A1:- Time 23:-

Process	P1	P3	P5	P7	P8
state. (which it tries to follow)	Read from disk unit 3. (R3)	R2	Exits (time expires)	W3	suspended Not given

Let Read from disk unit i = R_i & Write to disk unit i = W_i

So, at time 23, P1 is in ready state & reading from disk unit 3.
P3 is also in ready state & it is reading from disk unit 2.
P5 has exited from mem its time expired, so it is suspended.
P7 is in waiting state as it is waiting for P1 to leave resource R3 so that it can write to disk unit 3.
P8 is suspended.
→ Here P7 is waiting or in blocked state and is waiting for relinquish of disk unit 3 by P1.

Time 37:-
P1 read is completed, so its suspended.
P3 → ~~ready state & reading from disk unit 3.~~ read is completed, so suspended.
P5 → swapped out, so exited.
P7 → ready state & writing to disk unit 3.
P8 → still suspended.

Time 47:-
P1 → suspended
P3 → suspended.
P5 → As P5 was swapped, its interrupt at time 40 was for invalid transaction, so it will be put in queue for writing to disk unit 3 which is still held by P7, so it stays blocked.
P7 → ready state & writing to disk unit 3
P8 → Exited

Matrix:-

	P1	P3	P5	P7	P8
22	Ready.	Ready	Suspended	Blocked (P1 resource)	suspended
37	Suspended	suspended	Exited	Ready	suspended
47	suspended	suspended	Blocked (P7 resource)	Ready.	Exited

1.6

Ans- → Optimal is the best page-replacement algorithm that we can have. So, it gets '1'. ✓

→ LRU (inefficient in implementation not ~~inefficient~~) ✓

→ LRU is the second best algorithm although we don't implement it due to constraints (efficiency).

→ 2nd-chance replacement is an approximate algorithm for LRU which is implemented using FIFO stack so its the next best option.

→ FIFO is the worst among the given options.

Considering a scale of 1 for perfect & 5 for bad, we can say that none of them are close to optimal in real-world scenario & the rating is

Optimal → 1

LRU → 2

2nd-chance → 3

FIFO → 5

10

FIFO performs worst in these & it also suffers from Belady's anomaly among these replacement algorithms.

How about rest of the algorithms??

A3:- String : 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

LRU → replaces least recently used page.

FIFO → ~~replaces~~ use the first in first out principle

Optimal → Uses future view to replace the page which will occurs the farthest in future.

Algo \ Frame size	1	2	3	4	5	6	7
1.) LRU	20	18	15	10	8	7	7
2.) FIFO	20	18	16	14	10	8	7
3) Optimal	20	15	11	8	7	7	7.

(12)

Ans:- Given semaphore variables S & R, so a semaphore variable is acquired using semWait & released using semSignal.

```
foo(){
 do{
  1 -> semWait(S);
  2 -> semWait(R);
  3 -> x++;
  4 -> semSignal(S);
  5 -> semSignal(R);
  }while(1);
}
```

```
bar(){
 do{
  1 -> semWait(R);
  2 -> semWait(S);
       x--;
       semSignal(S);
       semSignal(R);
 }while(1);
}
```

a.) Consider line 1 of both functions. So, foo() acquires S & bar() acquires R, if such a thing happens during the concurrent execution of foo() & bar(), neither foo() will be able to complete its line 2 nor bar() will be able to complete its line 2. foo() wants R which is held by bar() & bar() wants S which is held by foo(). So, a deadlock occurs here & both processes are being blocked forever until one of them is forced to relinquish the semaphore. (12)

b.) No, concurrent execution of these 2 processes can't result in indefinite postponement of one of them. Definite postponement of either will happen. e.g. Consider foo() has reached line 3. but bar() is at line 1 (not able to complete the instruction, but foo() will keep going and as soon as 4 & 5 line of foo() is completed, bar() will acquire R and S and so bar() will be executed & foo() will wait. Infinite postponement of either func will not occur, although both together can be blocked forever as in part (a).

26, 37, 100, 14, 88, 33, 99, 12.

Ans:- a.) FCFS :-

Total movement = (37-26) + (100-37) + (100-14) + (88-14) + (88-33) + (99-33) + (99-12)

= 11 + 63 + 86 + 74 + 55 + 66 + 87

= 442.

b.) SSTF :-

Total movement = { 26 → 33 → 37 → 14 → 12 → 88 → 99 → 100

= 7 + 4 + 23 + 2 + 76 + 11 + 1

= 124.

c.) SCAN (going up) :- (Assuming cylinder length of 1 to 100).

Total movement = 26 → 33 → 37 → 88 → 99 → 100 → 14 → 12

= 7 + 4 + 51 + 11 + 1 + 86 + 2

= 162.

d.) C-SCAN (going up) :- (Assume cylinder length of 1 to 100). Total length = 100.

Total movement = 26 → 33 → 37 → 88 → 99 → 100 → 1 → 12 → 14

(went to beginning)

= 7 + 4 + 51 + 11 + 1 + 99 + 11 + 2

= 186.

Note, if we don't consider movement from one end to other, i.e., from 100 to 1 cylinder, the movement of 99, then total movement = 186 - 99 = 87.

(Considering going up means increasing cylinder value, i.e from 26 → 100)

A6:- Rotation speed = 15000 rpm

Avg. rotation latency $= \frac{1}{2}\left(\frac{60}{15000}\right)$ s

$= \frac{1}{2} \times \frac{1}{250} \times 1000$ ms $= 2$ ms.

b.) Avg. access time = Avg. seek time + avg rotation latency

$= 4$ ms $+ 2$ ms $= 6$ ms

Now, Tracks = 1000
Sectors / track = 400

bytes / sector = 512

File size = 1 MByte

$= (1024)^2$ Byte

$= 2^{20}$ Bytes $= 2^{20} \times 8$ bytes

$= 2^{23}$ bytes.

c.) Total transfer time :-

Total bytes in a track = 512×400 bytes

$= 2^9 \times 2^2 \times 100$ bytes

$= 2^{11} \times 100$ bytes $= 25 \times 2^{13}$ bytes.

Total tracks required $= \frac{2^{23}}{25 \times 2^{13}} = \frac{2^{10}}{25} = 40.96$

∴ Total of 41 tracks will be required for contiguous allocation on disk out of which 40 will be filled completely & 41th track will be partially filled.

Total transfer time = Avg. seek time to access 41 tracks
+ rotation time to completely access 40 tracks
+ rotation time for partial access to 41 tracks

$= 4$ ms $\times 41 + 2$ ms $\times 40 + 2$ ms $\times 0.96$ [as 41th track only 0.96 of it needs to be written).

$= 164$ ms $+ 160$ ms $+ 1.92$ ms

$= 325.92$ ms

(Assuming avg. seek time denotes avg. time to seek from one track to another).

c.) Rotational delay is already calculated = 2ms.

d.) Total time to read 1 sector = Avg. seek time to a track + time to read a sector on a track.

$= 4\,ms + \frac{1}{400} \times 2\,ms = 4\,ms + \frac{1}{200}\,ms$

$= 4.005\,ms$

e.) Total time to read 1 track = Avg. access time + Avg. rotation delay

$= 4\,ms + 2\,ms = 6\,ms.$

Ans:- Since a direct pointer can hold a block of 8KB, an indirect pointer can hold a block of (8×8) KB = 64 KB, double indirect can hold $(8 \times 8 \times 8)$ KB = 512 KB & triple indirect pointer can hold $(8 \times 8 \times 8 \times 8)$ KB = 4096 KB.

∴ Total (max) file size the give i-node can hold

$= 13 \times 8\,KB + 64\,KB + 512\,KB + 4096\,KB$

$= (104 + 64 + 512 + 4096)\,KB$

$= 4776\,KB.$

(6)

why 16 ??

Total size $= 13 \times 8\,KB + 1024 \times 16 \times 8\,KB + (1024 \times 16)^2 \times 8\,KB + (1024 \times 16)^3 \times 8\,KB$

(∵ 32-bit pointer → 8 KB = $\frac{8 \times 8 \times 8 \times 1024}{32} = 1024 \times 16$)

$\frac{8\,KB}{32b} = \frac{8 \times 1024 \times 8}{32} = 256 \times 8$

$= 2048$

$=$

A8:- 1.) The target can keep ID of all the challenges incoming connections so that are being given to others or is getting from others. So, there can be many challenges connections corresponding to different connections with the target. If any challenge is repeated (of asked challenges and being challenges asked to target), break the connection with that. connect

16

Now, it may happen that the target has got same ~~asked~~ challenge from another ~~atta~~ connection which he has asked to attacker. In such a case, we don't want to break the connection. So, we will let the target answer the challenge of valid / invalid connection wrong whenever same challenge appears from other side. If we get same answer as reply to our same challenge, then its attacker & so the connection be broken, else if we get correct response, we create connection.

→ Another approach is to keep ~~to~~ track of number of connections between any pair, ~~of~~ and simply reject more than one connection b/w a pair.

This won't work

→ OR ~~Be~~ Allow more than one connection b/w a pair and keep track of questions asked, if question is same, then reject the connection. The target will be determined in such a case by looking at timestamp and who asked the same question first.

END