MADE EASY
India's Best Institute for IES, GATE & PSUs

Delhi | Bhopal | Hyderabad | Jaipur | Pune | Bhubaneswar | Kolkata

Web: www.madeeasy.in | **E-mail:** info@madeeasy.in | **Ph:** 011-45124612

COMPUTER NETWORK

COMPUTER SCIENCE & IT

Date of Test : 23/10/2023

ANSWER KEY ➤

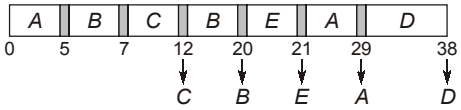
1.	(a)	7.	(c)	13.	(c)	19.	(c)	25.	(b)
2.	(d)	8.	(b)	14.	(d)	20.	(c)	26.	(d)
3.	(d)	9.	(b)	15.	(d)	21.	(c)	27.	(a)
4.	(d)	10.	(d)	16.	(d)	22.	(c)	28.	(c)
5.	(b)	11.	(c)	17.	(b)	23.	(d)	29.	(b)
6.	(b)	12.	(b)	18.	(d)	24.	(d)	30.	(a)

DETAILED EXPLANATIONS

1. (a)

Process	A_t	B_t	Priority
A	0	13	4
B	5	10	2
C	7	5	1
D	10	9	5
E	12	1	3

Preparing the Gantt Chart,



So, there are total 6 number of context switches.

2. (d)

(a) Round robin works on time quantum, after certain period of time every process gets the CPU unit for its completion, hence it's most suitable.

(b) Since OS is multiuser and multiprocessing, hence security is the primary concern so that user processes and Kernel processes can be isolated.
Hence two modes are required.

(c) When CPU temperature is too high, the BIOS initiate an interrupt. OS given top priority to this interrupt.

(d) Address translation table need to be changed when switching context from process A to process B.

3. (d)

1. Since, each process has its own address space, it needs to involve the Kernel when dealing with other process address space.
2. A software interrupt is required to switch between the two modes.
3. In both synchronous and asynchronous I/O an ISR is invoked after completion of the I/O.
4. Statement is correct.

4. (d)

- Switching between two user level threads only require procedure calls not context switching.
- All Kernal threads operations are implemented in Kernal, and OS schedules all threads in the system.
- Since user level threads are transport to Kernal, hence are not scheduled independently and hence are not given independent time slice.
- Threads do share the code segment.

5. (b)

Calculating the need matrix

	A	B	C	D	E
P_0	0	1	0	0	2
P_1	0	2	1	0	0
P_2	1	0	3	0	0
P_3	0	0	1	1	1

Since, available = 00123, hence only P_3 can be satisfied.
Remaining = (00123) – (00111) = (00012) + (11221) = (11233)
Now P_0 can be executed,
Remaining = (11233) – (01002) = (10231) + (11213) = (21444)

Now P_2 can be executed,
Remaining = (21444) – (10300) = (11144) + (21310) = (32454)
Now P_1 can be executed.

6. (b)

LRU :

	1	2	3	2	3	4	1	4	2	3	2	1	4	2	3	1
		2	3	H	H	3	3	H	2	2	H	2	2	H	2	2
			2			2	1		1	3		3	4		4	1
	1	1	1			4	4		4	4		1	1		3	3

Total page faults = 11

FIFO :

	1	2	3	2	3	4	1	4	2	3	2	1	4	2	3	1
		2	3	H	H	3	3	H	2	2	H	H	2	H	H	1
			2			2	1		1	1			4			4
	1	1	1			4	4		4	3			3			3

Total page faults = 9

Optimal :

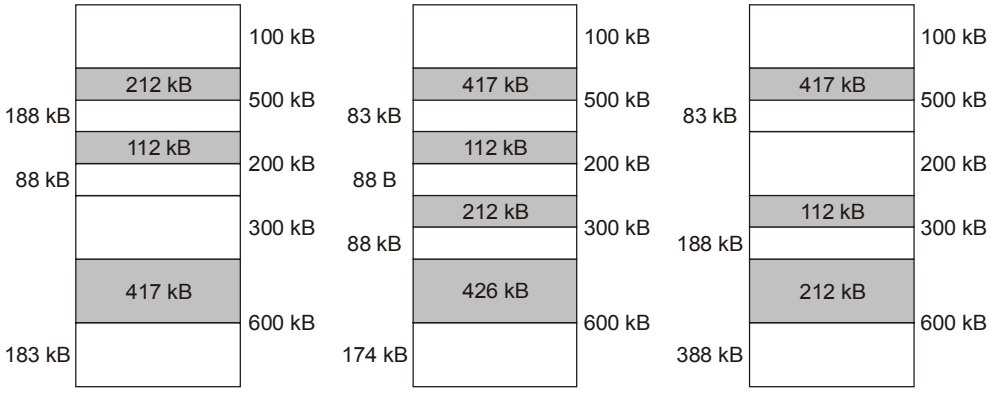
	1	2	3	2	3	4	1	4	2	3	2	1	4	2	3	1
			3	H	H	4	H	H	H	3	H	H	3	H	H	3
		2	2			2				2			2			1
	1	1	1			1				1			4			4

Total page faults = 7

7. (c)

1. CPU senses interrupt request line after every instruction.
2. Nearest cylinder next disk scheduling strategy gives the best throughput but the only problem is it can lead to starvation.
3. Using large file block size in a fixed block size file system leads to better disk throughput but poor disk space utilization.

8. (b)



9. (b)

To enter the critical section, process P_i first sets flag [i] to be true set $S_1 = S_2$, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time. The S_1 will be set S_2 or $S_2 + 1$ at roughly the same time. Only, one of these assignment will last; the other will occur, but will be overwritten immediately.

10. (d)

Since 4 distinct page numbers are only to be accessed. Hence the best condition i.e., the condition with minimum number of page faults will be accessing all those elements repeatedly that are in the frame already, which will give maximum 4 page faults.

If, considered the worst case, it will be on every iteration, we are accessing the same element that has been removed from the frame, which will give 52 page faults.

11. (c)

1. As files are allocated and deleted, the free disk space is broken into little pieces, hence can lead to external fragmentation.
2. Linked-allocation can be used effectively only for sequential access file. To find the i^{th} block of a file. We must start at the beginning of that file and follow the pointers until we get the i^{th} block.
3. Statement is correct.

12. (b)

In order to ensure a deadlock free system,

Sum of resource needs < [Number of resources + Number of processes] < [7 + 10] < 17

Maximum value that can be used is 16.

13. (c)

1. Computer ( ) → p (mutex) → mutex = 0
   p (Q) → Q = 0
2. Science ( ) → p (Q) → process sleep
3. Computer ( ) → p (R) → R = 0
   v(Q) → Q = 1, science ( ) awake
4. Science → p(Q); Q = 0; p(R) → process sleep
5. Computer → v(mutex) → mutex = 1
   p(Q) → process sleep

Hence a deadlock.

14. (d)

1. The policy is a deadlock prevention policy, but can lead to starvation.
2. In deadlock prevention, one of the four condition for deadlock must not be satisfied. So, state even being safe can't led to successful request.
3. It will help in violating circular wait condition for deadlock.
4. Under deadlock avoidance, just the safe state need to be checked and hence is less restrictive deadlock prevention scheme.

15. (d)

	X	*Y*	*Z*	*W*
P_0	2	2	2	2
P_1	3	2	0	0
P_2	0	3	2	4
P_3	2	5	0	2
P_4	2	0	0	1

Since available is a 0 0 b, let's suppose a takes value 2 and b takes the value 1.
Available = 2 0 0 1
$P_4 \rightarrow$ Complete $\rightarrow$ Avail = (0000 + 6214) = 6214
$P_1 \rightarrow$ Complete $\rightarrow$ Avail = (6214) – (3200) = (3014) + (3512) = (6526)
$P_0 \rightarrow$ Complete $\rightarrow$ Avail = (6526) – (2222) = (4304) + (3242) = (7546)
$P_2 \rightarrow$ Complete $\rightarrow$ Avail = (7546) – (0324) = (7222) + (2775) = (9, 9, 9, 7)
$P_3 \rightarrow$ Complete $\rightarrow$ Avail = (9997) – (2502) = 7495
Hence, the system is in a safe state will value of a as 2 and value of b as 1.

16. (d)

Direct block addressing = $16 * 256 \Rightarrow 4$ KB

Single indirect block addressing = $\left[\frac{256}{8}\right] * 256\,B$

$= 2^5 * 2^8\,B$

$= 2^{13}\,B \Rightarrow 8$ KB

1 doubly indirect block addressing $\Rightarrow \left[\frac{256}{8}\right]^2 * 256\,B$

$\Rightarrow (2^5)^2 * 2^8\,B$
$\Rightarrow 2^{10} * 2^8\,B$
$\Rightarrow 256$ KB

1 triple indirect block addressing $\Rightarrow \left[\frac{256}{8}\right]^3 * 256\,B$

$\Rightarrow (2^5)^3 * 2^8\,B$
$\Rightarrow 2^{15} * 2^8\,B$
$\Rightarrow 2^{23}\,B$
$\Rightarrow 8$ MB

17. (b)

Page Size = 8 K
Offset bits = 13
Virtual Address = 64 bits
Remaining bits = 64 – 13 = 51 bits

Number of sets = $\frac{256}{4}$ = 64 = 6 bits

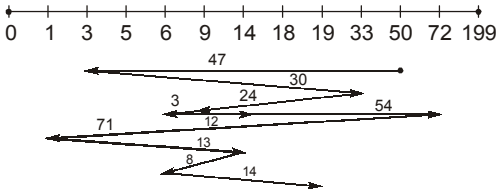
Tag bits = 51 – 6 = 45 bits

18. (d)

- The total size of address space in a virtual memory system is limited by the available secondary storage.
- Best fit technique can also suffer from fragmentation.
- Locality of reference implies that the page reference being made by a process is likely to be the page used in the previous page reference.
- In a system with virtual memory context switch includes extra overhead in switching of address space.

19. (c)

- FCFS



Total time = 47+30+27+12+54+71+13+8+14 = 276 msec

20. (c)

100 B

File size	100 B block	Bytes needed for Book keeping	Block for Information
4992	50	100	1
5172	52	104	2
11052	111	222	3
Total	**213**		**6**

Space = 213 × 100 + 6 × 100 = 21300 + 600 = 21900 B

200 B

File size	200 B block	Bytes needed for Book keeping	Block for Information
4992	25	50	1
5172	26	52	1
11052	56	112	1
Total	**107**		**3**

Space = 107 × 200 + 3 × 200 = 22000 B

21. (c)

Let 2^P be the page size.

Since page table entries are 4 bytes in size.

$$1^{st} \text{ Page table size} = \text{Page table entry} \times \text{Page table entry size}$$

$$= \frac{2^{32}}{P} \times 2^2$$

$$= 2^{34-P}$$

$$2^{nd} \text{ Page table size} = \text{Page table entry} \times \text{PTE size}$$

$$= \frac{2^{34-P}}{2P} \times 2^2$$

$$= 2^{36-P-P}$$

Last level page table must be fit into page size

So,

$$2^P = 2^{36-P-P}$$

$$P = 36 - P - P$$

$$3P = 36$$

$$P = 12$$

So page size will be 2^{12} bytes.

22. (c)

Given function compare and swap is like test and set. Or, we can say that test and set is just a special case of compare and swap, which maintain mutual exclusion and is deadlock free.

23. (d)

1. Statement is correct.
2. A page which was referenced last may also get replaced; although there is high possibility that the same page may be needed again since it ignores locality of reference.
3. The essential content in each entry of a page table are page frame number.
4. It is not feasible because of the large memory overhead in maintaining the page tables.

24. (d)

- $\text{Rotational latency} \Rightarrow 6000 \text{ rotation} \rightarrow 60 \text{ sec}$

  $1 \text{ rotation} \Rightarrow 1/100 \text{ sec}$

  $\text{Rotational latency} \Rightarrow \frac{1}{2} \times \frac{1}{100} = \frac{1}{200} \text{sec} = 0.005 \text{sec}$

  $\text{Transfer time} = 64 \text{ KB} \Rightarrow \frac{1}{100} \text{sec}$

  $1 \text{ KB} \Rightarrow \frac{1}{6400} \text{sec} = 0.000156 \text{ sec}$

  $\text{Data transfer rate} \Rightarrow \frac{1}{100} \text{sec} \rightarrow 64 \text{ KB}$

  $1 \text{ sec} \rightarrow 64 \text{ KB} \times 100$

  $\rightarrow 6400 \text{ KBPs}$
- Time required to read 800 random sectors

  $\text{Total time required} = [\text{Seek time} + \text{RL} + \text{TT (1 sector)}] \times 800$

  $= (0.005 + 0.005 + 0.000156) \times 800$

  $= 8.12 \text{ sec}$
- Total time = Seek time + Rotational latency + Transfer time

  $= 5 \text{ msec} + 0.005 \text{ sec} + 0.1248 \text{ sec}$

  $= 134.8 \text{ msec}$

25. (b)

If, we remove the lock while acquiring the fork. It may lead to deadlock, if all process execute (`i`) statement before any philosopher has execute (`ii`) statement.

Removal of (`iii`) and (`iv`) will not affect the code, since no conflict can occur doing the V operation on forks.

26. (d)

The output is 'TGE'. So, to print 'T', we must give a value of 1 to semaphore b and should block rest three processes.

Now, process 3, after printing T, will give signal to semaphore a, which will wake up process 1 and will print 'G' and given signal to semaphore 'b' and 'c'. On giving signal to semaphore 'c', process '2' will get awake. But 'a' should not be printed in the output hence 'c' should be given value '–1'.

Process 4 will also awake after process 3 on signal 'a', but it will again be blocked by wait (b).

27. (a)

Calculating the need matrix

Process	Need		
	X	*Y*	*Z*
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Since, the available resources are ⟨3, 3, 2⟩.

Hence the request can only be satisfied for P_1 or P_3 at initial stage.

Considering P_3 first,

Available after $P_3 \rightarrow \langle 3, 3, 2\rangle - \langle 0, 1, 1\rangle = \langle 3, 2, 1\rangle + \langle 2, 2, 2\rangle = \langle 5, 4, 3\rangle$

After P_3 only, P_1 or P_4 can be executed.

Considering P_1 first, rest all three processes can be scheduled in any way hence 6 possible ways.

After P_3, consider P_4, Next P_1 can only be scheduled, then P_0 and P_2 can be scheduled in any way hence 2 possible ways.

Considering P_1 now: Need after $P_1 \rightarrow \langle 3, 3, 2\rangle - \langle 1, 2, 2\rangle = \langle 2, 1, 0\rangle + \langle 3, 2, 2\rangle = \langle 5, 3, 2\rangle$

Now, condition can be satisfied either for P_3 or P_4.

Considering P_3 first, any possible combination on P_0, P_4 and P_2 possible hence 6 sequence.

Considering P_4 first, followed by P_3, then any combination of P_0 and P_1 hence sequence.

Total = 16 sequences.

28. (c)

Let's first analyze the wait and signal code;

Wait code: Initially value of C is 7, $S_1 = 1$, $S_2 = 0$

wait (S_1) $\rightarrow S_1 = 0$

C - -; $\rightarrow$ C = 6

If (value of C is less than 0, the thread is blocked by applying wait (S_2).

Hence, it can be seen that, it is the code for counting semaphore wait operation, implemented with the help of binary semaphore.

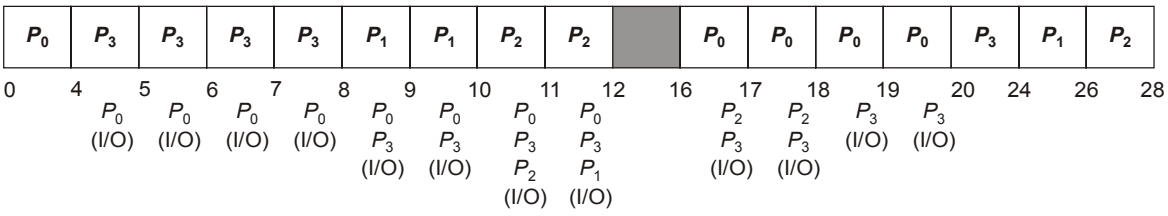
Similarly, for signal code

S = 7 – 5 = 2 + 3 = 5

29. (b)

Process	Burst Time	CPU	I/O	CPU	CT	TAT
P_0	20	4	12	4	20	20
P_1	10	2	6	2	26	26
P_2	10	2	6	2	28	28
P_3	20	4	12	4	24	24
						Average TAT = 98/4 = 24.5

Gantt Chart



30. (a)

Case-1: System accesses 200 distinct pages. So, all these 200 pages are the page fault, next these pages are accessed again, at that time page number 1, 2, 3 and 200 are in the frame. Now, when 4 will be accessed, it will be replaced by 1. Next when 5 will be accessed, it will also be replaced by 2 and so on till 199. So, total page faults = 200 + 196 = 396

Case-2: Again after the first access is over 197, 198, 199 and 200 are in the page frame. From 196 to 1 will be fault. So, total page faults = 200 + 196 = 396

Difference = 396 – 396 = 0

■■■■