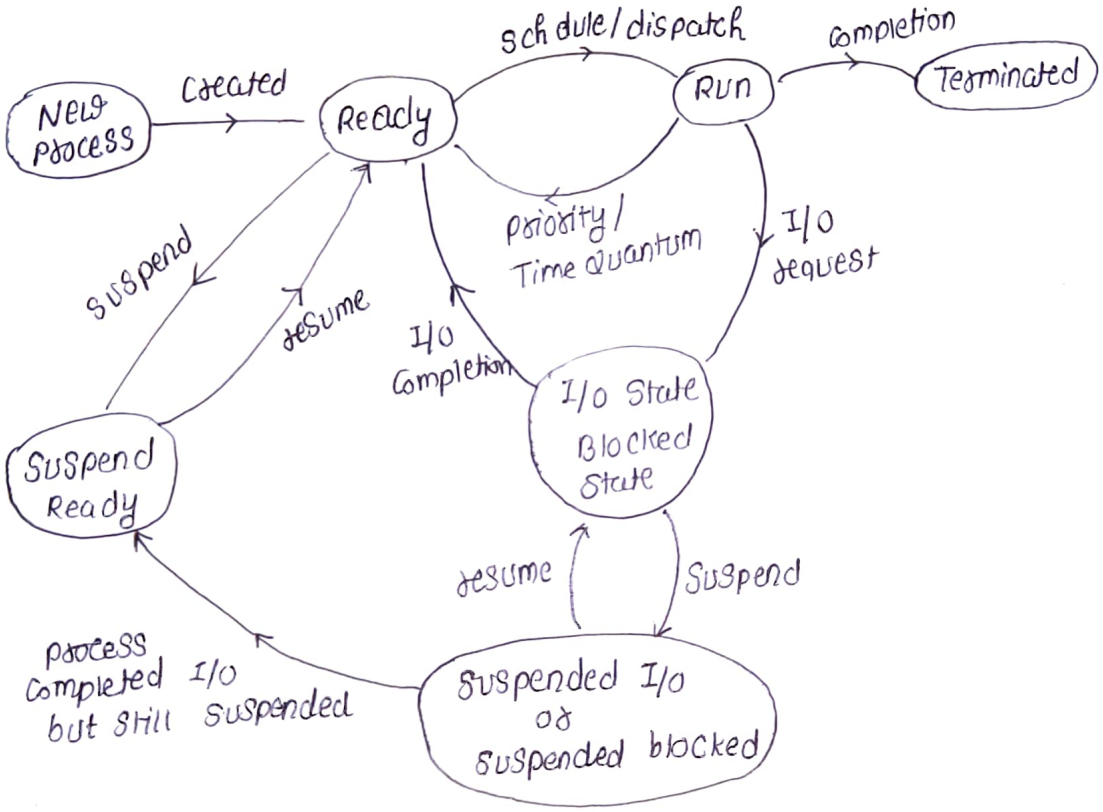
Solution 1: As given in question the program has its last instruction an I/O statement. Then the corresponding process will be in waiting state.

~~As~~ No, the corresponding process cannot jump directly to terminated state.

It can be understood by figure given below:-

Process state Diagram.



If the running process requires I/O option. It will move on wait or block state. If the waiting processes nothing more to execute then it goes to Suspend wait state. From Suspend wait state it goes to Suspend ready state. From that state, after Completion of the i/o process (given process) then the corresponding process acquires ready and after that the states changing then goes to terminated.

Solution 2> No.

The three Contions required are:-

1. Mutual exclusion.
2. Bounded waiting.
3. progress.

The processes can get into deadlock hence not satisfying bounded waiting, and execute an infinite boundry waiting, i.e. progress is not satisfied.

Consider the Scenario:-

i> P_1 executes, sets wants1 = true.

ii> Interrupt, P1 preempted.

iii> P_3 next runs, sets wants3 = true.

iv> Now since the assignment and loop Combined are not atomic operations, it may happen ps is preempted now,

v> P2 next runs, sets wants2 = true.

A solution to this can be from "peterson's solution

i.e. uses an wants and a turn variable

on the enter region and exit region code can be.

Entry

```
int other = 1 - user process.        //opposite
wants[user process] = true.
turn = Curr- process.
While (turn == Curr- process) && wants[other] == true );
```

Soln 2> Continue...

// Critical region

Exit { wants[Curr-process] = false; }

where Curr-process can be 0, 1 or 2.
(for P_1, P_2, & P_3 respectively)

This satisfies mutual exclusion, since a process waits while other is interested, or the other present can execute and then leave stating it doesn't want to enter now (false).

It satisfies progress, since eventually the other process make it turn and wants, so the other complete, then it can progress into critical region and complete.

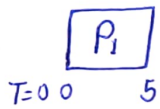
It satisfies bounded waiting at each process gets to enter and can set the wants array for it's turn.

Solution 3) Given,

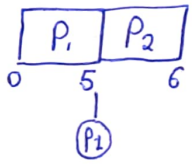
	Total execution	First execution	I/O	second exe.	I/O
P_1	10	$\frac{10 \times 50}{100} = 5$	1	2	2
P_2	20	$\frac{20 \times 50}{100} = 10$	2	4	4
P_3	30	$\frac{30 \times 50}{100} = 15$	3	6	6

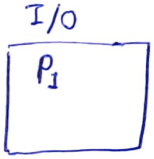
So Gantt chart will look like :- Shortest remaining compute time first :-

1.)

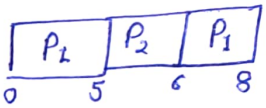


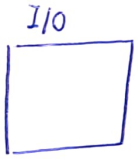
2.)



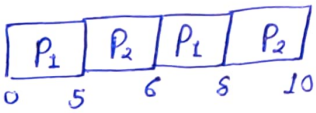


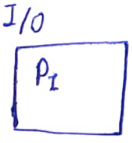
3.)



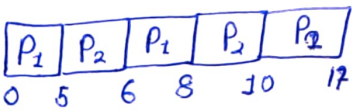


4.)

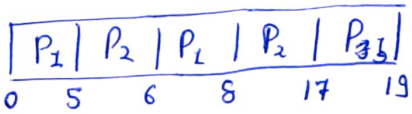


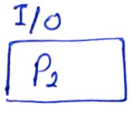


5.)

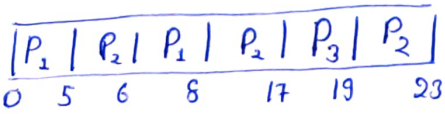


6.)

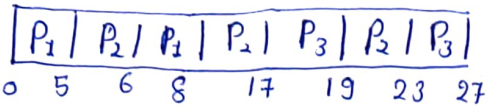




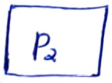
7.)



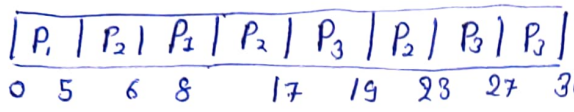
8.)



I/O

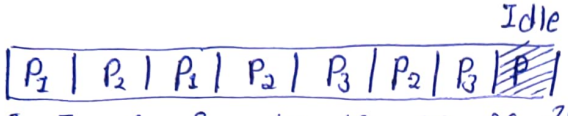


9.)



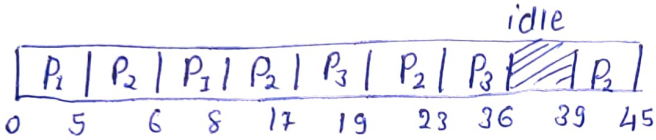
I/O

10.)



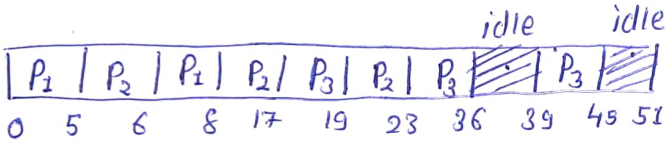
I/O

11.)



I/O

12.)



I/O

Total idle = 3 + 6 = 9 units

$$\% \text{ idle} = \frac{9}{51} \times 100$$

$$= 17.647$$

Solution4) i)

⇒ The entire disk is divided into platters

⇒ Each platter consists of Concentric circles called tracks

⇒ These tracks are further divided into sectors which are smallest divisions in the disk

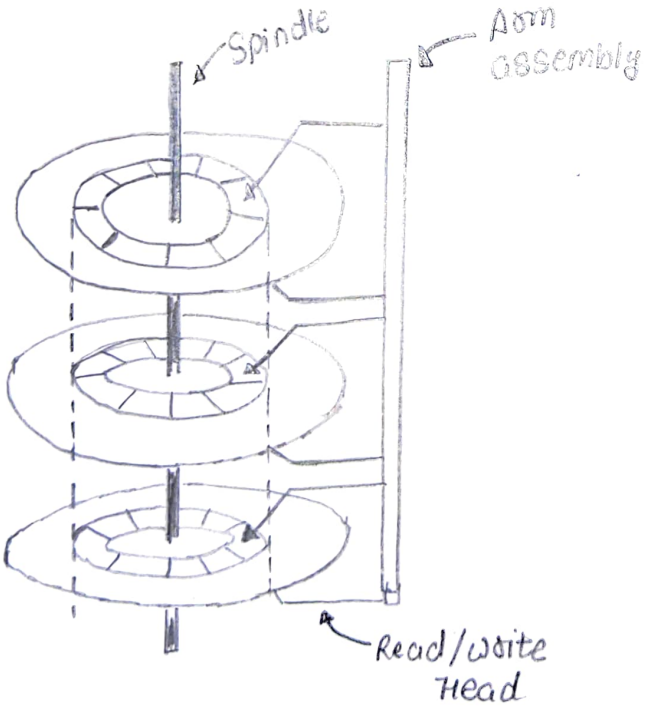
⇒ A Cylider is formed by combining the tracks at a given radius of disk pack.

⇒ There exists a mechanical arm called as Read/Write head.

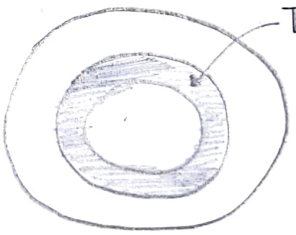
⇒ It is used to read and write to the disk.

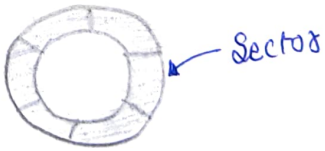
⇒ Head has to reach at the particular track then wait for rotation of platter.

⇒ The rotation causes the required sector of track to come under head.



Disk divided into tracks





Disk divi·

Track divided into sectors

Solution 4> i> The time taken by the disk to complete an I/O request is called disk Service Time.

Factors affecting disk service time are:-

Seek time:- It is the time taken by the arm to move the read/write head to desired track.

Rational Latency: Time taken by desired Sector to come under read/write head.

Data Transfer Rate: The amount of data that passes under read/write head in given amount of time is called data transfer rate.

Random Access Time: Sum of Seek time and rotational latency.

Controller overhead:- The overhead imposed by the disk controller

Queuing Delay:- Time spent waiting for disk to become free.

* Time Components affected by disk Scheduling algorithms are as follows:-

1. Seek time: disk scheduling algorithm that gives minimum average seek time is better.

2. ~~Rotational Latency~~: ~~disk shechedulling algorithm that gives minimum rotational latency is better.~~

3. Disk Response time: Response time is average of time spent by request waiting to perform its I/O operation. Response time can be affected by disk scheduling algorithms.

Solution: 4) ii) A multitasking operating system may just switch between processes to give the appearance of many processes executing simultaneously (that is, in parallel), though in fact only one process can be executing at any one time on a Single Core CPU.

As we know, in running state of Single Core CPU only one process can run, many processes will present in ready queue. They are scheduled on CPU with the use of scheduling algorithm using context switching.

For I/O operation they reside in waiting state because in runtime only one process can reside in running state.

Solution 5>: Given memory references:-

Hex	Decimal
0x3629	13685
0x7535	30005
0x7436	29750
0x6037	24631
0x8364	33636
0x379A	14234
0x52AA	21162
0x36BB	14011
0x336D	13165
0x432C	17196
0x89EE	35310
0x3629	13865
0x0001	00001
0xF034	64820

4kB page/frame size = 4×2^{10} Bytes

So, assuming byte addressable system.

ii> By optimal algorithm.

1> 13685 → 1st page fault

13685	–	–

Similarly

2> 30005 → 2nd page fault

3> 29750 → 3rd page fault

4> 24631

29750 - 24631 > 4kB ⇒ 4th page fault

Similarly, 30005 Cannot be replaced

$33636 - 30005 < 4kB$

So 29750 should be replaced.

5) $|30005 - 33636| < 4kB$ No page fault.

6) 14234

$|14234 - 13685| < 4kB$ no page fault

7) 21162

$|21162 - 24631| < 4kB$ no page fault

8) 14011

$|14011 - 13685| < 4kB$ no page fault

9) 13165

$|13165 - 13685| < 4kB$ no page fault

10) 17196

$17196 - 13685$

$= 3511 < 4kB$ no page fault

| 13685 | 30005 | 24631 |

11) 35310

$35310 - 30005 > 4kB$

5th page fault

| 13685 | 35310 | 24631 |

12> 13865

|13865 - 13365| < 4kB no page fault

13> 00001

Definitely page fault will occur hence 6th page fault

14> 64820

7th page fault.

Hence Allocated frame will look like

| 13685 | 00001 | 64820 |

iii> Solution : Size of physical address space = 2^{16} B.

(∵ 16 bits are used in physical memory).

[Physical address space = 64kB]