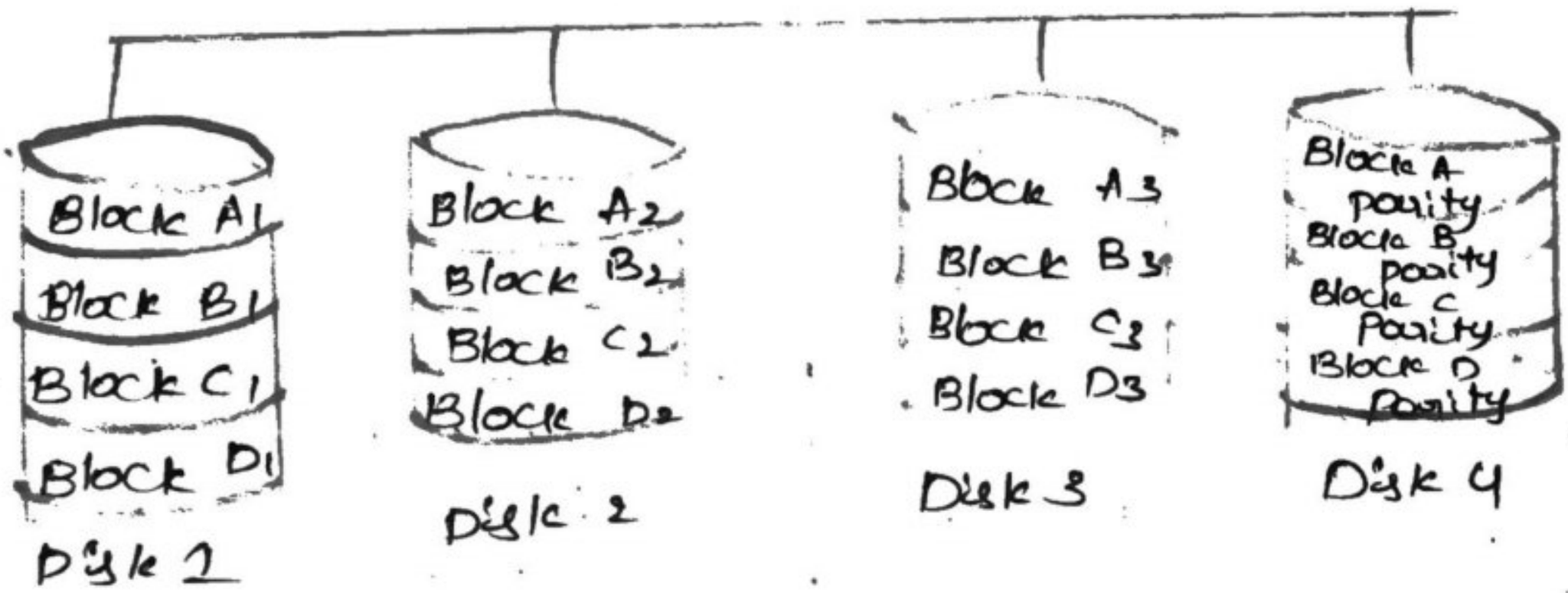
Parity on seprate Disk



RAID 4:-

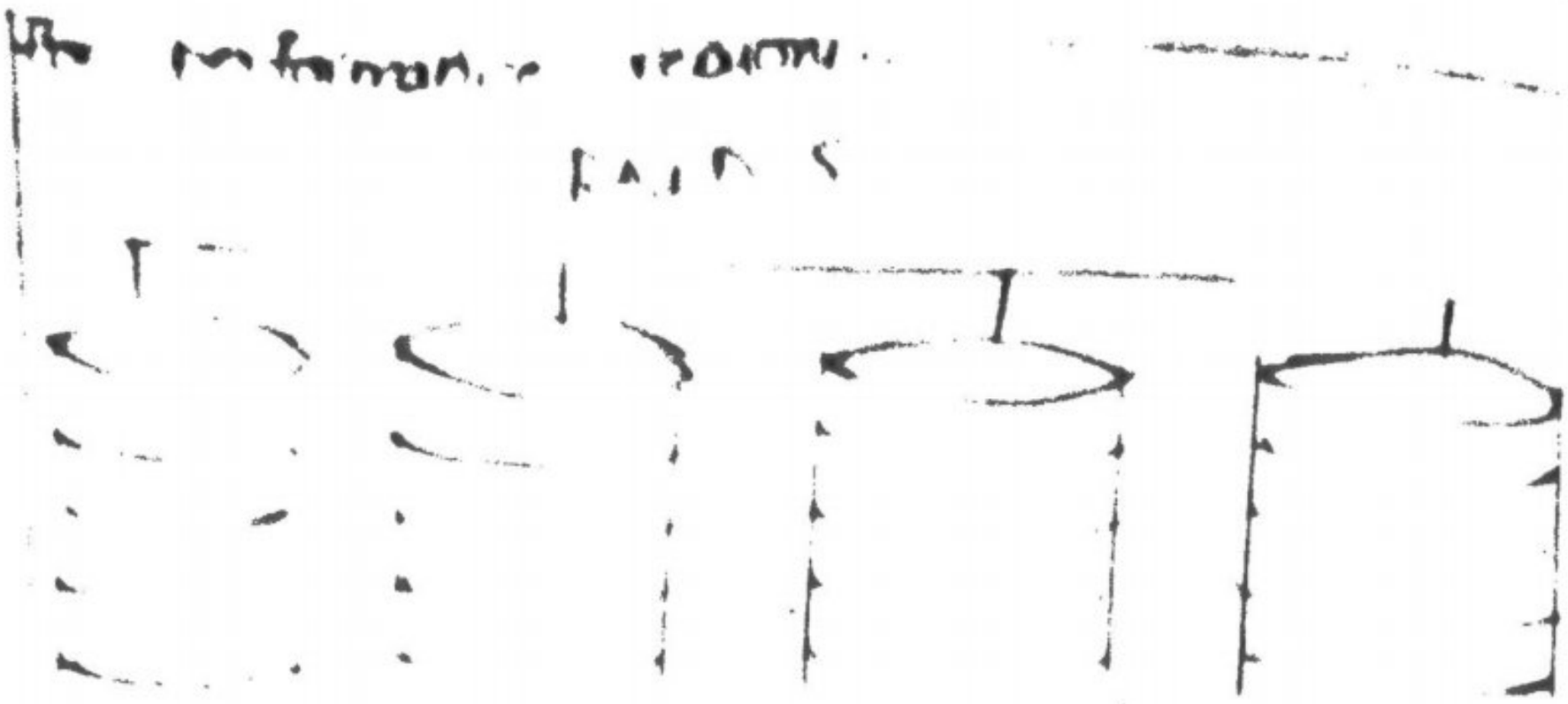
This level uses large strips, which means a user can read records from any single drive. overlapped I/o can then be used for read operations. Since all write operations are required for update the parity drive, no I/o overlapping is possible.

RAID 4



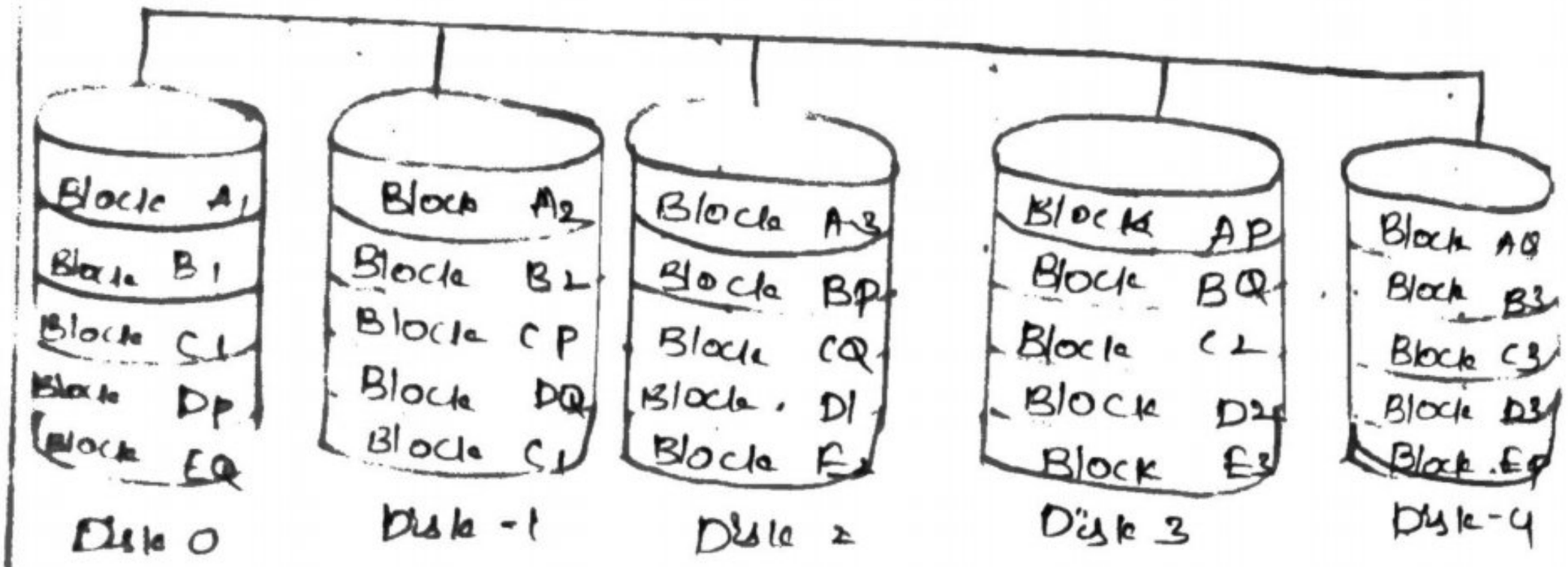
RAID-5 :-

This level is based on parity block-level Stripping. The parity information is stripped across each drive, enabling the array to function even if one drive were to fail. The array's architecture allows read and write operations to span multiple drives resulting in performance better than that of a single drive but not as high as that of a RAID 0 array. RAID 5 require atleast three disks, but it is often implemented to its at least five disk



RAID - 6

This technique is similar to RAID 5, but it includes a second parity scheme distributed across the drives in the array. The use of additional parity enables the array to continue to function even if two disk fail simultaneously. However, this extra protection comes at a cost. RAID 6 arrays often have slower write performance than RAID 5 arrays.



Disk Scheduling Algorithms

1. FCFS :-

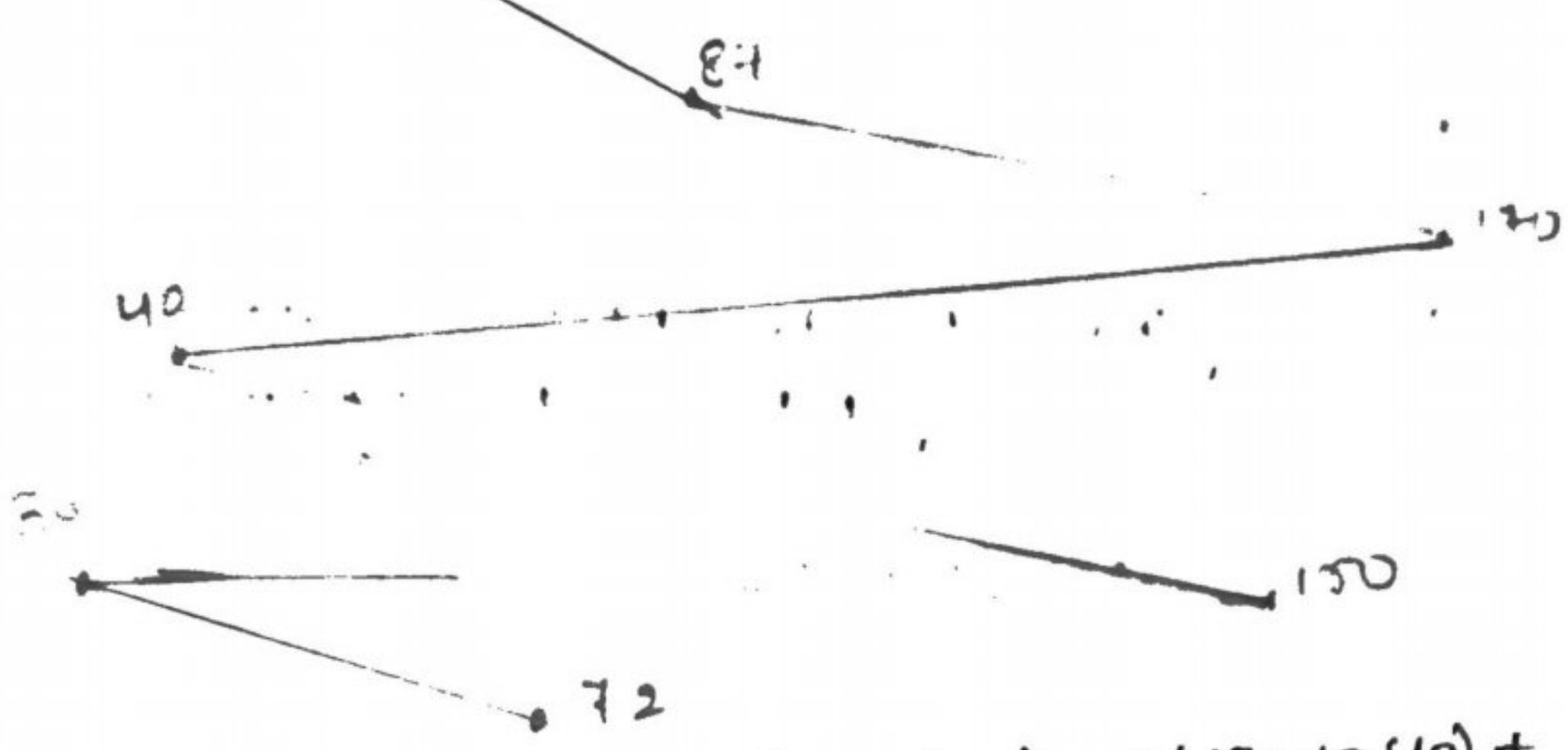
* FCFS stands for first come first serve FCFS disk scheduling algorithm is based on header value and blocks of string..
* According to FCFS disk scheduling algorithm:
  - Available string information required to arrange either ascending order (or) Descending order.

* This algorithm header movement starts with forward direction.
* Finally calculated no. of cylinders with help of using formula :-

$$\text{Total deviations bld header value \& blocks of string.} \over \text{Total blocks of string.}$$

* Consider a string 87, 170, 40, 150, 36, 72 Header value is 60.

0 20 40 60 80 100 120 140 160 180



$$\text{No. of cylinder} = \frac{D(60 \text{ to } 87) + D(87 \text{ to } 170) + D(170 \text{ to } 40) + D(40 \text{ to } 150) + D(150 \text{ to } 36) + D(36 \text{ to } 72)}{6}$$

$$= \frac{27 + 83 + 130 + 110 + 114 + 36}{6}$$

$$= 83.33 \text{ cylinders}$$

2. SSTF :-

SSTF stands for Shortest seek time First.

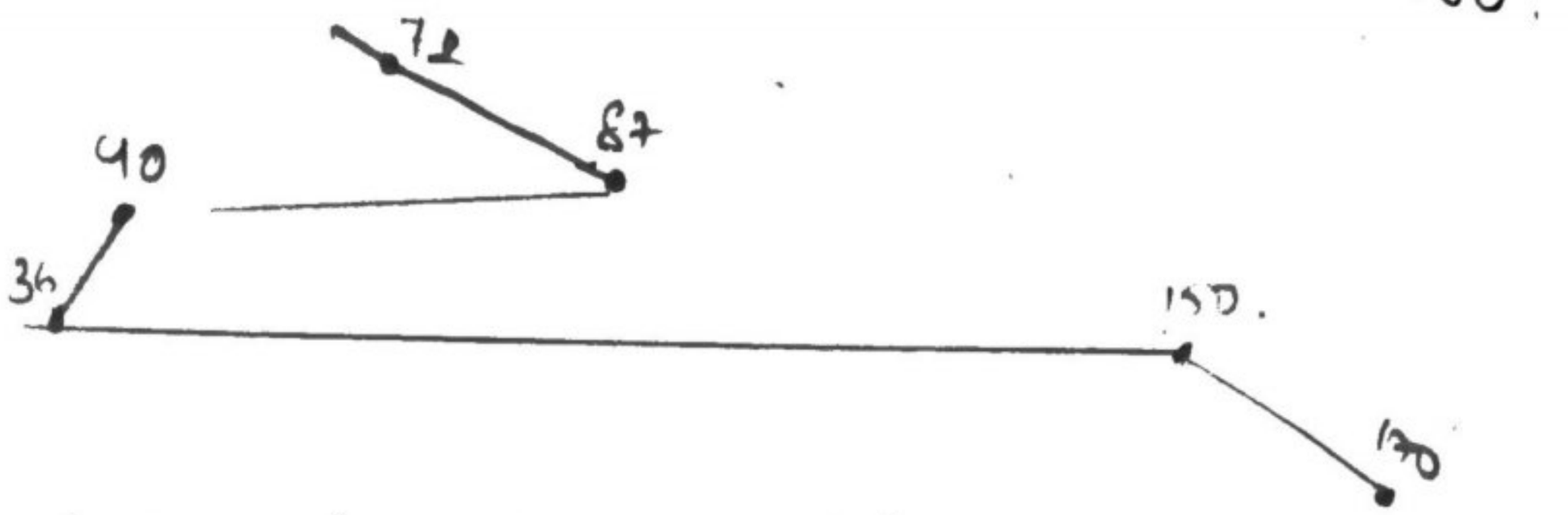
* SSTF initially required string (string contains no. of blocks)
* SSTF implementation based on seek time.
* Seek time means to find out difference b/w header value and blocks of string.
* According to SSTF, needed to arrange string information either ascending or descending order.
* Header movement starts with seek time value.

* finally calculated no. of cylinders using mathematical formula :

$$\text{Total deviations b/w header value \& blocks of string} \over \text{Total blocks of string.}$$

2. Consider a string 87, 170, 40, 150, 36, 72 Header value 4 60.

0 20 40 60 80 100 120 140 160 180



No. of cylinders = $\dfrac{D(60\text{to}72) + D(72\text{ to }87) + D(87\text{ to }40) + D(40\text{to }36) + D(36\text{ to }150) + D(150+170)}{6}$

$= \dfrac{12+15+47+4+114+20}{6}$

$= 35.33$ cylinders.

Scan Algorithm :-

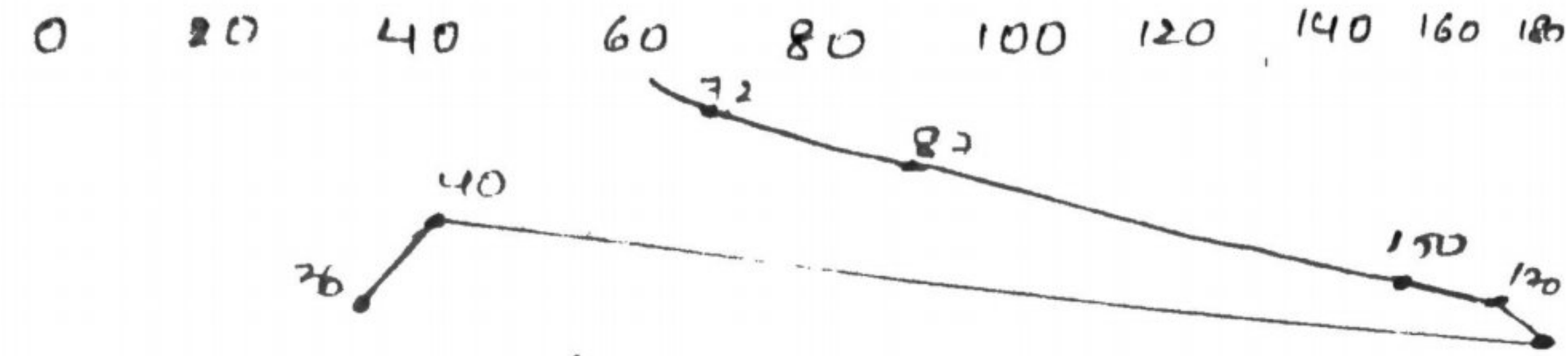
* It is a disk scheduling algorithm.
* Intially requires string value and header value.
* Scan algorithm is implemented based on header value and blocks of the string.
* According to scan algorithm intially header value, Moves either forward and backward Ends.
* String is a collection of blocks of disk (or) gathering of blocks of string.

Ex:

* Consider a string 87, 170, 40, 150, 36, 72.

According to scan algorithm need to arranging of

either according order (or) otherwise order.

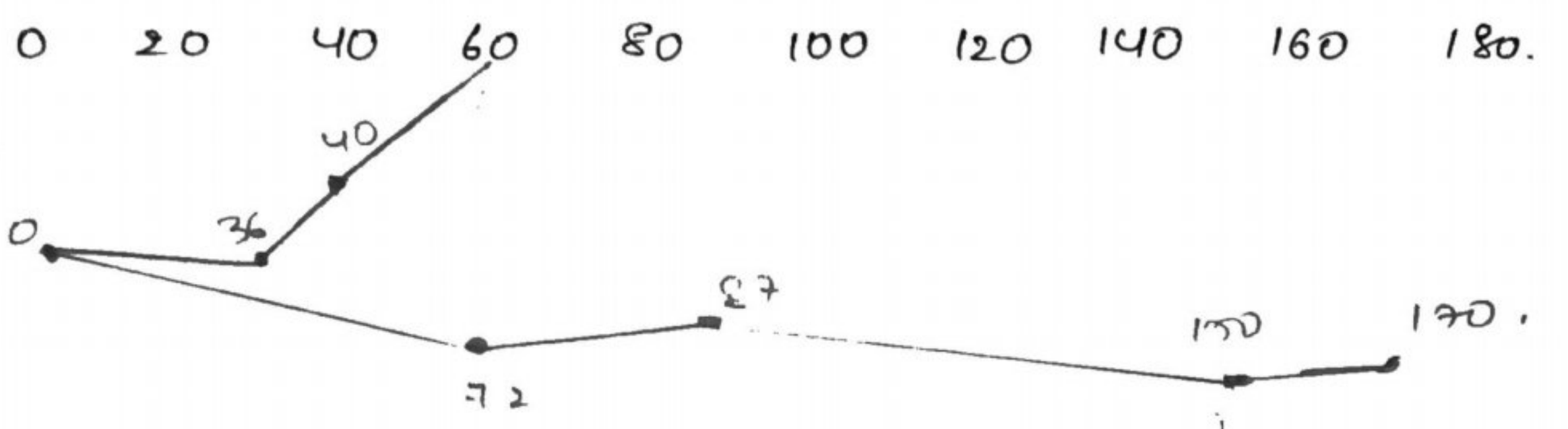


No. of cylinders = $\frac{D(60 \text{ to } 72) + D(72 \text{ to } 87) + D(87 \text{ to } 150) + D(150 \text{ to } 170) + D(170 \text{ to } 180) + D(80 \text{ to } 40) + D(40 \text{ to } 36)}{6}$

$= \frac{12 + 15 + 63 + 20 + 10 + 140 + 4}{6}$.

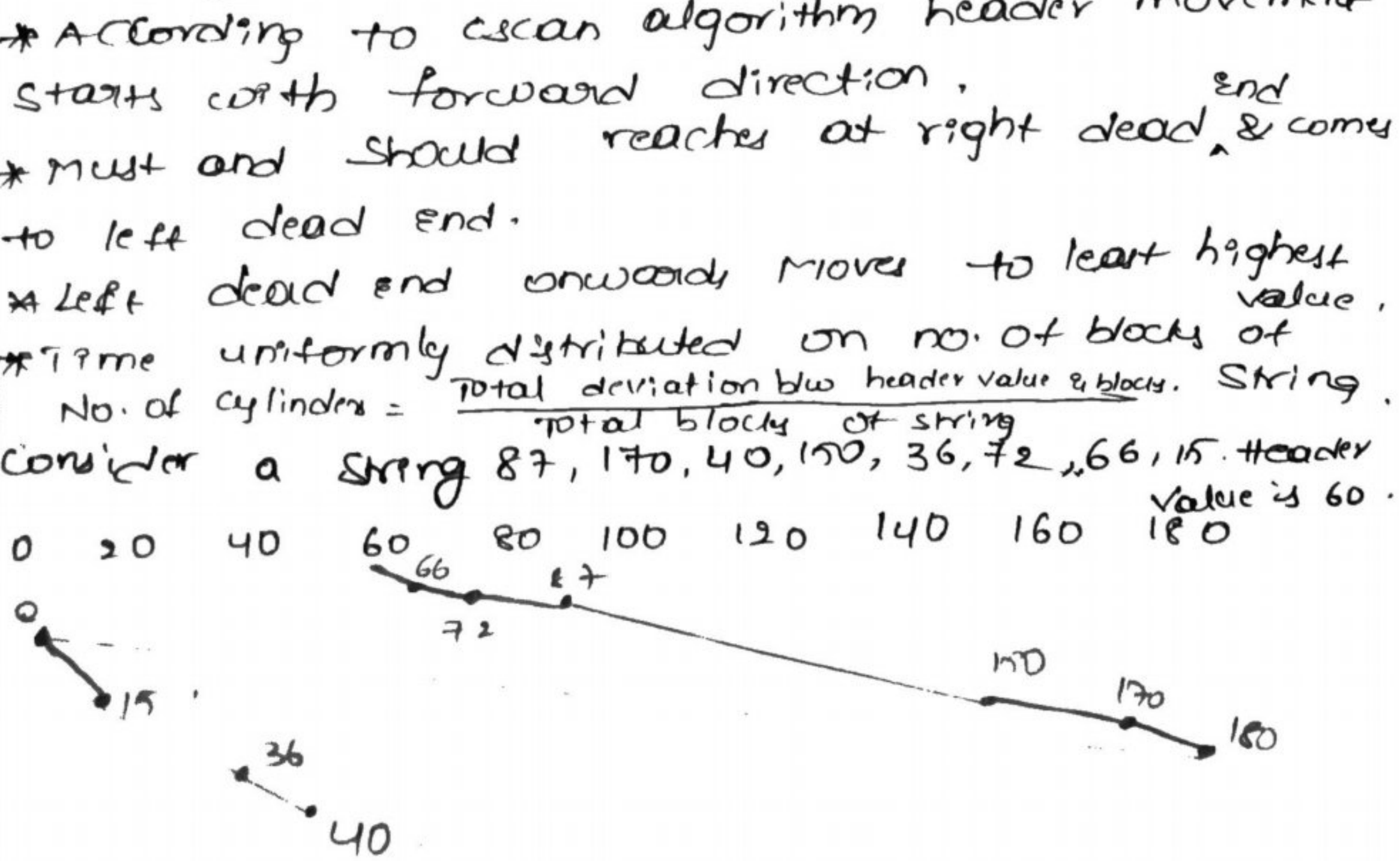
= 44 cylinders.

Left Backend



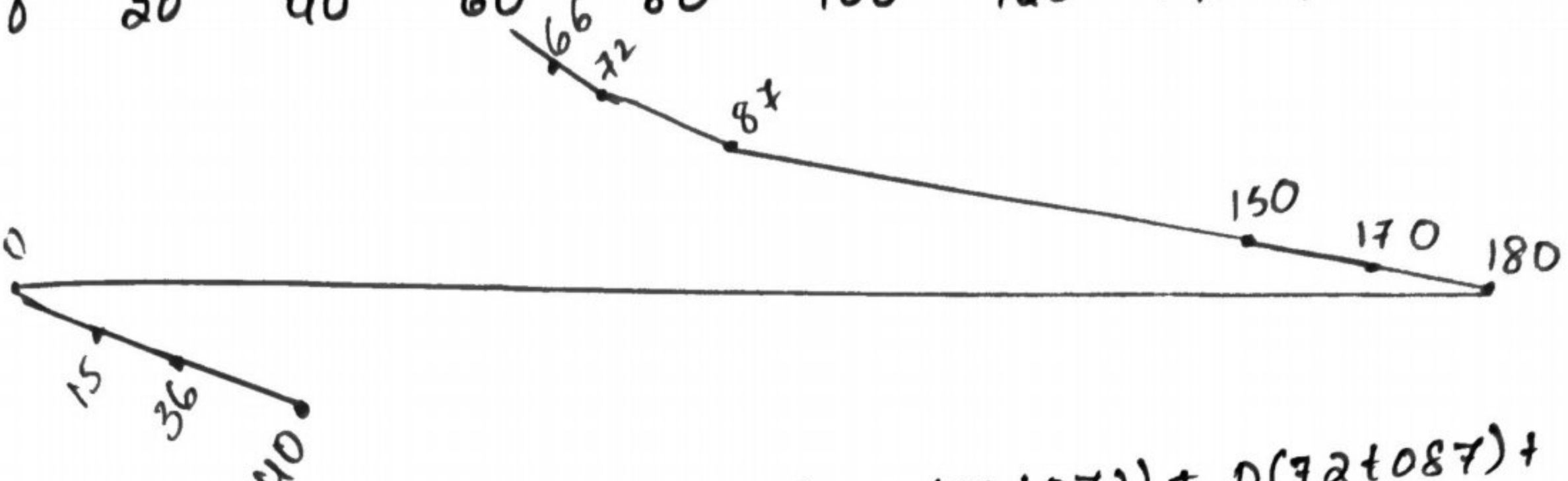
C Scan Algorithm :-

* Initially requires string (contains no. of blocks)
* According to cscan algorithm header movement starts with forward direction.
* must and should reaches at right dead end & comes to left dead end.
* Left dead end onwards moves to least highest value.
* Time uniformly distributed on no. of blocks of String.
* No. of cylinders = $\frac{\text{total deviation blw header value \& blocks.}}{\text{total blocks of string}}$
* Consider a String 87, 170, 40, 150, 36, 72, 66, 15. Header value is 60.



string.

consider a string 87, 170, 40, 150, 36, 72, 66, 15. Header value is 60.



No. of cylinders = $\frac{D(60\text{ to }66) + D(66\text{ to }72) + D(72\text{ to }87) + D(87\text{ to }150) + D(150\text{ to }170) + D(170\text{ to }180) + D(180\text{ to }0) + D(0\text{ to }15) + D(15\text{ to }36) + D(36\text{ to }40)}{8}$

$= \frac{6 + 6 + 15 + 63 + 20 + 10 + 180 + 15 + 21 + 4}{8}$

$= 42.5$ cylinders.

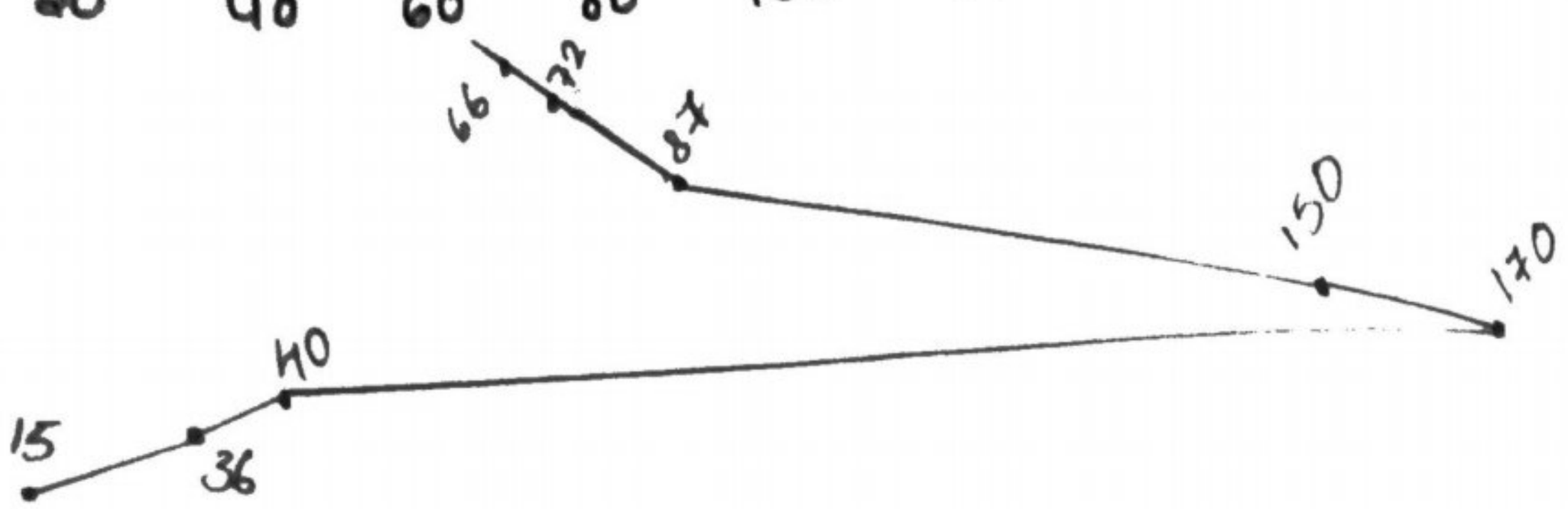
5. Look Algorithm:

- Initially require string (contains no. of blocks).
- According to LOOK Algorithm is required to arrange into queue order or ascending order.
- It is based on header value and forward direction.
- According to LOOK Algorithm, header value moves to right side highest queue value and covering all intermediate blocks.

• Later come back to left least value and covering all intermediate blocks.

Consider a string 87, 170, 40, 150, 36, 72, 66, 15.

Header value is 60.



No. of cylinders = $\frac{D(60 \text{ to } 66) + D(66 \text{ to } 72) + D(72 \text{ to } 87) + D(87 \text{ to } 150) + D(150 \text{ to } 170) + D(170 \text{ to } 40) + D(40 \text{ to } 36) + D(36 \text{ to } 15)}{8}$

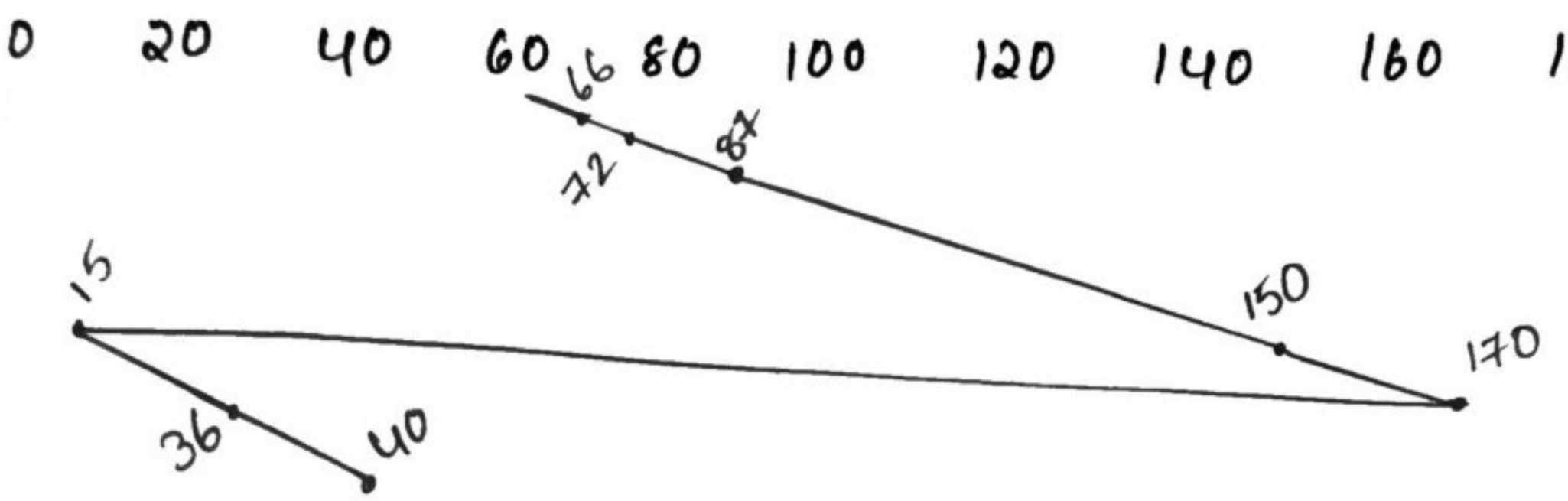
$= \frac{6 + 6 + 15 + 63 + 20 + 130 + 41 + 21}{8}$

$= 33.125$ cylinders.

6. C LOOK Algorithm:

• Initially require string (contains no. of blocks)
• According to CLOOK algorithm is required to arrange into queue order or ascending order.
• It is based on header value and forward direction.
• According to CLOOK Algorithm, header value moves to right side highest queue value and covering all intermediate blocks.
• Later come back to left least value without covering all intermediate blocks.
• Starts least value to least highest value.
• Finally calculate no. of cylinders.

consider a string 87, 170, 40, 150, 36, 72, 66, 15.
Header value is 60.



No. of cylinders = D(66 to 72) + D(72 to 87) + D(87 to 150) + D(150 to 170) + D(170 to 15) + D(15 to 36) + D(36 to 40)

$$= \frac{6 + 6 + 15 + 63 + 20 + 155 + 21 + 4}{8}$$

$$= 36.25 \text{ cylinder}$$

36.25