(d) The value of counter is 5 after all the threads successfully complete the execution of parop.
[2021 (Set-1) : 2 M]

2.42 Consider a computer system with multiple shared resource types, with one instance per resource type. Each instance can be owned by only one process at a time. Owning and freeing of resources are done by holding a global lock (L). The following scheme is used to own a resource instance:

```
function OwnResource (Resource R)
    Acquire lock L // a global lock
    if R is available then
        Acquire R
        Release lock L
    else
        if R is owned by another process P then
        Terminate P, after releasing all
        resources owned by P
            Acquire R
            Restart P
            Release lock L
        end if
    end if
end function
```

Which of the following choice(s) about the above scheme is/are correct?

(a) The scheme violates the mutual exclusion property.
(b) The scheme ensures that deadlocks will not occur.
(c) The scheme may lead to live-lock.
(d) The scheme may lead to starvation.
[2021 (Set-2) : 2 M]

2.43 Consider the following multi-threaded code segment (in a mix of *C* and pseudo-code), invoked by two processes P_1 and P_2, and each of the processes spawns two threads T_1 and T_2:

```
int x = 0;          // global
Lock L1;            // global
main() {
  create a thread to execute foo();   // Thread T1
  create a thread to execute foo();   // Thread T2
  wait for the two threads to finish execution;
  print (x);}
foo() {
  int y = 0;
  Acquire L1;
  x = x + 1;
  y = y + 1;
  Release L1;
  print (y); }
```

Which of the following statement(s) is/are correct?

(a) At least one of P_1 and P_2 will print the value of x as 4,
(b) Both P_1 and P_2 will print the value of x as 2
(c) Both T_1 and T_2, in both the processes, will print the value of y as 1.
(d) At least one of the threads will print the value of y as 2.
[2021 (Set-2) : 2 M]

2.44 Consider the following threads, T_1, T_2 and T_3 executing on a single processor, synchronized using three binary semaphore variables, S_1, S_2 and S_3, operated upon using standard wait( ) and signal( ). The threads can be context switched in any order and at any time.

T_1	T_2	T_3
while (true) { wait (S_3); print ("C"); signal (S_2); }	while (true) { wait (S_1); print ("B"); signal (S_3); }	while (true) { wait (S_2); print ("A"); signal (S_1); }

Which initialization of the semaphores would print the sequence BCABCABCA.....?

(a) $S_1 = 1; S_2 = 1; S_3 = 1$
(b) $S_1 = 1; S_2 = 1; S_3 = 0$
(c) $S_1 = 1; S_2 = 0; S_3 = 0$
(d) $S_1 = 0; S_2 = 1; S_3 = 1$
[2022 : 1 M]

2.45 Consider the two functions incr and decr shown below:

```
incr() {                decr() {
    wait(s);                wait(s);
    X = X + 1;              X = X - 1;
    signal(s);              signal(s);
}                       }
```

There are 5 threads each invoking incr once, and 3 threads each invoking decr once, on the same shared variable *X*. The initial value of *X* is 10. Suppose there are two implementations of the semaphore s, as follows:

I-1: *s* is a binary semaphore initialized to 1.
I-2: *s* is a counting semaphore initialized to 2.

Let V1, V2 be the values of *X* at the end of execution of all the threads with implementations I-1, I-2, respectively.

Which one of the following choices corresponds to the minimum possible values of V1, V2 respectively?

(a) 15, 7
(b) 7, 7
(c) 12, 7
(d) 12, 8
[2023 : 2 M]

2.46 Consider the following code snippet using the fork( ) and wait( ) system calls. Assume that the code compiles and runs correctly, and that the system calls run successfully without any errors.

```
int x = 3;
while (x > 0) {
    fork( );
    printf("hello");
    wait(NULL);
    x--;
}
```

The total number of times the printf statement is executed is _____.
[2024 (Set-1) : 2 M]

Answers Process Management-II

2.1	(c)	2.2	(c)	2.3	(d)	2.4	(b)	2.5	(b)	2.6	(b)	2.7	(a)	2.8	(b)	2.9	(b)		
2.10	(b)	2.11	(c)	2.12	(d)	2.13	(d)	2.14	(d)	2.15	(b)	2.16	(a)	2.17	(b)	2.18	(b)		
2.19	(c)	2.20	(d)	2.21	(c)	2.22	(c)	2.23	(a)	2.24	(a)	2.25	(a)	2.26	(b)	2.27	(b)		
2.28	(d)	2.29	(c)	2.30	(c)	2.31	(3)	2.32	(a)	2.33	(a)	2.34	(c)	2.35	(7)	2.36	(d)		
2.37	(c)	2.38	(80)	2.39	(a)	2.40	(c)	2.41	(a, c, d)	2.42	(b, c, d)	2.43	(b, c)	2.44	(c)				
2.45	(c)	2.46	(14)																

Explanations Process Management-II

2.1 (c)
Critical section is an area in a program where the resources can be accessed.

2.2 (c)
Ensure that one particular philosophers picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork.

2.3 (d)
Everytime when process P_{10} enters it performs the 'UP' operation so after that one more process can enter into it, so this phenomena continue till end. So 10 processes at a time can enter into critical section.

2.4 (b)
Race condition.

2.5 (b)
Initially semaphore is 10, then 6 down operations are performed means (10 – 6) and 4 up operation means (4 + 4) = 8.
So option (b) is correct.

2.6 (b)
All processes are in deadlock condition i.e., not a single one process can enter into critical section.

2.7 (a)
If stack pointer register is not available then activation records in the stack can not be created. So it cannot have subroutine call instruction.

2.8 (b)
Disk has swap space.

2.9 (b)
To ensure the mutual exclusion, we have to take care that 'turn' value which is 'j' so we should not allow that process in CS which is having flag [j] = true.
Hence option (b) is correct.

2.10 (b)
To get the expected output , The sequence of processes are P, Q, P, Q, P, Q and so on we must initialize S = 1 and T = 0.

2.11 (c)

If the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd then the procedures becomes.

S = 1

Process P:	Process Q:
While(1){	While(1){
W : P(S)	Y : P(S)
print '0';	print '1';
print '0';	print '1';
X : V(S)	Z : V(S)
}	}

2.12 (d)

S_x and S_y are two binary semaphores if we assume P means wait and V means signal then for two process P_1 and P_2 we take alternate then there in no chance of dead lock.

2.13 (d)

Here full and empty are integer semaphore which denote number of full slots in buffer and number of empty slot in buffer at any time respectively. While mutex is binary semaphore to provide mutual exclusion.

Initially full = 0 empty = n and mutex = 1.

- When no item into buffer, since mutex = 1. So to restrict consumer to remove item P (full) must be there at M and when it leave the critical section made V (empty) at N i.e. one empty slot increase.
- When producer tries to add an item into buffer before entering to critical section, it must perform P (empty) i.e. one empty decrease and while leaving critical section make V (full) i.e. one new item added successfully.

So, correct order will be P (empty), V (full), P (full) and V (empty) to ensure consumer will be block until 1 item added into buffer.

2.14 (d)

None of the processes will ever starve. Starvation is said to be possible if in ALL the possible ways that a process want to enter CS, it is DENIED the access. But that can't happens here.

P1 is never starved because this is INFACT the first process that can enter. V(X) by P1 is always successful and hence it always enter the CS. So 1 is FALSE.

When P1 is in CS, X = 1. At this time P2 can make a successful DOWN operation P(X) and enter CS. So 2 is also FALSE.

But P2 must exit the CS before P1.

2.15 (b)

With one semaphore it is not possible, since binary semaphore is used.

So, try with 2 binary semaphore a and b assume initialize to 0.

P1:	P2:
P(a) R1	
	R1 V(a) P(b) R2
R2 V(b) P(a) R3	
	R3 V(a) P(b) R4
R4 V(b)	

So, minimum 2 semaphore needed.

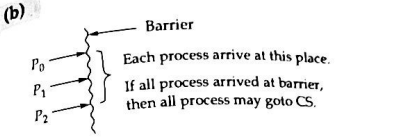
2.16 (a)

The P and V function is as follows:

```
1.  void P (binary-semaphore * S){
2.      unsigned y;
3.      unsigned * x = & (S → value);
4.      do {
5.          fetch-and-set x, y;
6.      } while (y);
7.  }
8.  void V (binary-semaphore * S){
9.      S → value = 0;
10. }
```

fetch-and-set instruction always sets the memory location x = 1 and fetches the old value of x into y. The binary semphore * S takes only two values either 0 and 1. When we initiliaze S = 0 then in statement 3 this value will be start at location x and fetch-and-set instruction changes the value of x = 0 to x = 1 and y becomes 0. If there are more than two processes and context switching between processes is disabled in P then this implementation doesn't work properly and can't synchronize the processes.

2.17 (b)



Given code:

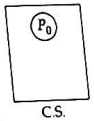
```
1.  P(S);
2.  process_arrived++;
3.  V(S);
4.  while (process_arrived != 3);
5.  P(S);
6.  process_left++;
7.  if(process_left == 3)
8.  {  process_arrived = 0;
9.     process_left = 0;
10. }
11. V(S);
```

This time act as barrier

These code work like critical section

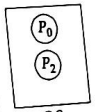
Each process in the set arrive at barrier. We need to provide synchronization to process to enter barrier and wait for all other processes to arrive. Once they arrive, they start leaving the barrier and same thing goes on.

Process	Code
P_0	1. S = 0 2. P_A = 1 3. S = 1 4. (1! = 3); → condition true



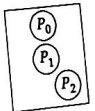
So, P_0 arrive at barrier and waiting for other process to arrive.

Process	Code
P_1	1. S = 0 2. P_A = 2 3. S = 1 4. (2! = 3); True



So, P_1 stop and waiting for other process.

Process	Code
P_2	1. S = 0 2. P_A = 3 3. S = 1 4. (3! = 3); False



Now all 3-process arrived barrier, so while condition become false, to now the processes start leaving the barrier one by one.

Now suppose P_0 leaves the barrier, and not waiting for other processes (P_1 and P_2) to leave the barrier and immediately P_1 entering to barrier.

i.e.

Process	Code
P_0	5. S = 0 6. P_L = 1 7. (1 == 3) : false (so, 8 and 9 do not execute) 11. S = 1

Now S = 1

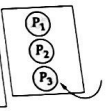
P_1 and P_2 still in loop.

So, we have two choices:

Choice 1: P_1 can leave the barrier

Choice 2: P_0 can enter into the barrier (Case for deadlock)

Process	Code
1.	S = 0
2.	P_A = 4
3.	S = 1
4.	(4! = 3); True



So, P_0 also enters the barrier and the while loop now, will be true forever because, we have 3 processes and all three processes stuck up at barrier.

Hence, no possibility of while loop to become false. Hence, deadlock will occur.

Hence we conclude that, as soon as all the 3 processes arrive at barrier and until and unless, all the process leave the barrier, we will not call barrier again. So, two invocations if used in immediate succession, may lead to deadlock.

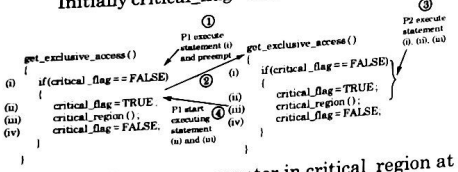
2.18 (b)

```
void barrier (void)
{
    while (process_left != 0);
    P(S);
    P_A++;
    V(S);
    while
    (P_A! = 3);
    P(S);
    P_L++;
    if (P_L == 3)
    {
        P_A = 0;
        P_L = 0;
    }
        V(S);
}
```

Correction in implementation of code

2.19 (c)

Initially critical_flag = false



So, both process can enter in critical_region at same time.

So, no mutual exclusion.

We know that Dead lock → Mutual exclusion.

So, no Mutual exclusion → No deadlock. Here no deadlock possible.

Note: Here arrow shows order of execution and preemption is temporary, any process can preempt at any time.

2.20 (d)

In process P_1 if wants1 is true then wants2 entered in critical section and in process P_2 if wants2 is true then wants1 entered in critical section but in both cases a deadlock can occur but it ensures the mutual exclusion.

2.21 (c)

Writers code

wait (wrt)
writing is performed
signal (wrt)

- If read count = = 1 then it must perform wait (wrt) So, at same time reader and writer cannot access critical section.
- Since more than one reader an access the critical section simultaneously So, at S2 signal (mutex) must be done.

Readers code

wait (mutex)
read count = read count + 1
if (read count = =1)
then wait (wrt) S1
signal (mutex) S2
reading is performed
wait (mutex) S3
read count = read count = 1
If (read count = = 0)
then signal (wrt) S4
Signal (mutex)

Since by seeing S1 and S2 option (c) matches. Hence it is correct answer.

2.22 (c)

In given code, P(S) decrements the value of semaphore and V(S) increment the value of semaphore. P_b and V_b are wait and signal operations on binary semaphore. X_b and Y_b are binary semaphore. To avoid the mutual exclusion condition if the value of X_b is 1 and the value of Y_b = 0 then P(S) and V(S) works properly.

2.23 (a)

II. False let $P_1, P_2, ... P_n$ are processes let P_1 is currently holding lock and P_2 is waiting to enter CS after P_1 it is not definite that P_2 will surely enter critical section and it may happen that each time P_2 try to enter CS it is blocked by some other process which may lead P_2 to starvation.

III. False as there is not fixed order whichever process get lock access can enter so not necessary to follow FIFO.

IV. Given code of test and set instruction satisfy mutual exclusion so atmost one process can enter CS

∴ By elimination only option (a) is true.

2.24 (a)

P1	P2
While (S1 == S2);	While (S1! = S2);
Critical section	Critical section
S1 = S2;	S2 = Not(S1);

We initially take $S_1 = 1$, and $S_2 = 1$.

When P1 execute S1 = S2 then it do nothing and process P1 go into critical section after executing it make the value of S1 = S2 and when P1 execute then P2 not execute because that checks different condition in while so both show mutual exclusion. But progress is not shown by these two because in this one process can be delayed infinitely.

Because again both P1 make its own condition true (S1 = S2 and S2 = not (S1);).

2.25 (a)

When $S_0 = 1$, $S_1 = 0$

So firstly the value of S_1 and $S_2 = 0$, P_1 and P_2 Not execute. The value of $S_0 = 1$, So it execute, process P_0 Now value of $S_0 = 0$

And it print zero one time when it release (S_1) call then value of S_1, S_2 increase by 1. So P_1 execute. It makes value of S_1 again zero and $S_0 = 1$. after it release (S_2) make value of $S_2 = 0$ by applying wait and $S_0 = 2$. So again P_0 process execute and wait (S_0) means it decrease by one so it print zero and this process continue. So P_0 print zero at least twice.

2.26 (b)

Assume P_1 executes until while condition and preempts before executing L = 1. Now P_2 executes all statements, hence L = 0. Then P_1 without checking L it makes L = 1 by executing the statement where it was preempted.

∴ It takes a non-zero value (L = 1) when the lock is actually available (L = 0)

2.27 (b)

Trick: To check deadlock in the given code proceed parallely i.e. first executed the first line of all the processes then executed the second line of all processes and so on.

(a)

X	Y	Z
1. P(a)	1. P(b)	1. P(c)
2. P(b)	2. P(c)	2. P(d)
3. P(c)	3. P(d)	3. P(a)

Value of a, b, c are 1 initially.

Execute first line of all processes i.e. $X_1 \rightarrow Y_1 \rightarrow Z_1$ after the executive of $X_1 \rightarrow Y_1 \rightarrow Z_1$ value of {a,b,c} = 0. Now no process is allowed to enter into the critical section. So deadlock.

(b)

X	Y	Z
1.P(b)	1.P(b)	1.P(a)
2.P(a)	2.P(c)	2.P(c)
3.P(c)	3.P(d)	3.P(d)

1. Execute X_1 so value of b gets 0.
2. As value of b = 0 when we execute Y_1 it get blocked (Y is blocked).
3. Now execute Z_1 so value of a get 0.
4. Now try to execute the second line of unblocked processes as value of a = 0 when we try to execute X_2 it get blocked (X is blocked)
5. Now execute Z_2, value of c gets 0.
6. As process X and Y are blocked so we can easily execute Z_3.

So no deadlock, like wise we can check option c and d, so option (b) is correct.

2.28 (d)

	X	W	Y	Z
1.	P(s)	P(s)	P(s)	P(s)
2.	R(x)	R(x)	R(x)	R(x)
3.	x = x +1	x = x + 1	x = x + 2	x = x + 2
4.	Store to MM	Store to MM	Store to MM	Store to MM
5.	V(s)	V(s)	V(s)	V(s)

1. Start with X. P(s) ⇒ s = 1 read x = 0, x = x+1 Now before store into the memory process X preempt to process Y.
2. Y will perform P(s) ⇒ s = 0, read x = 0 x = x – 2 then store x= – 2 and V(s) ⇒ s = 1
3. Z will perform P(s) ⇒ s = 0, read x= – 2 x = x – 2 then store x = – 4 and V(s) ⇒ s = 1
4. Now process x will store value of variable x into the MM so x = 1 and V(s) ⇒ s = 2
5. Now process W perform P(s) ⇒ s = 1 read x = 1 x = x + 1, store x = 2, V(s) ⇒ s = 2, Option (d) is correct

2.29 (c)

The solution is using a binary semaphore. X and Y are two different processes whatever the values inserted by the processes in the array that will be used by process Y afterwards.

Option (a): It is wrong, because there exists a deadlock.

Option (b): It is wrong, because process X can insert multiple values in an array and terminate but process Y can consume only one value.

Option (c): It is correct, because each and every value inserted by the process X in the array will be immediately consumed by process Y.

Option (d): It is wrong, because process X can insert two values in the array but the process Y can consume only one value if the preemption occurs after V(S) operation in the entry seciton of process Y.

Process Y :	------- ------- Entry Y(R, S) { V(S);	R = 0; S = ~~0~~ 1;
	⟶ Preemption	
Process X :	------- ------- Exists (R, S) { V(R); P(S);	R = ~~0~~ ~~1~~ 1; S = ~~1~~ 0;
	------- Exists (R, S) { V(R);	
	⟶ Preemption	
Process Y :	------- P(R); ------- ------- Entry Y(R, S) { V(S);	R = ~~1~~ 0; S = ~~0~~ 1;
	⟶ Preemption	
Process X :	P(S);	R = 0; S = ~~1~~ 0;

2.30 (c)

Option (a): If producer want to produce an item, it will first perform wait(s); then add to buffer( ); then signal semaphore s, n

∴ Value of semaphore become

n = ~~0~~ 1;

s = ~~1~~ ~~0~~ 1;

If we now run consumer code, we have to perform wait option on semaphore s, n

∴ n = ~~0~~ ~~1~~ 0;

s = ~~1~~ ~~0~~ ~~1~~ 0;

after removing from buffer( ); it will signal(s) and consume( )

∴ Consumer can consume item produced by producer so option (a) is false.

Option (b): It is also false as after removing one item from buffer we can repeat above process any number of times.

Option (c): If consumer run before producer has produced i.e. if buffer is empty, it will perform wait option on 's' which is successful after that it perform an unsuccessful wait option on semaphore 'n'

So, s = ~~1~~ 0;

n = 0 (wait will lead to suspension)

Now if producer try to run and perform wait(s); it will also get suspended as both process are suspended and no other process to signal them so deadlock occurs.

∴ Option (c) is true.

Option (d): False, if we consume item in the buffer using consumer code, followed by producer in option (c), deadlock will occur.

2.31 (3)

B = 2

C = B − 1 B = 2 × C D = 2 × B B = D − 1 ⇓ B = 3	D = 2 × B B = D − 1 C = B − 1 B = 2 × C ⇓ B = 4	C = B − 1 D = 2 × B B = 2 × C B = D − 1 ⇓ B = 3	C = B − 1 D = 2 × B B = D − 1 B = 2 × C ⇓ B = 2	D = 2 × B C = B − 1 B = D − 1 B = 2 × C ⇓ B = 2	D = 2 × B C = B − 1 B = 2 × C B = D − 1 ⇓ B = 2

⇒ B = 2, 3 or 4

∴ Three distinct values are possible.

2.32 (a)

(i) Mutual exclusion fail in the given program. First, process X executes all statements till var$P$ = true. Second, process Y executes all statements till var$Q$ = true; Now, both process X and process Y both can execute while condition as true and then enters critical section simultaneously. Therefore mutual exclusion fails.

(ii) Deadlock is prevented by the given code.

2.33 (a)

Since $p$max returns as integer not smaller than any of its argument there may be a case when $t[\ j]$ == $t[i]$ and $t[\ j]$ is not equal 0 for it.

∴ while ($t[\ j]$!= 0 && $t[\ j]$ <= $t[i]$);

will be true and process may get stuck in infinite loop which lead to deadlock and hence progress is also not satisfied.

As there is not proper definite waiting for process

∴ Among process P_0 ..... P_{n-1} some process may not get chance to execute and may starve.

∴ Bounded wait is not satisfied.

2.34 (c)

Given problem is strict alteration problem it violates progress. As if initially when turn is initialized to zero and process 1 try to enter into critical section it will be stuck in while loop as a result condition for progress is violates because a process outside the critical section should not block another process from entering to critical section if it is free. Bounded wait is satisfied as after process 0 has executed process 1 will definitely get a chance to execute so there is bounded wait with respect to number of processes.

2.35 (7)

S − 20 + 12 = −1

S − 8 = −1

S = − 1 + 8 = + 7

So, the initial values of the semaphore should be '7'.

2.37 (c)

Full = N, empty = 0, mutex = 1

- Initially buffer will be empty, so consumer should not start first, so option b, D are eliminated.
- With option A consumer will never consume the item, so it is wrong.
- Option 'c' is correct answer which proper functionality of produce and consumer.

2.38 (80)

Initial value of D = 100

Minimum value occur when P1, P2, P3 read and P1, P3 update the value

P1 update D = 110

P3 update D = 120

and last P2 will update the D = 100 − 50 = 50 = X

Maximum value will occur when P2 and P3 read the initial value of D and update, first run P2 and update D = 50. P3 will run and update

D = 100 + 10 = 110

Now P1 will read the value of D = 110 and update

D = D + 20 = 110 + 20

D = 130 = Y

Y − X = 130 − 50 = 80

2.39 (a)

- Process 'p' is holding X_p resources and does not request any new resources. So 'p' can complete it's execution and releases X_p resources.
- Process 'q' is holding X_q resources and does not request any new resources. So 'q' can complete it's execution and release X_q resources.

  Now available resources = $X_p + X_q$
- Other than 'p' and 'q' no other processes completes it execution, means available resources ($X_p + X_q$) is less than the resources demanded or requested by all other processes.

  So, $X_p + X_q < \text{Min}\ \{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$ holds true.

2.40 (c)

All the processes running the given code will remain block due to wait(b) until the value of count becomes n.

When the value of count will become equals to n, value of b changes to 1 which will subsequently allow a process to go into section Q.

Hence, no process can go to section Q until all the process executes code section P.

2.41 (a, c, d)

(a) True: (P_1) first process came executed wait(S); wait(S)

counter++; was partially executed and the process preempted. Other process came updated counter to 4.

Now P_1 came and wrote counter to 1. So (a) is correct.

(b) Not possible at least counter will be 1.

(c) True: All 5 processes executed first wait(S) and are blocked.

(d) True: Each process executed the code sequentially and counter was finally updated to 5.

Option (a), (c) and (d) are correct.

2.42 (b, c, d)

- Mutual exclusion is not violated.
- Also, there will be no deadlock because of forceful preemption of resources.
- This may lead to starvation if the process is keeps on coming and preempting each other like P_1 is preempted by P_2 and P_2 is preempted by P_3.
- Live-lock is also possible due to continuous preemption of resources.

For option (c) consider two processes P_1 and P_2 now P_1 enter the code acquires lock and resource. Now P_2 enters the else part kills P_1 and acquire R and restart P_1.

Now P_1 again acquire lock and kills the process P_2 this continues creating a live lock scenario but there is ambiguity in the code since "Release R" is not written any where so ambiguity is regarding how the process will release Resource R. According to the code, the only way to release the resource is by getting killed.

2.43 (b, c)

P_1 and P_2 can spawn two threads T_1 and T_2.

```
int x = 0; //global
Lock L1; //global
main( ) {
    foo( ); //Thread T1
    foo( ); //Thread T2
    print(x);
}
foo( ) {
a.  int y = 0;
b.  Acquire L1;
c.  x = x + 1;
d.  y = y + 1;
    Release L1;
    print(y);
}
```

- Let P_1 executed T_1 and in foo( ).

(a) $P_1 - T_1$ $\quad$ X $\boxed{\cancel{0}}$ 1

(b) $P_1 - T_1$

(c) $x = x + 1$ $(P_1 - T_1)$

.... Preempt T_1 of P_1.

Similarly, perform thread T_2 of P_1 then $x = 2$.

Now, if we similarly perform both threads of P_2 then x will be maximum 2.

Note: It is not mentioned about IPC or concurrent mechanism so the process will not shared the global variable. Each process will have its on own address space so the maximum value of x by each process can be 2 only.

Note: But as we know every foo call will have its own copy of variable y so y can not be more than 1 in any case.

2.44 (c)

Inorder to get the required output only semaphore S_1 should be initialized to 1, other semaphore should be initialized to 0.

So, $S_1 = 1$; $S_2 = 0$; $S_3 = 0$

Answer is option (c).

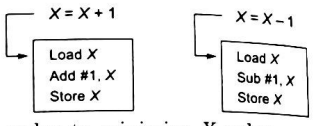
2.45 (c)

I-1: s is a binary semaphore initialized to 1. With binary semaphore, there can't be two threads that can simultaneously access the critical section, Thus, each thread with run incr( ) or decr( ) one by one.

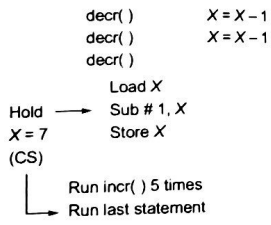
Hence, only possible value of X at the end is 12.

I-2: With counting semaphore of value 2. Two processes at a time can access critical section.

Note:



In order to minimize X value, we need to maximize the impact of decr( ) and minimize the impact of incr( ).



Thus, finally $\boxed{X = 7}$ will get stored.

3 Deadlock

3.1 An operating system contains 3 user processes each requiring 2 units of resource R. the minimum number of units of r such that no deadlocks will ever arise is

(a) 3 (b) 5

(c) 4 (d) 6 **[1997 : 2 M]**

3.2 A computer has six tape drives, with n processes competing for them. Each process may need two drives. What is the maximum value of n for the system to be deadlock free?

(a) 6 (b) 5

(c) 4 (d) 3 **[1998 : 1 M]**

3.3 Which of the following is NOT a valid deadlock prevention scheme?

(a) Release all resources before requesting a new resource.

(b) Number the resources uniquely and never request a lower numbered resource than the last one requested.

(c) Never request a resource after releasing any resource

(d) Request and all required resources be allocated before execution. **[2000 : 2 M]**

3.4 In a certain operating system, deadlock prevention is attemped using the following scheme. Each process is assigned a unique timestamp, and is restarted with the same timestamp if killed. Let P_h be the process holding a resource R, P_r be a process requesting for the same resource R, and $T(P_h)$ and $T(P_r)$ be their timestamps respectively. The decision to wait or preempt one of the processes is based on the following algorithm.

if $T(P_r) < T(P_h)$ then
kill P_r
else
wait

Which one of the following is TRUE?

(a) The scheme is deadlock-free, but not starvation-free

(b) The scheme is not deadlock-free, but starvation-free

(c) The scheme is neither deadlock-free nor starvation-free

(d) The scheme is both deadlock-free and starvation-free **[2004 : 2 M]**

3.5 Suppose n processes, $P_1, ..., P_n$ share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process P_i is s_i where $s_i > 0$. Which one of the following is a sufficient condition for ensuring that deadlock does not occur?

(a) $\forall i, s_i < m$ (b) $\forall i, s_i < n$

(c) $\sum_{i=1}^{n} s_i < (m + n)$ (d) $\sum_{i=1}^{n} s_i < (m * n)$

[2005 : 2 M]

3.6 Consider the solution to the bounded buffer producer/consumer problem by using general semaphores S, F, and E. The semaphore S is the mutual exclusion semaphore initialized to 1. The semaphore F corresponds to the number of free slots in the buffer and is initialized to N. The semaphore E corresponds to the number of elements in the buffer and is initialized to 0.

Producer Process	Consumer Process
Produce an item;	Wait (E);
Wait (F);	Wait (S);
Wait (S);	Remove an item from the buffer;
Append the item to the buffer;	Signal (S);
Signal (S);	Signal (F);
Signal (E);	Consumer the item;

Which of the following interchange operations may result in a deadlock?

I. Interchanging Wait (F) and Wait (S) in the Producer process.

II. Interchanging Signal (S) and Signal (F) in the Consumer process.

(a) I only (b) II only

(c) Neither I nor II (d) Both I and II **[2006 : 2 M]**

3.7 Consider the following snapshot of a system running n processes. Process i is holding x_i instances of a resource R, for $1 \le i \le n$. Currently, all instances of R are occupied. Further, for all i,

process i has placed a request for an additional y_i instances while holding the x_i instances it already has. There are exactly two processes p and q such that $y_p = y_q = 0$. Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock?

(a) $\min(x_p, x_q) < \max_{k \neq p, q} y_k$
(b) $x_p + x_q \geq \max_{k \neq p, q} y_k$
(c) $\min(x_p, x_q) < 1$
(d) $\min(x_p, x_q) > 1$ **[2006 : 2 M]**

3.8 A single processor system has three resource types X, Y, and Z, which are shared by three processes. There are 5 units of each resource type. Consider the following scenario, where the column alloc denotes the number of units of each resource type allocated to each process, and the column request denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST?

	alloc	request
	X Y Z	X Y Z
P0	1 2 1	1 0 3
P1	2 0 1	0 1 2
P2	2 2 1	1 2 0

(a) P0 (b) P1
(c) P2
(d) None of the above, since the system is in a deadlock **[2007 : 2 M]**

3.9 The following is a code with two threads, producer and consumer, that can run in parallel. Further, S and Q are binary semaphores equipped with the standard P and V operations.

```
semaphore S = 1, Q = 0;
integer x ;
producer :              consumer :
while (true) do         While (true) do
P(S) ;                  P(Q) ;
x = produce( ) ;        consume (x);
V(Q) ;                  V(S) ;
done                    done
```

Which of the following is TRUE about the program above?

(a) The process can deadlock
(b) One of the threads can starve
(c) Some of the items produced by the producer may be lost
(d) Values generated and stored in 'x' by the producer will always be consumed before the producer can generate a new value **[2008 : 2 M]**

3.10 An operating system implements a policy that requires a process to release all resources before making a request for another resource. Select the TRUE statement from the following:

(a) Both starvation and deadlock can occur
(b) Starvation can occur but deadlock cannot occur
(c) Starvation cannot occur but deadlock can occur
(d) Neither starvation nor deadlock can occur **[2008 : 2 M]**

3.11 Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes?

(a) In deadlock prevention, the request for resources is always granted if the resulting state is safe
(b) In deadlock avoidance, the request for resources is always granted if the resulting state is safe
(c) Deadlock avoidance is less restrictive than deadlock prevention
(d) Deadlock avoidance requires knowledge of resource requirements a priori **[2008 : 2 M]**

3.12 Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-preemptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the resources as follows if executed independently.

Process P1 :	Process P2 :	Process P3 :
t = 0 : requests 2 units of R2	t = 0 : requests 2 units of R3	t = 0 : requests 1 unit of R4
t = 1 : requests 1 unit of R3	t = 2 : requests 1 unit of R4	t = 2 : requests 2 units of R1
t = 3 : requests 2 units of R1	t = 4 : requests 1 unit of R1	t = 5 : releases 2 units of R1
t = 5 : releases 1 unit of R2 and 1 unit of R1	t = 6 : releases 1 unit of R3	t = 7 : requests 1 unit of R2
t = 7 : releases 1 unit of R3	t = 8 : Finishes	t = 8 : requests 1 unit of R3
t = 8 : requests 2 units of R4		t = 9 : Finishes
t = 10 : Finishes		

Which one of the following statements is TRUE if all three processes run concurrently starting at time t = 0?

(a) All processes will finish without any deadlock
(b) Only P1 and P2 will be in deadlock
(c) Only P1 and P3 will be in deadlock
(d) All three processes will be in deadlock **[2009 : 2 M]**

3.13 A system has n resources $R_0, R_1, \ldots, R_{n-1}$, and k processes $P_0, P_1, \ldots, P_{k-1}$. The implementation of the resource request logic of each process P_i, is as follows:

```
if (i%2 = = 0) {
    if (i < n) request R_i ;
    if (i + 2 < n) request R_{i+2};

}
else {
    if (i < n) request R_{n-i};
    if (i + 2 < n) request R_{n-i-2};

}
```

In which one of the following situations is a deadlock possible?

(a) n = 40, k = 26 (b) n = 21, k = 12
(c) n = 20, k = 10 (d) n = 41, k = 19 **[2010 : 2 M]**

3.14 An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P0, P1, and P2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a **safe** state. Consider the following independent requests for additional resources in the current state:

REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z

REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z

Which one of the following is **TRUE**?

(a) Only REQ1 can be permitted.
(b) Only REQ2 can be permitted.
(c) Both REQ1 and REQ2 can be permitted.
(d) Neither REQ1 nor REQ2 can be permitted. **[2014 (Set-1) : 2 M]**

3.15 A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is ____. **[2014 (Set-3) : 2 M]**

3.16 A system has 6 identical resources and N processes competing for them. Each process can request atmost 2 resources. Which one of the following values of N could lead to a deadlock?

(a) 1 (b) 2
(c) 3 (d) 4 **[2015 (Set-2) : 1 M]**

3.17 Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

I. Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.
II. The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers.
III. The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers.
IV. The resources are numbered uniquely. A process is allowed to request only for a resource with resource number larger than its currently held resources.

Which of the above policies can be used for preventing deadlock?

(a) Any one of I and III but not II or IV
(b) Any one of I, III and IV but not II
(c) Any one of II and III but not I or IV
(d) Any one of I, II, III and IV **[2015 (Set-3) : 2 M]**

3.18 A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for three processes are shown below:

Process	Current Allocation	Maximum Requirement
P1	3	7
P2	1	6
P3	3	5

Which of the following best describes current state of the system?

(a) Safe, Deadlocked
(b) Safe, Not Deadlocked
(c) Not Safe, Deadlocked
(d) Not Safe, Not Deadlocked **[2017 (Set-2) : 2 M]**

3.19 Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a maximum of K instances. Resource instances can be requested and released only one at a time. The largest value of K that will always avoid deadlock is _____. **[2018 : 1 M]**

3.20 In a system, there are three types of resources: E, F and G. Four processes P_0, P_1, P_2 and P_3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example Max[P_2, F] is the maximum number of instances of F that P_2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

Consider a state of the system with the Allocation matrix as shown below and in which 3 instances of E and 3 instances of F are the only resources available.

Allocation	E	F	G
P_0	1	0	1
P_1	1	1	2
P_2	1	0	3
P_3	2	0	0

Max	E	F	G
P_0	4	3	1
P_1	2	1	4
P_2	1	3	3
P_3	5	4	1

From the perspective of deadlock avoidance, which one of the following is true?

(a) The system is in safe state.
(b) The system is not in safe state, but would be safe if one more instance of E were available.
(c) The system is not in safe state, but would be safe if one more instance of F were available.
(d) The system is not in safe state, but would be safe if one more instance of G were available.

[2018 : 2 M]

3.21 Which of the following statements is/are TRUE with respect to deadlocks?

(a) Circular wait is a necessary condition for the formation of deadlock.
(b) In a system where each resource has more than one instance, a cycle in its wait-for graph indicates the presence of a deadlock.
(c) If the current allocation of resources to processes leads the system to unsafe state, then deadlock will necessarily occur.
(d) In the resource-allocation graph of a system, if every edge is an assignment edge, then the system is not in deadlock state.

[2022 : 1 M]

Answers Deadlock

3.1	(c)	3.2	(b)	3.3	(c)	3.4	(a)	3.5	(c)	3.6	(a)	3.7	(b)	3.8	(c)	3.9	(d)		
3.10	(b)	3.11	(a)	3.12	(a)	3.13	(b)	3.14	(b)	3.15	(7)	3.16	(*)	3.17	(d)	3.18	(b)		
3.19	(2)	3.20	(a)	3.21	(a, d)														

Explanations Deadlock

3.1 (c)

There are 3 processes, and demands '2' resources. If any one gets equal resources as it's demand, deadlock won't occur, and according to this question 4 resources are enough.

3.2 (b)

Every process needs 2 resources. With 5 resources atleast one can get whole 2 and executes. So after execution it frees 2 resources which can be used by other processes.

P_1	P_2	P_3	P_4	P_5
1	1	1	1	1

3.3 (c)

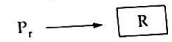
It is not a valid deadlock prevention scheme.

3.4 (a)

Given a deadlock prevention scheme, where each process is assigned a unique time stamp and if killed, it get restarted with same time stamp.

P_h : Process holding resource R.

P_r : Process requesting a resource R.



Now, consider the following scenario, where P_1, P_2, ... P_n are n-processes with given time stamp.

Process	P_1	P_2	P_3	P_4	P_5	.....	P_n
Time stamp	10	20	30	40	50	.....	.....

Case-I: Let P_3 is holding resource R ($\therefore P_h = P_3$) and P_5 is requesting for resource R (so, $P_r = P_5$)

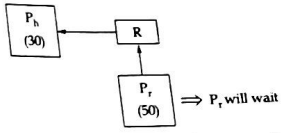
$\therefore$ Condition

if $T(P_r) < T(P_h)$

i.e. $T(P_5) < T(P_3)$

$50 < 30$ // False

Hence P_5 will wait for P_3.



Case-II: Now consider P_3 is requesting for resource R currently hold by process P_5, so checking condition.

if $T(P_r) < T(P_h)$

i.e. $T(P_3) < T(P_5)$

$30 < 50$ // True

$\therefore$ Kill P_r

So, P_3 will be killed and P_5 get resource R now suppose after sometime.

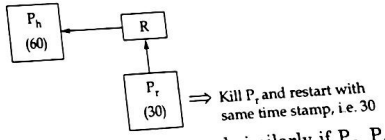
Process P_3 again requesting for resource R let now R is hold by P_6

Again checking

if $T(P_r) < T(P_h)$

$30 < 60$ // True

$\Rightarrow$ P_3 will get killed



P_3 will be killed again and similarly if P_7, P_8 ..... so on keep on arriving and holding resource R then innocent process P_3 may go to indefinite waiting which lead to starvation.

So, option (a) is answer.

3.5 (c)

Total number of resources shared by process P_1, ... P_n = m

Maximum resource requirement of process $P_i = S_i$

i.e.

Process	Requirement
P_1	S_1
P_2	S_2
P_n	S_n

So the worst case allocation of resources for deadlock prevention. So, that deadlock do not occur is:

[Allot the resources to the processes, one less than the actual requirement (worst case deadlock will occur) + 1 (for no deadlock)]

So, we have

$(S_1 - 1) + (S_2 - 1) \ldots (S_n - 1) + 1 \le m$

$$\sum_{i=1}^{n} (S_i - 1) + 1 \le m$$

$$\sum_{i=1}^{n} S_i - \sum_{i=1}^{n} 1 + 1 \le m$$

$$\sum_{i=1}^{n} S_i - n + 1 \le m$$

$$\sum_{i=1}^{n} S_i \le m + n - 1$$

$$\sum_{i=1}^{n} S_i < (m + n)$$

3.6 (a)

Suppose the slots are full → F = 0.

Now, if Wait(F) and Wait(S) are interchanges and Wait(S) succeeds, The producer will wait for Wait(F) which is never going to succeed as Consumer would be waiting for Wait(S).

So, deadlock can happen.

If Signal(S) and Signal(F) are interchanged in Consumer, deadlock won't happen.

It will just give priority to a producer compared to the next consumer waiting.

3.7 (b)

Total number of processes = n

Process i is holding x_i instances of a resource R for $1 \le i \le n$.

Process i holding resource x_i and additional required y_i. There are two processes p and q such that $y_p = y_q = 0$.

For no dead lock can occur

$$x_p + x_q \ge (\max_{k \ne p,q} y_k)$$

3.8 (c)

Consider the matrix

	alloc XYZ	request XYZ
P_0	121	103
P_1	201	012
P_2	221	120

So the available matrix is (0, 1, 2) and the requirement of process P_1 is (0, 1, 2) so P_1 can use the available resources after releasing the resources the matrix becomes:

	alloc XYZ	request XYZ
P_0	121	103
P_1	000	000
P_2	221	120

In this time available matrix is (2 1 3) and required matrix of P_0 is (1 0 3) so P_0 uses the resources of the releasing the resource. The available matrix is (3, 3, 4) so P_2 will use the resources, P_2 will finish last.

3.9 (d)

Consumer can consume only once the producer has produced the item, and producer can produce (except the first time) only once the consumer has consumed the item.

3.10 (b)

Releasing all resources before making a request for another resource is a technique of deadlock prevention. So, deadlock cannot occur.

However, we cannot say anything about starvation, it may occur or may not.

3.11 (a)

In deadlock prevention, the request for a resource may not be granted even if the resulting state is safe.

3.12 (a)

t	R1(3)	R2(2)	R3(3)	R1(2)
0	3	0	1	1
1	3	0	0	1
2	1	0	0	0
3	1	0	0	0
4	0	0	0	0
5	1	1	0	0
6	1	1	1	0
7	1	0	2	0
8	2	0	2	1
9	2	1	3	0
10				

3.13 (b)

Logic in question:

If both odd and even type of process need same resource, then deadlock may occur.

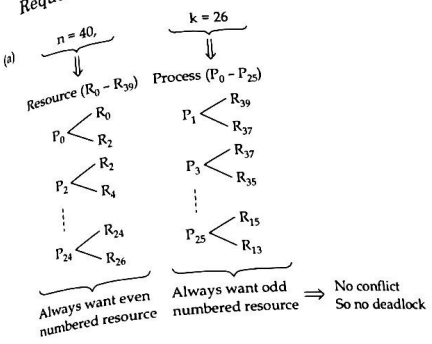
For even processes:

Request : R_i and R_{i+2}

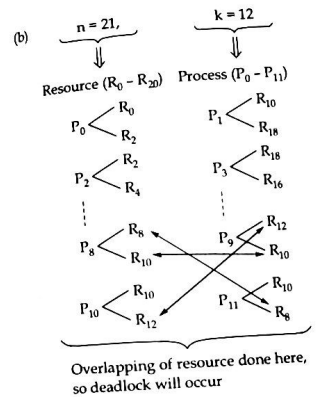
For odd processes:

Request : R_{n-1} and R_{n-i-2}

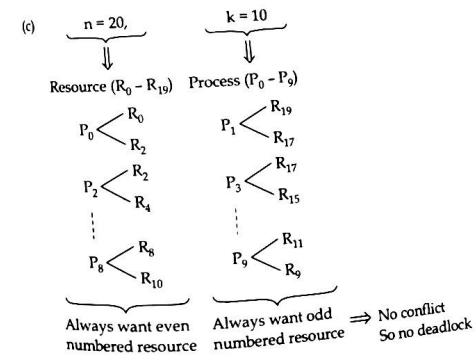
(a)



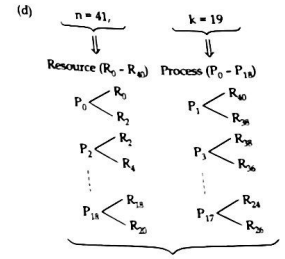
(b)



(c)



(d)



3.14 (b)

	Allocation	Max	Need	Need with additional requests
	X Y Z	X Y Z	X Y Z	X Y Z
P0	0 0 1	8 4 3	8 4 2	8 4 4 (REQ1)
P1	3 2 0	6 2 0	3 0 0	5 0 0 (REQ2)
P2	2 1 1	3 3 3	1 2 2	1 2 2

Available: X Y Z

3 2 2 → P2 can satisfy it needs

2 1 1 → P2 releases its allocation

5 3 3 → P1 satisfies it needs

3 2 0 → P1 releases its allocation

8 5 3 → P0 can not satisfy with additional requests

∴ Only REQ2 can be permitted.

3.15 (7)

Number of programs = 3

(P_1, P_2, P_3)

Tape required per program = 3

Allocating tape units, so that deadlock does not occur:

$$\underbrace{\overset{P_1}{(3-1)} + \overset{P_2}{(3-1)} + \overset{P_3}{(3-1)} + 1}_{\Downarrow \; 7}$$

Answer is 7.

3.16 (*)

N is number of processes, 6 identical resources

$N \ge 6 \times (2 - 1)$

If $N \ge 6 \Rightarrow$ It could lead to a deadlock

No option matches with $N \ge 6$

3.17 (d)

I. Dissatisfying hold and wait for deadlock prevention.
II. Dissatisfying circular wait for deadlock prevention.
III. Dissatisfying circular wait for deadlock prevention.
IV. Dissatisfying circular wait for deadlock prevention.

3.18 (b)

P.id	Current Allocation	Maximum Requirement	Remaining need	Current available
P1	3	7	4	9 – 7 = 2
P2	1	6	5	5
P3	3	5	2	8
				9

Safe sequence ⇒ P3 → P1 → P2
Safe and Not deadlocked

3.19 (2)

P_1 P_2 P_3
| | |
|

No deadlock

Maximum each process can request for 2 resources so, that there will not be any deadlock, because we, have only 4 resource available.
So, K value is '2'.

3.20 (a)

	Max need			Current allocation			Current available			Remaining need		
	E	F	G	E	F	G	E(3)	F(3)	G(0)	E	F	G
P_0	4	3	1	1	0	1	4	3	1	3	3	0
P_1	2	1	4	1	1	2	5	3	4	1	0	2
P_2	1	3	3	1	0	3	6	4	6	0	3	0
P_3	5	4	1	2	0	0	8	4	6	3	4	1

Safe sequence: P_0, P_2, P_1, P_3
Hence, system is in safe state.

3.21 (a, d)

(a) Circular wait is one of the necessary condition for the formation of deadlock hence it is true.
(b) If the resource has more than one instance then cycle is just a necessary condition but not the sufficient condition, hence it is false.
(c) Unsafe state will not always leads to deadlock because it depends on behaviour of the processes, hence it is false.
(d) If every edge is only allocation, it means there is no requirement of the resources, so deadlock not possible, hence it is true.

4 Memory Management and Virtual Memory

4.1 In a paged segmented scheme of memory management, the segment table itself must have a page table because
(a) the segment table is often too large to fit in one page
(b) each segment is spread over a number of pages
(c) segment tables point to page table and not to the physical locations of the segment
(d) the processor's description base register points to a page table **[1995 : 1 M]**

4.2 The capacity of a memory unit is defined by the number of words multiplied by the number of bits/word. How many separate address and data lines are needed for a memory of 4 K × 16?
(a) 10 address, 16 data lines
(b) 11 address, 8 data lines
(c) 12 address, 16 data lines
(d) 12 address, 12 data lines **[1995 : 2 M]**

4.3 The address sequence generated by tracing a particular program executing in a pure demand paging system with 100 records per page with 1 free main memory frame is recorded as follows. What is the number of page faults?
0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370
(a) 13 (b) 8
(c) 7 (d) 10 **[1995 : 2 M]**

4.4 In a virtual memory system the address space specified by the address lines of the CPU must be ____ than the physical memory size and ____ than the secondary storage size.
(a) smaller, smaller (b) smaller, larger
(c) larger, smaller (d) larger, larger **[1995 : 2 M]**

4.5 A ROM is used to store the table for multiplication of two 8-bit unsigned integers. The size of ROM required is
(a) 256 × 16 (b) 64 K × 8
(c) 4K × 16 (d) 64 K × 16 **[1996 : 1 M]**

4.6 A 1000 Kbyte memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 Kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for
(a) 151 (b) 181
(c) 231 (d) 541 **[1996 : 2 M]**

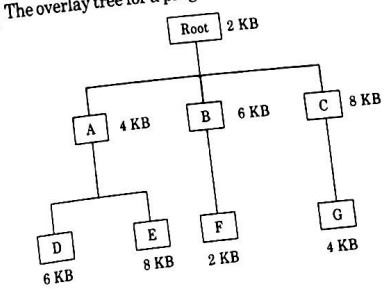
4.7 Locality of reference implies that the page reference being made by a process
(a) will always be to the page used in the previous page reference
(b) is likely to be to one of the pages used in the last few page references
(c) will always be to one of the pages existing in memory
(d) will always lead to a page fault **[1997 : 1 M]**

4.8 Thrashing
(a) Reduces page I/O
(b) Decreases the degree of multipro-gramming
(c) Implies excessive page I/O
(d) Improve the system performance **[1997 : 1 M]**

4.9 Dirty bit for a page in a page table
(a) Helps avoid unnecessary writes on paging device
(b) Helps maintain LRU information
(c) Allows only read on a page
(d) None of the above **[1997 : 1 M]**

4.10 In a resident–OS computer, which of the following system software must reside in the main memory under all situations?
(a) Assembler (b) Linker
(c) Loader (d) Compiler **[1998 : 1 M]**

4.11 The overlay tree for a program is as shown below:



What will be the size of the partition (in physical memory) required to load (and run) this program?
(a) 12 KB (b) 14 KB
(c) 10 KB (d) 8 KB
[1998 : 2 M]

4.12 If an instruction takes i microseconds and a page fault takes an additional j microseconds, the effective instruction time if on the average a page fault occurs every k instruction is
(a) $i + \frac{j}{k}$ (b) $i + j * k$
(c) $\frac{i+j}{k}$ (d) $(i + j) * k$
[1998 : 2 M]

4.13 Which of the following is/are advantage of virtual memory?
(a) Faster access to memory on an average.
(b) Processes can be given protected address spaces.
(c) Linker can assign addresses independent of where the program will be loaded in physical memory.
(d) Programs larger than the physical memory size can be run.
[1999 : 2 M]

4.14 Suppose the time to service a page fault is on the average 10 milliseconds, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in average memory access time of
(a) 1.9999 milliseconds
(b) 1 millisecond
(c) 9.999 microseconds
(d) 1.9999 microseconds
[2000 : 2 M]

4.15 Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size of 4 KB, what is the approximate size of the page table?
(a) 16 MB (b) 8 MB
(c) 2 MB (d) 24 MB
[2001 : 2 M]

4.16 Which of the following statements is/are false?
(a) Virtual memory implements the translation of a program's address space in to physical memory address space.
(b) Virtual memory allows each program to exceed the size of the primary memory.
(c) Virtual memory increases the degree of multiprogramming.
(d) Virtual memory reduces the context switching overhead.
[2001 : 1 M]

4.17 The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned addresses is called
(a) assembly (b) parsing
(c) relocation (d) symbol resolution
[2001 : 1 M]

4.18 Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will
(a) always decrease the number of page faults
(b) always increase the number of page faults
(c) sometimes increase the number of page faults
(d) never affect the number of page faults
[2001 : 1 M]

4.19 Which of the following is not a form of memory?
(a) Instruction cache
(b) Instruction register
(c) Instruction opcode
(d) Translation look a side buffer
[2002 : 1 M]

4.20 The optimal page replacement algorithm will select the page that
(a) Has not been used for the longest time in the past
(b) Will not be used for the longest time in the future
(c) Has been used least number of times
(d) Has been used most number of times
[2002 : 1 M]

4.21 Dynamic linking can cause security concerns because
(a) Security is dynamic
(b) The path for searching dynamic libraries is not known till runtime
(c) Linking is insecure
(d) Cryptographic procedures are not available for dynamic linking
[2002 : 2 M]

4.22 In a system with 32 bit virtual addresses and 1 KB page size, use of one-level page tables for virtual to physical address translation is not practical because of
(a) the large amount of internal fragmentation
(b) the large amount of external fragmentation
(c) the large memory overhead in maintaining page tables
(d) the large computation overhead in the translation process
[2003 : 1 M]

Common Data for Q. 4.23 & 4.24:
A processor uses 2-level page tables for virtual to physical address translation. Page tables for both levels are stored in the main memory. Virtual and physical addresses are both 32 bits wide. The memory is byte addressable. For virtual to physical address translation, the 10 most significant bits of the virtual address are used as index into the first level page table while the next 10 bits are used as index into the second level page table. The 12 least significant bits of the virtual address are used as offset within the page. Assume that the page table entries in both levels of page tables are 4 bytes wide. Further, the processor has a translation look-aside buffer (TLB), with a hit rate of 96%. The TLB caches recently used virtual page numbers and the corresponding physical page numbers. The processor also has a physically addressed cache with a hit ratio of 90%. Main memory access time is 10 ns, cache access time is 1 ns, and TLB access time is also 1 ns.

4.23 Assuming that no page faults occur, the average time taken to access a virtual address is approximately (to the nearest 0.5 ns)
(a) 1.5 ns (b) 2 ns
(c) 3 ns (d) 4 ns
[2003 : 2 M]

4.24 Suppose a process has only the following pages in its virtual address space: two contiguous code pages starting at virtual address 0 × 00000000, two contiguous data pages starting at virtual address 0 × 00400000, and a stack page starting at virtual address 0 × FFFFF000. The amount of memory required for storing the page tables of this process is
(a) 8 KB (b) 12 KB
(c) 16 KB (d) 20 KB
[2003 : 2 M]

4.25 Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to given 50% better benchmark results, what is the expected improvement in the I/O performance of user programs?
(a) 50% (b) 40%
(c) 25% (d) 0%
[2004 : 1 M]

4.26 The minimum number of page frames that must be allocated to a running process in a virtual memory environment is determined by
(a) the instruction set architecture
(b) page size
(c) physical memory size
(d) number of processes in memory
[2004 : 1 M]

4.27 Consider a system with a two-level paging scheme in which a regular memory access takes 150 nanoseconds, and servicing a page fault takes 8 milliseconds. An average instruction takes 100 nanoseconds of CPU time, and two memory accesses. The TLB hit ratio is 90%, and the page fault rate is one in every 10,000 instructions. What is the effective average instruction execution time?
(a) 645 nanoseconds (b) 1050 nanoseconds
(c) 1215 nanoseconds (d) 1260 nanoseconds
[2004 : 2 M]

4.28 Consider a fully associative cache with 8 cache blocks (numbered 0-7) and the following sequence of memory block requests: 4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7.
If LRU replacement policy is used, which cache block will have memory block 7?
(a) 4 (b) 5
(c) 6 (d) 7
[2004 : 2 M]

4.29 In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?
(a) 2 (b) 10
(c) 12 (d) 14
[2004 : 2 M]

4.30 For each of the four processes P_1, P_2, P_3 and P_4. The total size in kilobytes (KB) and the number of segments are given below.

Process	Total size (in KB)	Number of segments
P_1	195	4
P_2	254	5
P_3	45	3
P_4	364	8

The page size is 1 KB. The size of an entry in the page table is 4 bytes. The size of an entry in the segment table is 8 bytes. The maximum size of a segment is 256 KB. The paging method for memory management uses two-level paging, and