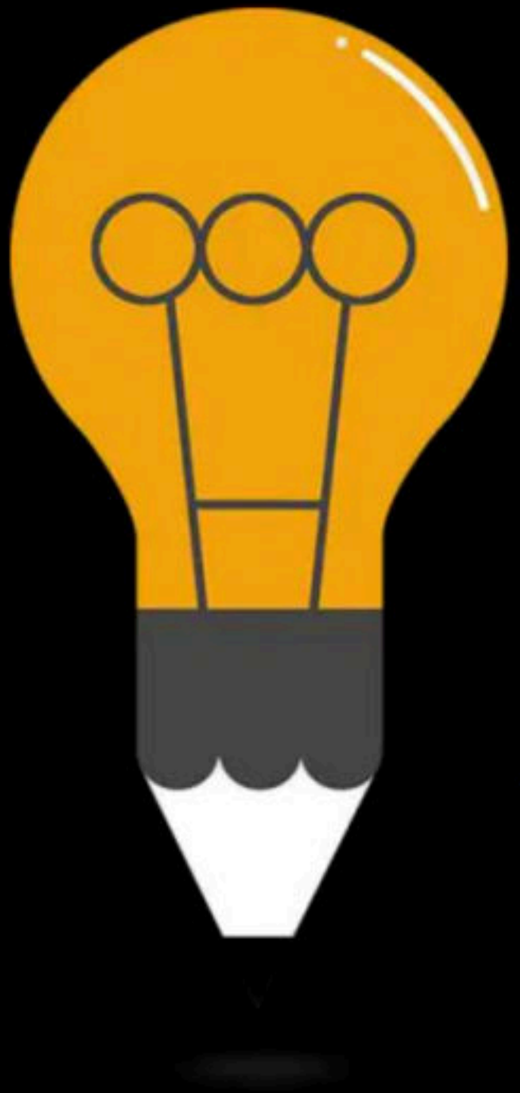
unacademy

Doubt Clearing Session & Questions on Deadlock

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 23 • May 15, 2023

unacademy



Operating System
Doubts & Deadlock Detection

By: Vishvadeep Gothi

unacademy

Quiz - 3

Ques - 1

2) $S = 1$

mutual exclusion Run processes one by one

$x = 1 + 3 - 5 + 2 = 1$

3)

P_1	P_2
$P(Sy)$	$P(Sx)$
$P(Sy)$	$P(Sy)$

$Sy = 1 \not\to 0$

$Sy = 1 \not\to 0$

no deadlock

$P(S_x)$	$P(S_x)$	$P(S_y)$	$P(S_y)$
$P(S_y)$	$P(S_y)$	$P(S_x)$	$P(S_x)$

4] 36 P(s)

9 V(s)

Successful P(s) operations = 36 − 1 = 35

1 P(s) blocked

S − 35 + 9 = 0

S = 26

5]

Ques Non-negative counting semaphore S.

Total 59 P(S)

31 V(S)

atleast 16 P(S) are blocked.

Initial value of S must be ?

Sol^n

Successful P(S) operat^ns

$59 - 16 = 43$

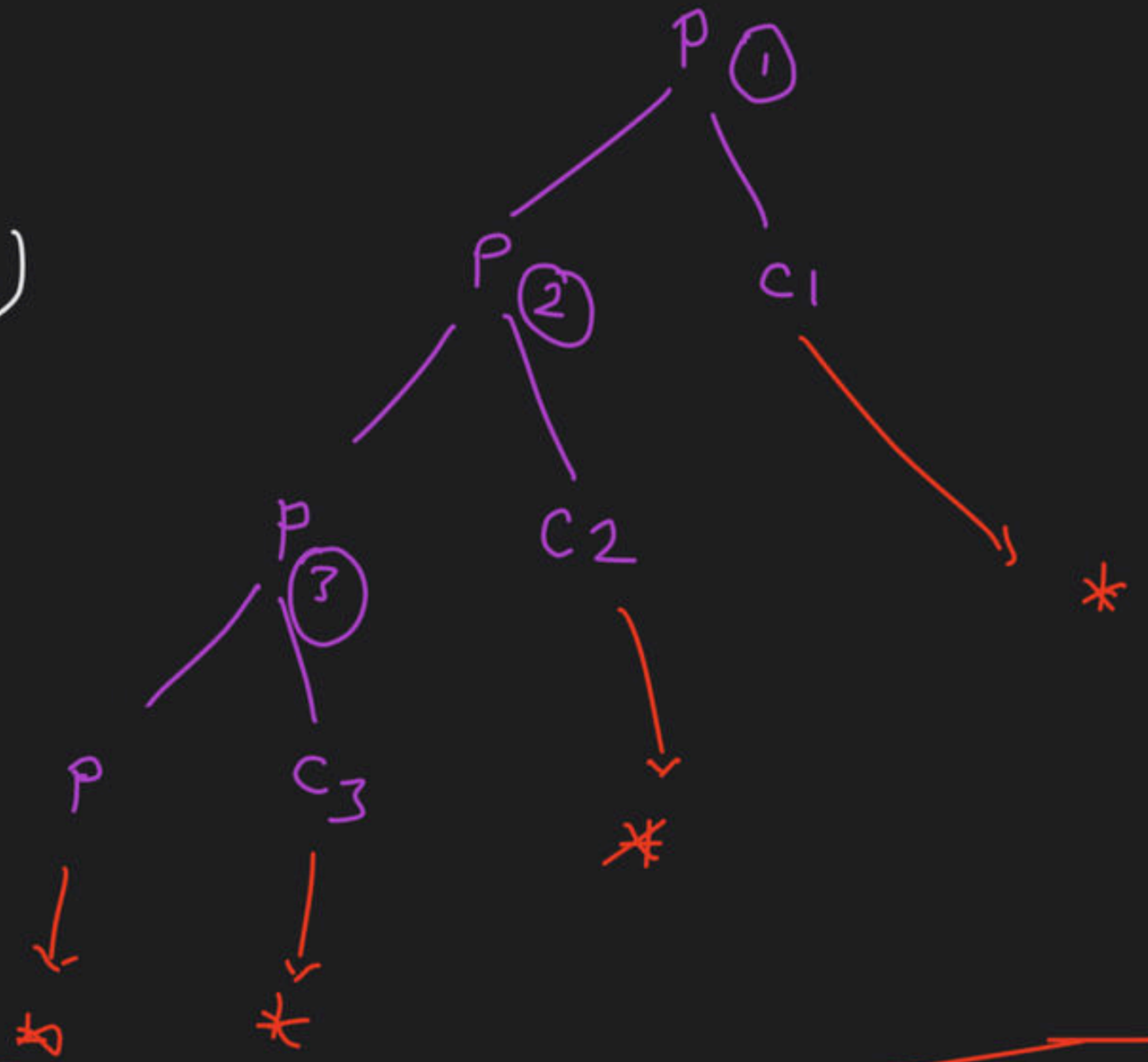
$S - 43 + 31 = 0$

$S = 12$

Ans.

Parent

```
void main()
{
  if (fork() && fork())
  {
    fork();
  }
  printf(" * ");
}
```



4 times * is printed.

Deadlock

If two or more processes are waiting for such an event which is never going to occur

Necessary Conditions for Deadlock

Deadlock can occur only when all following conditions are satisfied:

1. Mutual Exclusion
2. Hold & Wait
3. No-preemption
4. Circular Wait

Recovery From Deadlock

1. Make Sure that deadlock never occur
   - Prevent the system from deadlock or avoid deadlock
2. Allow deadlock, detect and recover
3. Pretend that there is no any deadlock

unacademy

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

need

AOK

7 4 3

0 2 0

6 0 0

~~0 1 1~~

6 0 1

4 3 1

$P_1 \Rightarrow 5\ 2\ 2$ / P_0, P_2, P_4

$P_3 \Rightarrow 7\ 4\ 3$

Question

1. What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C? Req P1 <1,0,2>
2. What will happen if process P3 requests one additional instance of resource type B? Req P3 <0,1,0>

both can be granted together

Deadlock Detection

1. When all resources have single instance
2. When resources have multiple instances

Deadlock Detection

When all resources have single instance:

Deadlock detection is done using wait-for-graph

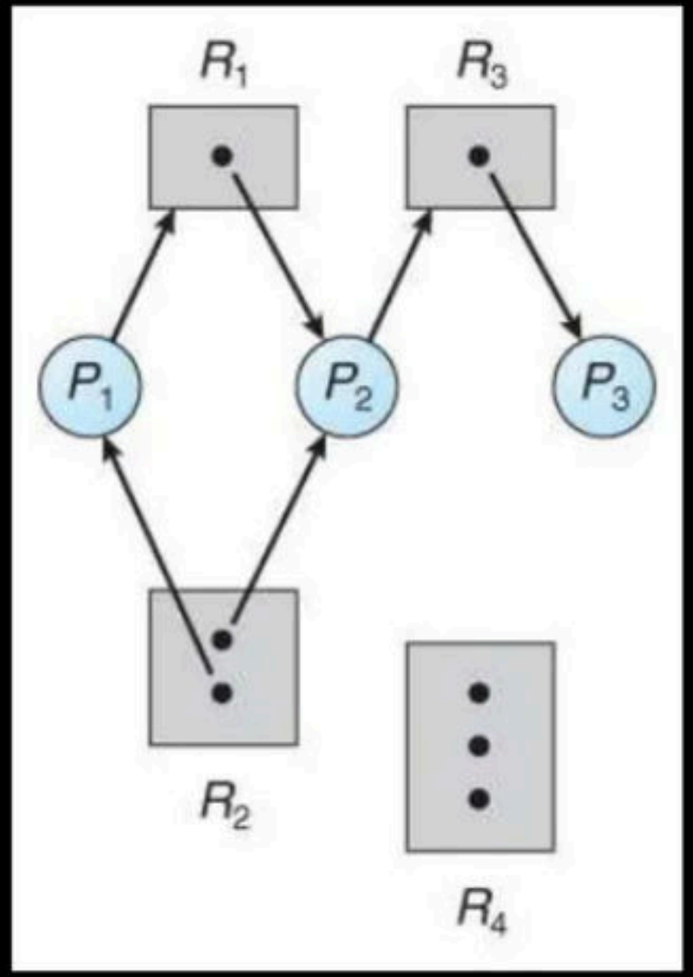
↳ construct from resource allocation graph

Wait For Graph

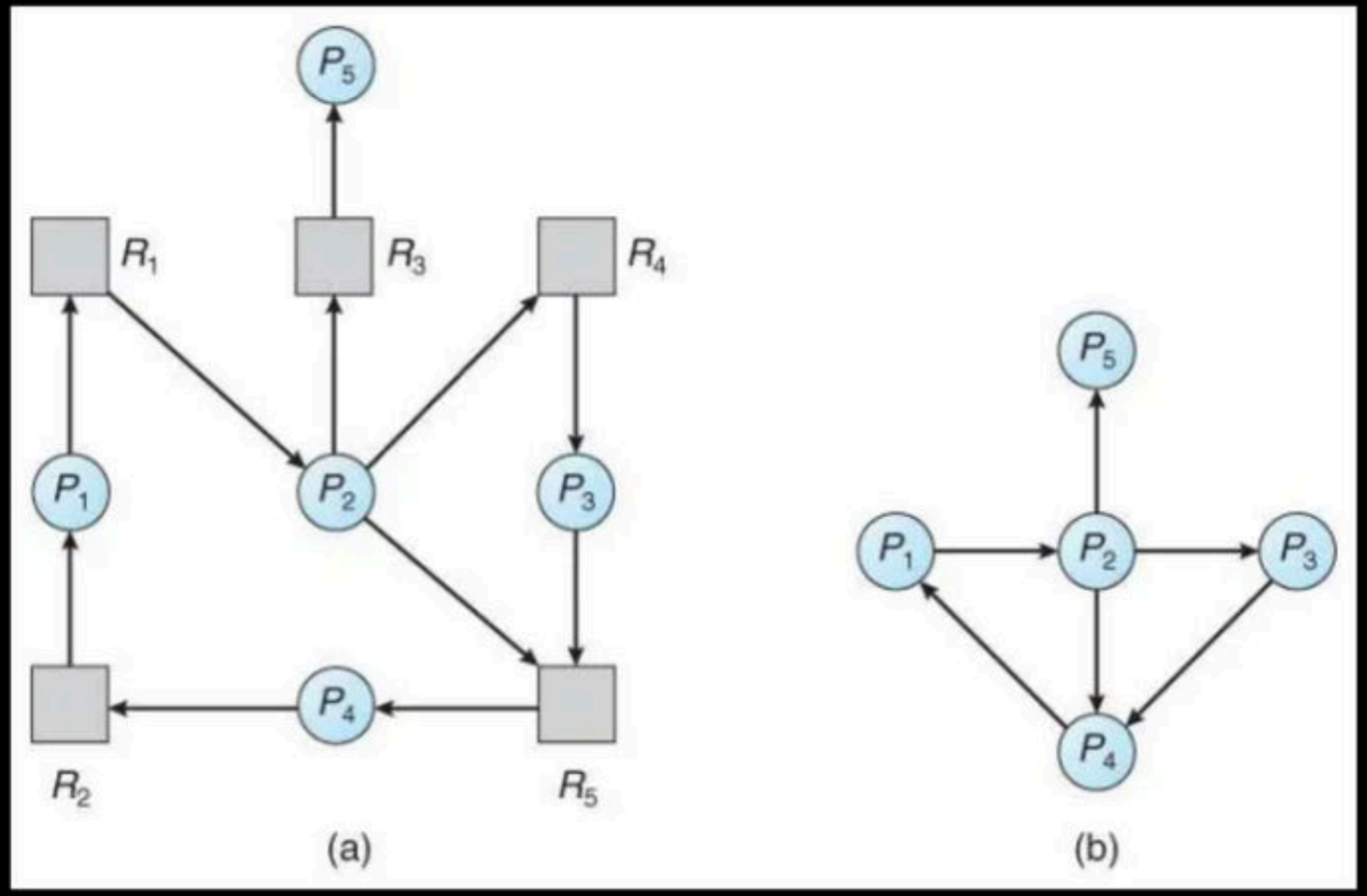
It is created from resource allocation graph

→ It has only processes.

Wait For Graph



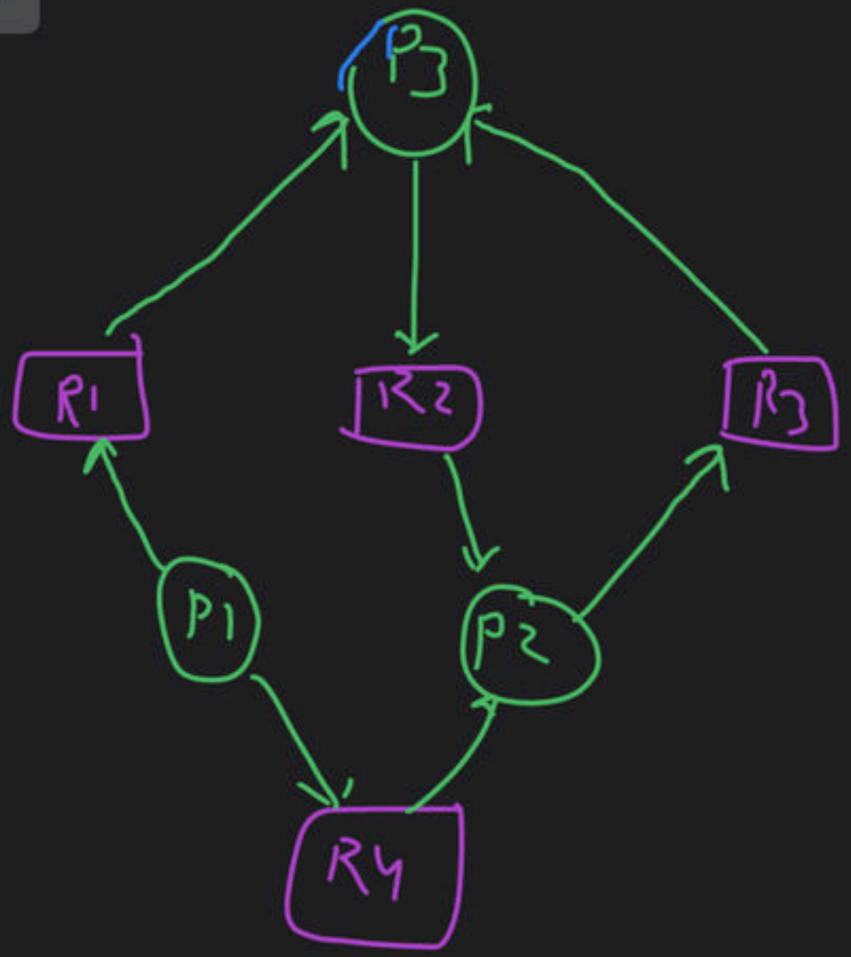
Wait For Graph

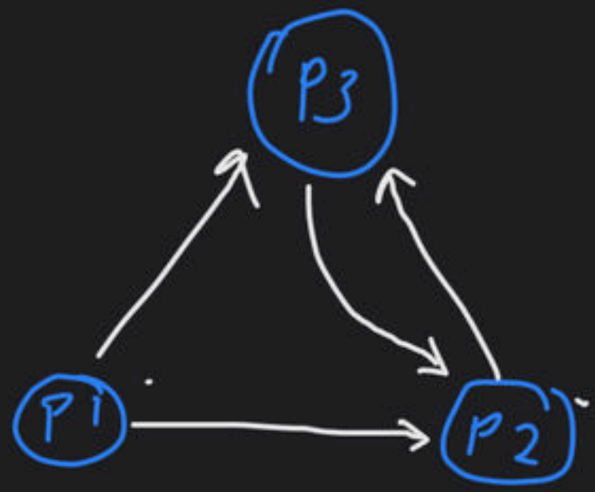


if there is any cycle in this graph

⇓

Deadlock

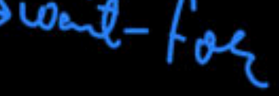




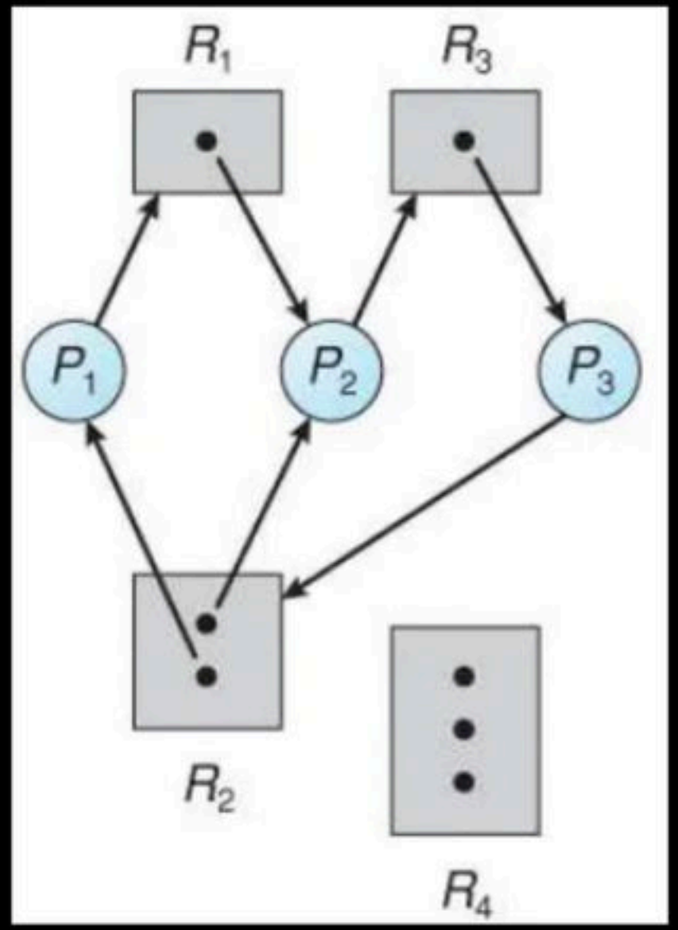
Deadlock

Wait For Graph

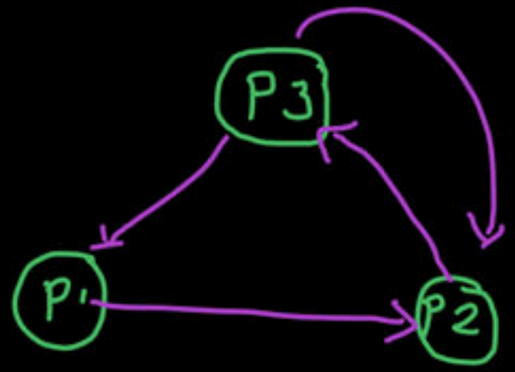
If a resource category contains more than one instance, then the presence of a cycle in the resource-allocation graph indicates the possibility of a deadlock, but does not guarantee one.



Wait For Graph: Example

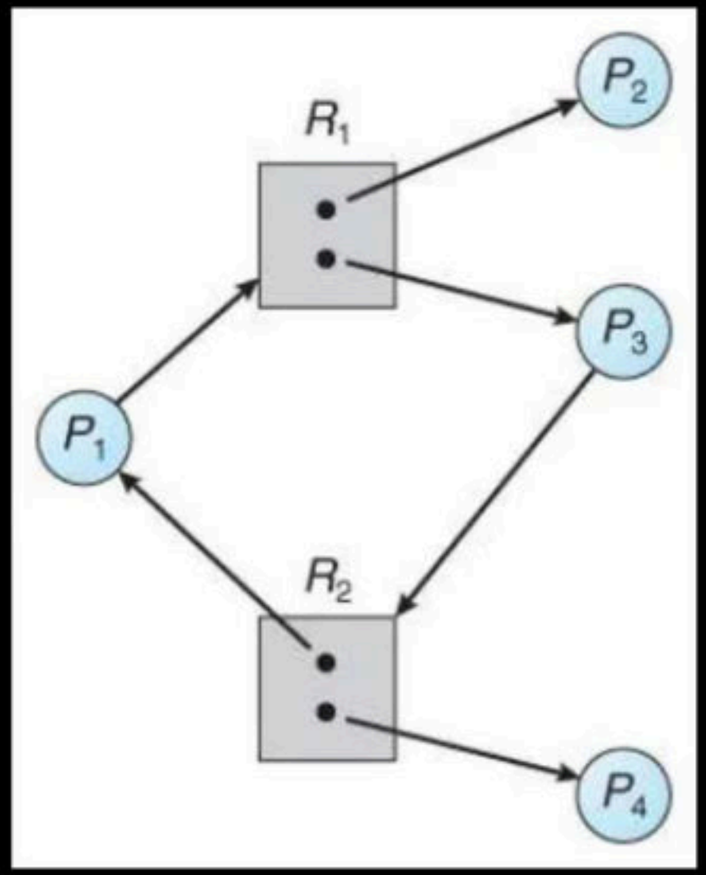


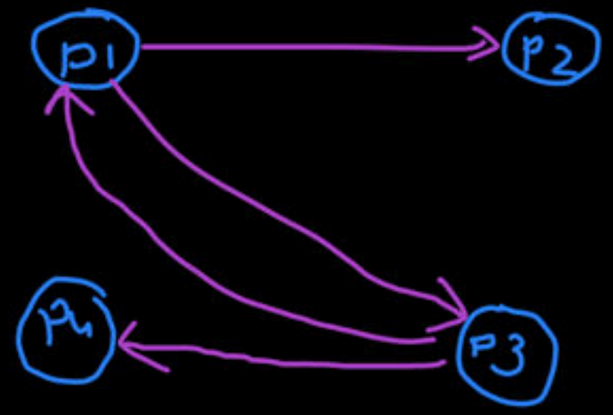
wait-for graph :-



deadlock

Wait For Graph: Example





Cycle in wait-for graph
but no deadlock

	Allocation		Request		Available	
	R1	R2	R1	R2	R1	R2
P1	0	1	1	0	0	0
P2	1	0	0	0		
P3	1	0	0	1		
P4	0	1	0	0		

after P2 => 1 0

after P1 => 1 1

after P3 => 2 1

after P4 => 2 2

No deadlock

Deadlock Detection

When resources have multiple instance:

Deadlock detection is done using a specific algorithm

↳ similar to banker's alg.

Deadlock Detection Algorithm

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	~~0 0 0~~
P_1	2 0 0	2 0 2	P0 0 1 0
P_2	3 0 3	0 0 0	P2 3 1 3
P_3	2 1 1	1 0 0	P1 5 1 3
P_4	0 0 2	0 0 2	P3 7 2 4
			P4 7 2 6

all processes can finish.
No deadlock.

if for any process P_i
$Request_i \leq available$

$available = available + Allocation_i$

All process can complete ⇒ no deadlock

Deadlock Detection Algorithm

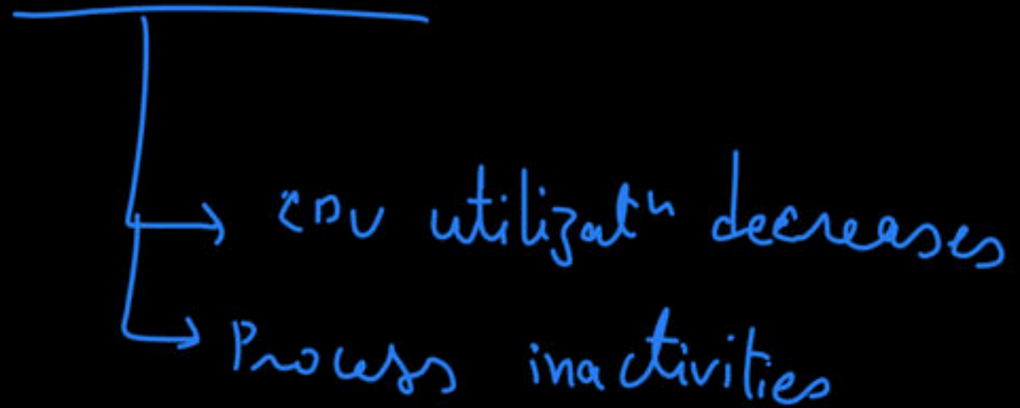
1. Let *Work* and *Finish* be vectors of length m and n respectively.
   Initialize *Work= Available*. For *i=0, 1, …., n-1*, if $Request_i$ = 0, then *Finish[i]* = true; otherwise, *Finish[i]*= false.
2. Find an index i such that both
   a) *Finish[i] == false*
   b) $Request_i$ <= *Work*
   If no such *i* exists go to step 4.
3. *Work= Work+ Allocation*$_i$
   Finish[i]= true
   Go to Step 2.
4. If *Finish[i]== false* for some i, 0<=i<n,
   then the system is in a deadlocked state.
   Moreover, if *Finish[i]==false* the process P_i is deadlocked.

Detection-Algorithm Usage

When should the deadlock detection be done? Frequently, or infrequently?

Detection-Algorithm Usage

1. Do deadlock detection after every resource allocation
2. Do deadlock detection only when there is some clue



Recovery From Deadlock

There are three basic approaches to recovery from deadlock:

1. Inform the system operator and allow him/her to take manual intervention
2. Terminate one or more processes involved in the deadlock
3. Preempt resources.

Process Termination

1. Terminate all processes involved in the deadlock
2. Terminate processes one by one until the deadlock is broken

Process Termination

Many factors that can go into deciding which processes to terminate next:

1. Process priorities.
2. How long the process has been running, and how close it is to finishing.
3. How many and what type of resources is the process holding
4. How many more resources does the process need to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

Resource Preemption

Important issues to be addressed when preempting resources to relieve deadlock:

1. Selecting a victim
2. Rollback
3. Starvation

Question 1

Consider a system with 3 processes A, B and C. All 3 processes require 4 resources each to execute. The minimum number of resources the system should have such that deadlock can never occur?

	max	allocation
A	4	3
B	4	3
C	4	3

available
1

} all 3 processes are still waiting

⇓

deadlock possibility

Ans : 3 + 3 + 3 + 1 = 10 Ans.

for n processes

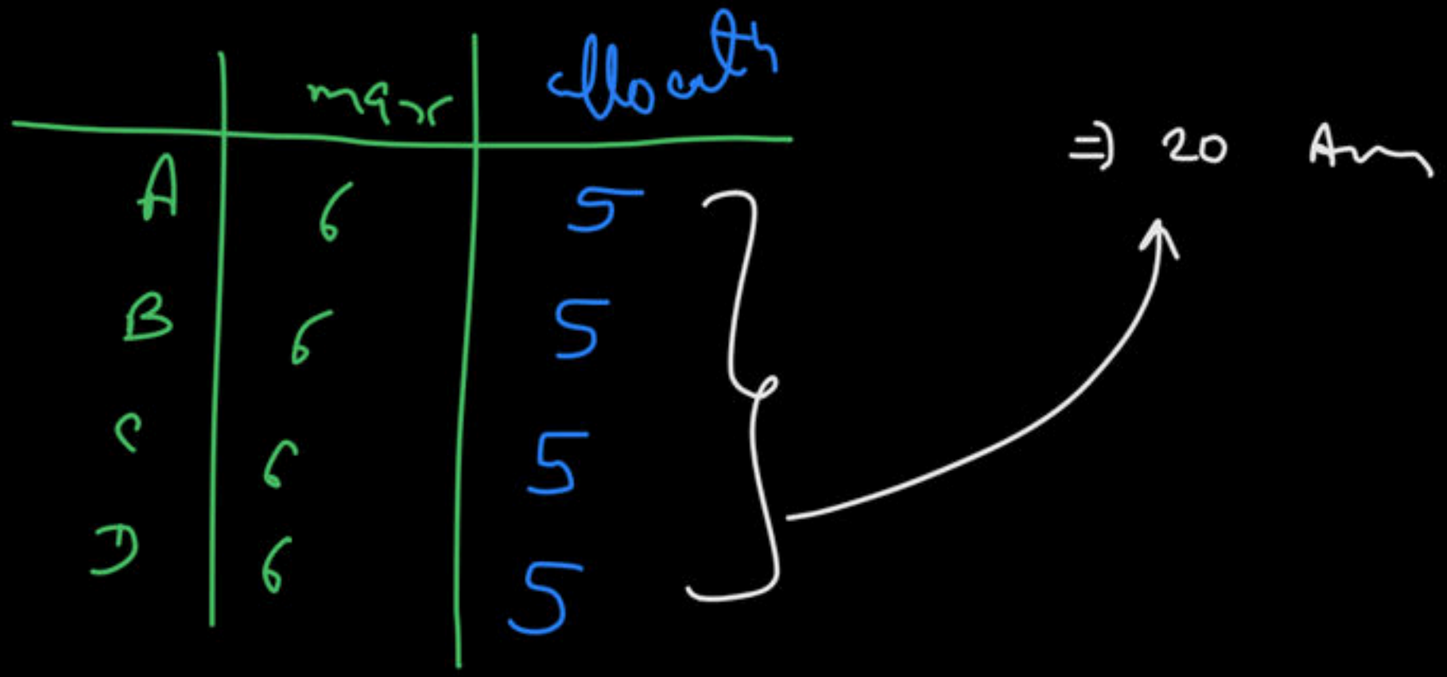
	max
P_1	m_1
⋮	⋮
P_n	m_n

min.
no. of resources needed such that deadlock never occurs

$$= \left[\sum_{i=1}^{n} (m_i - 1) \right] + 1$$

Question 2

Consider a system with 4 processes A, B, C and D. All 4 processes require 6 resources each to execute. The maximum number of resources the system should have such that deadlock may occur?



Question 3

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 ____. Ans = 2

$3 * (k-1) + 1 \leq 4$

$3k - 3 \leq 3$

$3k \leq 6$

$k \leq 2$

$k_{max} = 2$

Ques 5 process.

each require maximum x resources to execute.

no. of resources total = 15.

max value of x = (so that no deadlock in system ever)

$$5 * (x-1) + 1 \leq 15$$

$$5x - 5 \leq 14$$

$$5x \leq 19$$

$$x \leq 3.8$$

$$x_{max} = 3$$

Ques) There are n processes, each require 2 instances max to execute.

Total no. of resources are 18.

max value of n, such that there is no deadlock ever?

Ans:-

$$n * (2-1) + 1 \leq 18$$

$$n \leq 17$$

$$n_{max} = 17$$

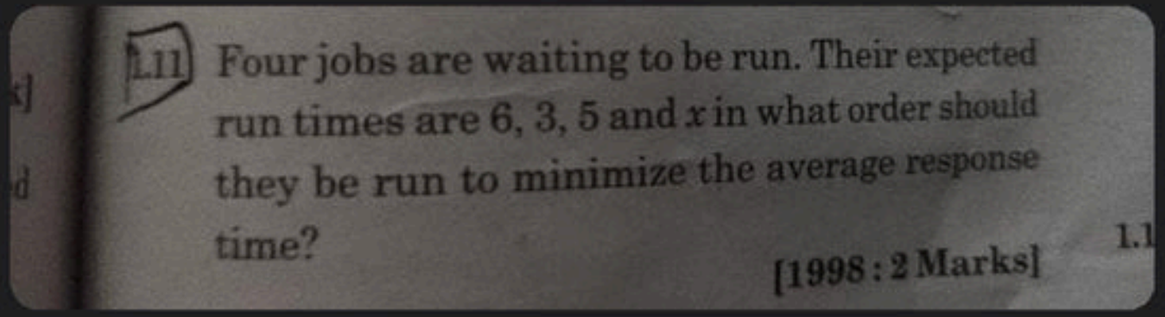
unacademy

Happy Learning.!

unacademy

▲ 1 • Asked by Shreyas

Please help me with this doubt



1.11 Four jobs are waiting to be run. Their expected run times are 6, 3, 5 and x in what order should they be run to minimize the average response time?

[1998 : 2 Marks]

$x < 3$ $x > 6$

unacademy

▲ 1 • Asked by Shreyas

Please help me with this doubt

1.44 A computer handles several interrupt sources of which the following are relevant for this question.

- Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
- Interrupt from Mouse (raises interrupt if the mouse is moved or a button is pressed)
- Interrupt from Keyboard (raises interrupt when a key is pressed or released)
- Interrupt from Hard Disk (raises interrupt when a disk read is completed)

Which one of these will be handled at the **HIGHEST** priority?

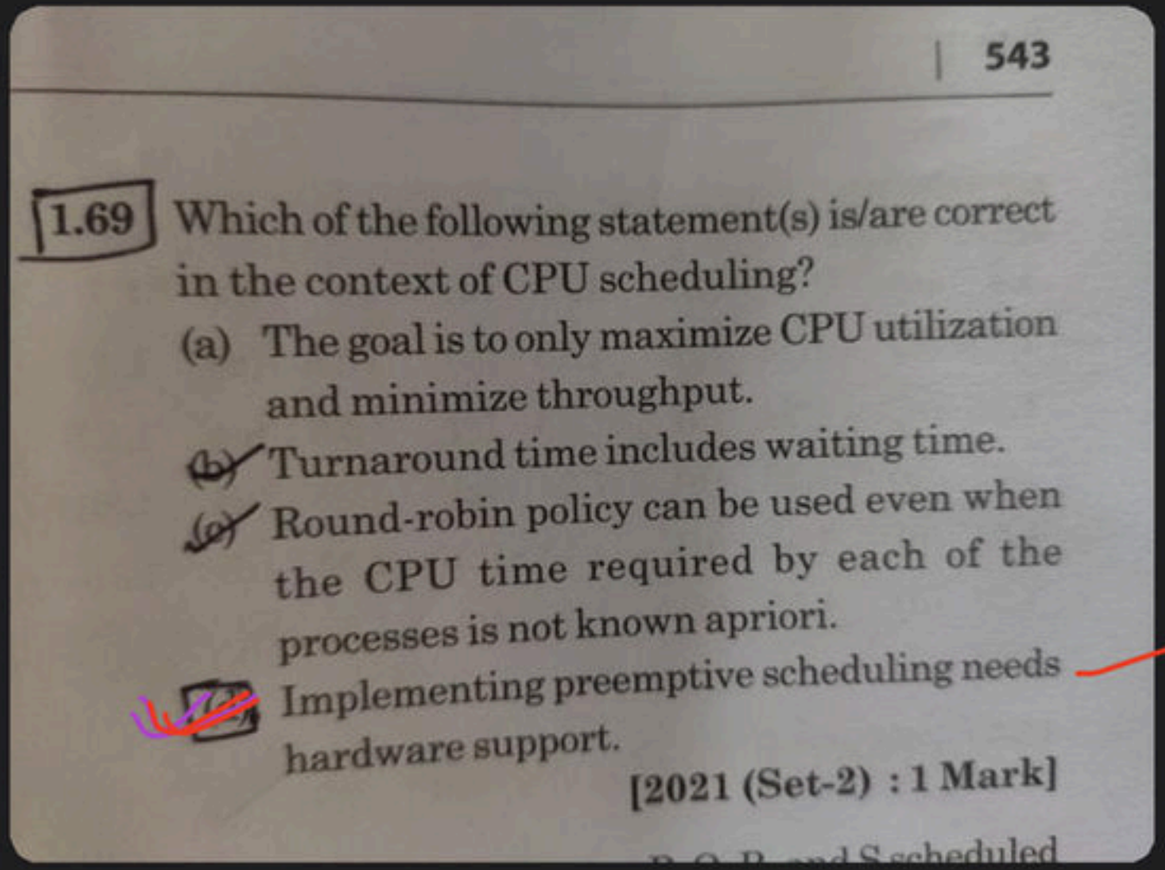
(a) Interrupt from Hard Disk
(b) Interrupt from Mouse
(c) Interrupt from Keyboard
(d) Interrupt from CPU temperature sensor

[2011 : 1 Mark]

unacademy

▲ 1 • Asked by Shreyas

Please help me with this doubt



1.69 Which of the following statement(s) is/are correct in the context of CPU scheduling?

(a) The goal is to only maximize CPU utilization and minimize throughput.

(b) Turnaround time includes waiting time.

(c) Round-robin policy can be used even when the CPU time required by each of the processes is not known apriori.

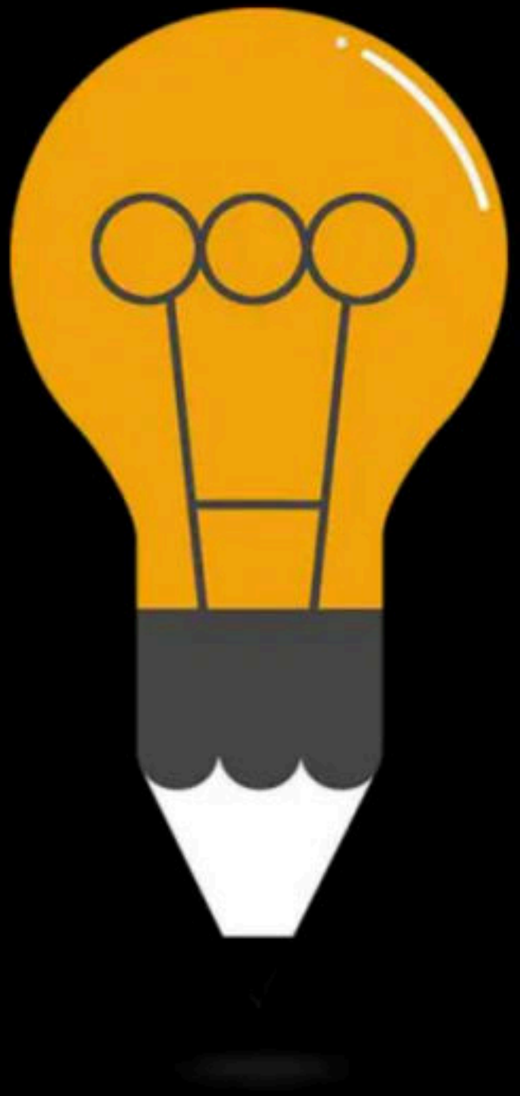
(d) Implementing preemptive scheduling needs hardware support.

[2021 (Set-2) : 1 Mark]

P, Q, R and S scheduled

→ needs interrupt implementation

unacademy



Operating System
Synchronization, Deadlock PYQs

By: Vishvadeep Gothi

GATE 1987

A critical region is

A. One which is enclosed by a pair of P and V operations on semaphores.
B. A program segment that has not been proved bug-free.
C. A program segment that often causes unexpected system crashes.
D. A program segment where shared resources are accessed.

GATE 1990

Match the pairs in the following questions:

(a) Critical region	(p) Hoare's monitor
(b) Wait/Signal	(q) Mutual exclusion
(c) Working Set	(r) Principle of locality
(d) Deadlock	(s) Circular Wait

GATE 1996

A critical section is a program segment

A. which should run in a certain amount of time
B. which avoids deadlocks
C. where shared resources are accessed
D. which must be enclosed by a pair of semaphore operations, P and V

GATE 1996

A solution to the Dining Philosophers Problem which avoids deadlock is to

A. ensure that all philosophers pick up the left fork before the right fork

B. ensure that all philosophers pick up the right fork before the left fork

C. ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork

D. None of the above

unacademy

GATE 1997

Each Process P_i, i = $1....9$ is coded as follows

```
repeat
    P(mutex)
    {Critical section}
    V(mutex)
forever
```

The code for P_{10} is identical except it uses V(mutex) in place of P(mutex). What is the largest number of processes that can be inside the critical section at any moment?

A. 1
B. 2
C. 3
D. None

unacademy

GATE 1998

When the result of a computation depends on the speed of the processes involved, there is said to be

A. cycle stealing
B. race condition
C. a time lock
D. a deadlock

unacademy

GATE 2000

Let m[0]....m[4] be mutexes (binary semaphores) and P[0].....P[4] be processes. Suppose each process P[i] executes the following:

```
wait (m[i]; wait (m(i+1) mode 4]);
.....
release (m[i]); release (m(i+1) mod 4]);
```

This could cause

A. Thrashing
B. Deadlock
C. Starvation, but not deadlock
D. None of the above

unacademy

unacademy

GATE 2001

Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below.

```
repeat
     flag[i] = true;
     turn = j;
     while (P) do no-op;
     Enter critical section, perform actions, then
     exit critical section
     Flag[i] = false;
     Perform other non-critical section actions.
Until false;
```

For the program to guarantee mutual exclusion, the predicate P in the while loop should be

A. flag[j] = true and turn = i
B. flag[j] = true and turn = j
C. flag[i] = true and turn = j
D. flag[i] = true and turn = i

GATE 2003

```
Process P:          Process Q:
while(1) {          while(1) {
W:                  Y:
   print '0';          print '1';
   print '0';          print '1';
X:                  Z:
}                   }
```

Synchronization statements can be inserted only at points W, X, Y, and Z

Which of the following will always lead to an output staring with '001100110011'?

A. P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1

B. P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0

C. P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1

D. P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S initially 1 , and T initially 0

unacademy

GATE 2003

Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below.

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

Synchronization statements can be inserted only at points W, X, Y, and Z

Which of the following will ensure that the output string never contains a substring of the form 01^n0 and 10^n1 where n is odd?

A. P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
B. P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
C. P(S) at W, V(S) at X, P(S) at Y, V(S) at Z, S initially 1
D. V(S) at W, V(T) at X, P(S) at Y, P(T) at Z, S and T initially 1

unacademy

GATE 2004

Consider two processes P_1 and P_2 accessing the shared variables X and Y protected by two binary semaphores S_X and S_Y respectively, both initialized to 1. P and V denote the usual semaphore operators, where P decrements the semaphore value, and V increments the semaphore value. The pseudo-code of P_1 and P_2 is as follows:

P_1:	P_2:
While true do {	While true do {
L_1:.....	L_3:.....
L_2:.....	L_4:.....
X = X + 1;	Y = Y + 1;
Y = Y -1;	X = Y -1;
$V(S_X)$;	$V(S_Y)$;
$V(S_Y)$; }	$V(S_X)$;

In order to avoid deadlock, the correct operators at L_1, L_2, L_3 and L_4 are respectively.

A. $P(S_Y), P(S_X); P(S_X), P(S_Y)$

B. $P(S_X), P(S_Y); P(S_Y), P(S_X)$

C. $P(S_X), P(S_X); P(S_Y), P(S_Y)$

D. $P(S_X), P(S_Y); P(S_X), P(S_Y)$

GATE 2004

The semaphore variables full, empty and mutex are initialized to 0, n and 1, respectively. Process P_1 repeatedly adds one item at a time to a buffer of size n, and process P_2 repeatedly removes one item at a time from the same buffer using the programs given below. In the programs, K, L, M and N are unspecified statements.

P_1

```
while (1) {
    K;
    P(mutex);
    Add an item to the buffer;
    V(mutex);
    L;
}
```

P_2

```
while (1) {
    M;
    P(mutex);
    Remove an item from the buffer;
    V(mutex);
    N;
}
```

The statements K, L, M and N are respectively

A. P(full), V(empty), P(full), V(empty)
B. P(full), V(empty), P(empty), V(full)
C. P(empty), V(full), P(empty), V(full)
D. P(empty), V(full), P(full), V(empty)

GATE 2005

Given below is a program which when executed spawns two concurrent processes :
semaphore X : = 0 ;
/* Process now forks into concurrent processes P1 & P2 */

P1	**P2**
repeat forever	repeat
V (X) ;	forever
Compute ;	P(X) ;
	Compute ;
P(X) ;	V(X) ;

Consider the following statements about processes P1 and P2:

I. It is possible for process P1 to starve.
II. It is possible for process P2 to starve.

Which of the following holds?

A. Both I and II are true.
B. I is true but II is false.
C. II is true but I is false
D. Both I and II are false

GATE 2005

Two concurrent processes P1 and P2 use four shared resources R1, R2, R3 and R4, as shown below.

P1	**P2**
Compute:	Compute;
Use $R1$;	Use $R1$;
Use $R2$;	Use $R2$;
Use $R3$;	Use $R3$;
Use $R4$;	Use $R4$;

Both processes are started at the same time, and each resource can be accessed by only one process at a time The following scheduling constraints exist between the access of resources by the processes:

- P2 must complete use of R1 before P1 gets access to R1.
- P1 must complete use of R2 before P2 gets access to R2.
- P2 must complete use of R3 before P1 gets access to R3.
- P1 must complete use of R4 before P2 gets access to R4.

There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed?

GATE 2006

The atomic *fetch-and-set* x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. Consider the following implementation of P and V functions on a binary semaphore S.

```
void P (binary_semaphore *s) {
    unsigned y;
    unsigned *x = &(s->value);
    do {
        fetch-and-set x, y;
    } while (y);
}

void V (binary_semaphore *s) {
    S->value = 0;
}
```

Which one of the following is true?

A. The implementation may not work if context switching is disabled in P
B. Instead of using *fetch-and –set*, a pair of normal load/store can be used
C. The implementation of V is wrong
D. The code does not implement a binary semaphore

unacademy

GATE 2006

Barrier is a synchronization construct where a set of processes synchronizes globally i.e., each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on left.

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

GATE 2006

The variables *process_arrived* and *process_left* are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

The above implementation of barrier is incorrect. Which one of the following is true?

A. The barrier implementation is wrong due to the use of binary semaphore S
B. The barrier implementation may lead to a deadlock if two barrier in invocations are used in immediate succession.
C. Lines 6 to 10 need not be inside a critical section
D. The barrier implementation is correct if there are only two processes instead of three.

unacademy

GATE 2006

The variables process_arrived and process_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

Which one of the following rectifies the problem in the implementation?

A. Lines 6 to 10 are simply replaced by process_arrived--
B. At the beginning of the barrier the first process to enter the barrier waits until process_arrived becomes zero before proceeding to execute P(S).
C. Context switch is disabled at the beginning of the barrier and re-enabled at the end.
D. The variable process_left is made private instead of shared

unacademy

GATE 2006

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);	Consume 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

synchronization construct used by the processes:

```
/*   P1    */
while (true) {
     wants1 = true;
     while (wants2 == true);
     /* Critical Section */
     wants1 = false;
}
/* Remainder section */
```

```
/*   P2    */
while (true) {
     wants2 = true;
     while (wants1 == true);
     /* Critical Section */
     wants2=false;
}
/* Remainder section */
```

Here, wants1 and wants2 are shared variables, which are initialized to false.

Which one of the following statements is TRUE about the construct?

A. It does not ensure mutual exclusion.

B. It does not ensure bounded waiting.

C. It requires that processes enter the critical section in strict alteration.

D. It does not prevent deadlocks, but ensures mutual exclusion.

unacademy

GATE 2007

Processes P1 and P2 use critical_flag in the following routine to achieve mutual exclusion. Assume that critical_flag is initialized to FALSE in the main program.

```
get_exclusive_access ( )
{
    if (critical _flag == FALSE) {
        critical_flag = TRUE ;
        critical_region () ;
        critical_flag = FALSE;
    }
}
```

Consider the following statements.

i. It is possible for both P1 and P2 to access critical_region concurrently.
ii. This may lead to a deadlock.

Which of the following holds?

A. i is flase ii is true
B. Both i and ii are false
C. i is true ii is flase
D. Both i and ii are true

unacademy

GATE 2007

Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization

```
wait (wrt)
writing is performed
signal (wrt)
wait (mutex)
readcount = readcount + 1
if readcount = 1 then S1
S2
reading is performed
S3
readcount = readcount - 1
if readcount = 0 then S4
signal (mutex)
```

The values of S1, S2, S3, S4, (in that order) are

A. signal (mutex), wait (wrt), signal (wrt), wait (mutex)
B. signal (wrt), signal (mutex), wait (mutex), wait (wrt)
C. wait (wrt), signal (mutex), wait (mutex), signal (wrt)
D. signal (mutex), wait (mutex), signal (mutex), wait (mutex)

unacademy

GATE 2008

The following is a code with two threads, producer and consumer, that can run in parallel. Further, S and Q are binary semaphores quipped 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

The **enter_CS()** and **leave_CS()** functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
    while(test-and-set(X));
}

void leave_CS(X)
{
    X = 0;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements:

I. The above solution to CS problem is deadlock-free

II. The solution is starvation free

III. The processes enter CS in FIFO order

IV. More than one process can enter CS at the same time

Which of the above statements are TRUE?

A. I only
B. I and II
C. II and III
D. IV only

unacademy

GATE 2010

Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared boolean variables S1 and S2 are randomly assigned.

Method used by P1	**Method used by P2**

`while (S1==S2);`
`Critical Section`
`S1=S2;`
	`while (S1!=S2);`
	`Critical Section`
	`S2 = not(S1)`

Which one of the following statements describes the properties achieved?

A. Mutual excusion but not progress
B. Progress but not mutual exclusion
C. Neither mutual exclusion nor progress
D. Both mutual exclusion and progress

unacademy

GATE 2010

The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0.

Process P0	Process P1	Process P2

while (true) {
wait (S0);
print '0';
release (S1);
release (S2);
}
	wait (S1);
	release (S0);
		wait (S2);
		release (S0);

How many times will process P0 print '0'?

A. At least twice
B. Exactly twice
C. Exactly thrice
D. Exactly once

unacademy

GATE 2012

Fetch_And_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock(L){
    while (Fetch_And_Add(L,1))
        L = 1;
}

ReleaseLock(L){
    L = 0;
}
```

This implementation

A. fails as L can overflow
B. fails as L can take on a non-zero value when the lock is actually available
C. works correctly but may starve some processes
D. works correctly without starvation

A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?

A. –2

B. –1

C. 1

D. 2

A certain computation generates two arrays a and b such that a[i] = f(i) for $0 \leq i < n$ and b[i] = g(a[i]) for $0 \leq i < n$. Suppose this computation is decomposed into two concurrent processes X and Y such that X computes the array a and Y computes the array b. The processes employ two binary semaphores R and S, both initialized to zero. The array a is shared by the two processes. The structures of the processes are shown below.

Process X:

```
private i;
for (i=0; i< n; i++) {

 a[i] = f(i);

 ExitX(R, S);

}
```

Process Y:

```
private i;
for (i=0; i< n; i++) {

 EntryY(R, S);

 b[i] = g(a[i]);

 }
```

Which one of the following represents the **CORRECT** implementations of ExitX and EntryY?

A.
```
ExitX(R, S) {
P(R);
V(S);
}
EntryY(R, S) {
P(S);
V(R);
}
```

B.
```
ExitX(R, S) {
V(R);
V(S);
}
EntryY(R, S) {
P(R);
P(S);
}
```

C.
```
ExitX(R, S) {
P(S);
V(R);
}
EntryY(R, S) {
V(S);
P(R);
}
```

D.
```
ExitX(R, S) {
V(R);
P(S);
}
EntryY(R, S) {
V(S);
P(R);
}
```

unacademy

GATE 2014

Consider the procedure below for the *Producer-Consumer* problem which uses semaphores:

```
semaphore n = 0;
semaphore s = 1;
```

```
void producer()
{
    while(true)
    {
        produce();
        semWait(s);
        addToBuffer();
        semSignal(s);
        semSignal(n);
    }
}
```

```
void consumer()
{
    while(true)
    {
        semWait(s);
        semWait(n);
        removeFromBuffer();
        semSignal(s);
        consume();
    }
}
```

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

A. The producer will be able to add an item to the buffer, but the consumer can never consume it.
B. The consumer will remove no more than one item from the buffer.
C. Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.
D. The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.

The following two functions $P1$ and $P2$ that share a variable B with an initial value of 2 execute concurrently.

```
P1() {
    C = B - 1;
    B = 2 * C;
}
```

```
P2(){
    D = 2 * B;
    B = D - 1;
}
```

The number of distinct values that B can possibly take after the execution is_____.

GATE 2015

Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes

Process X

```
/* other code for process x*/
while (true)  {
     varP = true;
     while (varQ == true)
     {
          /* Critical Section */
          varP = false;
     }
}
/* other code for process X */
```

Process Y

```
/* other code for process Y */
while (true)
{
     varQ = true;
     while (varP == true)
     {
          /* Critical Section */
          varQ = false;
     }
}
/* other code for process Y */
```

Here *varP* and *varQ* are shared variables and both are initialized to false. Which one of the following statements is true?

A. The proposed solution prevents deadlock but fails to guarantee mutual exclusion
B. The proposed solution guarantees mutual exclusion but fails to prevent deadlock
C. The proposed solution guarantees mutual exclusion and prevents deadlock
D. The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion

unacademy

GATE 2016

Consider the following two-process synchronization solution.

```
PROCESS 0                                   Process 1

Entry: loop while (turn == 1);              Entry: loop while (turn == 0);
   (critical section)                          (critical section)
Exit: turn = 1;                             Exit turn = 0;
```

The shared variable turn is initialized to zero . Which one of the following is TRUE?

A. This is a correct two- process synchronization solution.
B. This solution violates mutual exclusion requirement.
C. This solution violates progress requirement.
D. This solution violates bounded wait requirement.

unacademy

GATE 2017

A multithreaded program P executes with x number of threads and uses y number of locks for ensuring mutual exclusion while operating on shared memory locations. All locks in the program are *non-reentrant*, i.e., if a thread holds a lock l, then it cannot re-acquire lock l without releasing it. If a thread is unable to acquire a lock, it blocks until the lock becomes available. The *minimum* value of x and the *minimum* value of y together for which execution of P can result in a deadlock are:

A. $x = 1, y = 2$
B. $x = 2, y = 1$
C. $x = 2, y = 2$
D. $x = 1, y = 1$

unacademy

GATE 2018

Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores $empty$, $full$ and $mutex$ are defined with respective initial values of 0, N and 1. Semaphore $empty$ denotes the number of available slots in the buffer, for the consumer to read from. Semaphore $full$ denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either $empty$ or $full$. The valid semaphore operations are: $wait()$ and $sigmal()$.

Producer :	*Consumer* :

`do {`
`  wait (P);`
`  wait (mutex);`
`  //Add item to buffer`
`  signal (mutex);`
`  signal (Q);`
`}while(1);`
	`do {`
	`  wait (R);`
	`  wait (mutex);`
	`  //consume item from buffer`
	`  signal (mutex);`
	`  signal(S);`
	`}while(1)`

Which one of the following assignments tp P, Q, R and S will yield the correct solution?

A. P: full, Q:full, R:empty, S:empty
B. P: empty, empty, R:full, S:full
C. P: full, Q:empty, R:empty, S:full
D. P: empty, Q:full, R:full, S:empty

Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100.

P1	P2	P3
: : D = D + 20 : :	: : D = D – 50 : :	: : D = D + 10 : :

The processes are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then the value of Y – X is _____.

unacademy

GATE 2021

Consider the following pseudocode, where S is a semaphore initialized to 5 in line $\#2$ and **counter** is a shared variable initialized to 0 in line $\#1$. Assume that the increment operation in line $\#7$ is not atomic.

```
1.  int counter = 0;
2.  Semaphore S = init(5);
3.  void parop(void)
4.  {
5.          wait(S);
6.          wait(S);
7.          counter++;
8.          signal(S);
9.          signal(S);
10. }
```

If five threads execute the function **parop** concurrently, which of the following program behavior(s) is/are possible?

A. The value of **counter** is 5 after all the threads successfully complete the execution of **parop**

B. The value of **counter** is 1 after all the threads successfully complete the execution of **parop**

C. The value of **counter** is 0 after all the threads successfully complete the execution of **parop**

D. There is a deadlock involving all the threads

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 ensures that deadlocks will not occur

B. The scheme may lead to live-lock

C. The scheme may lead to starvation

D. The scheme violates the mutual exclusion property

unacademy

GATE-1990

Fill in the blanks:

Semaphore operations are atomic because they are implemented within the OS _____.

GATE-1992

At a particular time of computation, the value of a counting semaphore is 7. Then 20 P operations and 15 V operations were completed on this semaphore. The resulting value of the semaphore is :

A. 42
B. 2
C. 7
D. 12

unacademy

GATE-1998

A counting semaphore was initialized to 10. Then $6P$ (wait) operations and $4V$ (signal) operations were completed on this semaphore. The resulting value of the semaphore is

A. 0
B. 8
C. 10
D. 12

unacademy

GATE-2006

The wait and signal operations of a monitor are implemented using semaphores as follows. In the following,

- x is a condition variable,
- mutex is a semaphore initialized to 1,
- x_sem is a semaphore initialized to 0,
- x_count is the number of processes waiting on semaphore x_sem, initially 0,
- next is a semaphore initialized to 0,
- next_count is the number of processes waiting on semaphore next, initially 0.

The body of each procedure that is visible outside the monitor is replaced with the following:

```
P(mutex);
...
body of procedure
...
if (next_count > 0)
    V(next);
else
    V(mutex);
```

Each occurrence of x.wait is replaced with the following:

```
x_count = x_count + 1;
if (next_count > 0)
    V(next);
else
    V(mutex);
---------------------------------------------------------- El;
x_count = x_count - 1;
```

GATE-2006

Each occurrence of x.signal is replaced with the following:

```
if (x_count > 0)
{
    next_count = next_count + 1;
    ------------------- E2;
    P(next);
    next_count = next_count - 1;
}
```

For correct implementation of the monitor, statements E1 and E2 are, respectively,

A. P(x_sem), V(next)
B. V(next), P(x_sem)
C. P(next), V(x_sem)
D. P(x_sem), V(x_sem)

The P and V operations on counting semaphores, where s is a counting semaphore, are defined as follows:

$P(s)$: $s = s - 1;$
If $s < 0$ then wait;

$V(s)$: $s = s + 1;$
If $s \leq 0$ then wake up process waiting on s;

Assume that P_b and V_b the wait and signal operations on binary semaphores are provided. Two binary semaphores x_b and y_b are used to implement the semaphore operations $P(s)$ and $V(s)$ as follows:

$P(s)$:
```
P_b(x_b);
s = s - 1;
if (s < 0)
{
    V_b(x_b);
    P_b(y_b);
}
else V_b(x_b);
```

$V(s)$:
```
P_b(x_b);
s = s + 1;
if (s ≤ 0) V_b(y_b)
;
V_b(x_b);
```

GATE-2008

The initial values of x_b and y_b are respectively

A. 0 and 0
B. 0 and 1
C. 1 and 0
D. 1 and 1

unacademy

GATE-2016

Consider a non-negative counting semaphore S. The operation $P(S)$ decrements S, and $V(S)$ increments S. During an execution, 20 $P(S)$ operations and 12 $V(S)$ operations are issued in some order. The largest initial value of S for which at least one $P(S)$ operation will remain blocked is

_____

GATE-1989

(i) A system of four concurrent processes, P, Q, R and S, use shared resources A, B and C. The sequences in which processes, P, Q, R and S request and release resources are as follows:

```
Process P:  1. P requests A
            2. P requests B
            3. P releases A
            4. P releases B

Process Q:  1. Q requests C
            2. Q requests A
            3. Q releases C
            4. Q releases A
Process R:  1. R requests B
            2. R requests C
            3. R releases B
            4. R releases C
Process S:  1. S requests A
            2. S requests C
            3. S releases A
            4. S releases C
```

If a resource is free, it is granted to a requesting process immediately. There is no preemption of granted resources. A resource is taken back from a process only when the process explicity releases it.

Can the system of four processes get into a deadlock? If yes, give a sequence (ordering) of operations (for requesting and releasing resources) of these processes which leads to a deadlock.

(ii) Will the processes always get into a deadlock? If your answer is no, give a sequence of these operations which leads to completion of all processes.

(iii) What strategies can be used to prevent deadlocks in a system of concurrent processes using shared resources if preemption of granted resources is not allowed?

GATE-1992

A computer system has 6 tape devices, with n processes competing for them. Each process may need 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock-free is:

A. 2
B. 3
C. 4
D. 1

unacademy

GATE-1993

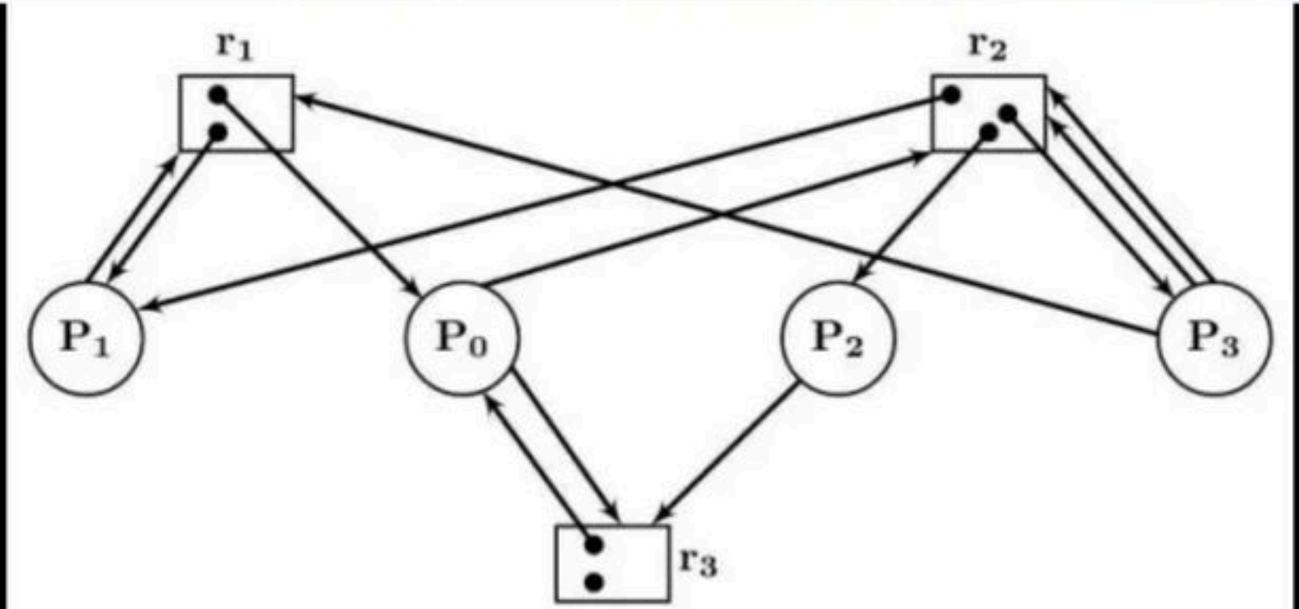
Consider a system having m resources of the same type. These resources are shared by 3 processes A,B, and C which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur?

A. 7
B. 9
C. 10
D. 13
E. 15

unacademy

GATE-1994

Consider the resource allocation graph in the figure.



A. Find if the system is in a deadlock state

B. Otherwise, find a safe sequence

unacademy

GATE-1996

A computer system uses the Banker's Algorithm to deal with deadlocks. Its current state is shown in the table below, where $P0$, $P1$, $P2$ are processes, and $R0$, $R1$, $R2$ are resources types.

	Maximum Need			Current Allocation			Available		
	R0	R1	R2	R0	R1	R2	R0	R1	R2
P0	4	1	2	1	0	2	2	2	0
P1	1	5	1	0	3	1			
P2	1	2	3	1	0	2			

A. Show that the system can be in this state

B. What will the system do on a request by process $P0$ for one unit of resource type $R1$?

GATE-1997

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

GATE-1998

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

GATE-2000

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.

GATE-2001

Two concurrent processes $P1$ and $P2$ want to use resources $R1$ and $R2$ in a mutually exclusive manner. Initially, $R1$ and $R2$ are free. The programs executed by the two processes are given below.

	Program for P1:		*Program for P2:*
S1:	While (R1 is busy) do no-op;	Q1:	While (R1 is busy) do no-op;
S2:	Set R1 ← busy;	Q2:	Set R1 ← busy;
S3:	While (R2 is busy) do no-op;	Q3:	While (R2 is busy) do no-op;
S4:	Set R2 ← busy;	Q4:	Set R2 ← busy;
S5:	Use R1 and R2;	Q5:	Use R1 and R2;
S6:	Set R1 ← free;	Q6:	Set R2 ← free;
S7:	Set R2 ← free;	Q7:	Set R1 ← free;

A. Is mutual exclusion guaranteed for $R1$ and $R2$? If not show a possible interleaving of the statements of $P1$ and $P2$ such mutual exclusion is violated (i.e., both $P1$ and $P2$ use $R1$ and $R2$ at the same time).

B. Can deadlock occur in the above program? If yes, show a possible interleaving of the statements of $P1$ and $P2$ leading to deadlock.

C. Exchange the statements $Q1$ and $Q3$ and statements $Q2$ and $Q4$. Is mutual exclusion guaranteed now? Can deadlock occur?

unacademy

GATE-2004

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(Pr) < T(Ph) then
    kill Pr
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

unacademy

GATE-2005

Suppose n processes, $P_1, \ldots P_n$ 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. $\Sigma_{i=1}^{n}\ s_i < (m+n)$

D. $\Sigma_{i=1}^{n}\ s_i < (m \times n)$

GATE-2005

Two shared resources R_1 and R_2 are used by processes P_1 and P_2. Each process has a certain priority for accessing each resource. Let T_{ij} denote the priority of P_i for accessing R_j. A process P_i can snatch a resource R_h from process P_j if T_{ik} is greater than T_{jk}.
Given the following :

I. $T_{11} > T_{21}$
II. $T_{12} > T_{22}$
III. $T_{11} < T_{21}$
IV. $T_{12} < T_{22}$

Which of the following conditions ensures that P_1 and P_2 can never deadlock?

A. I and IV
B. II and III
C. I and II
D. None of the above

GATE-2006

Consider the following snapshot of a system running n processes. Process i is holding x_i instances of a resource R, $1 \leq i \leq n$. 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 and 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. $x_p + x_q \geq \min_{k \neq p,q} y_k$
C. $\max(x_p, x_q) > 1$
D. $\min(x_p, x_q) > 1$

unacademy

GATE-2007

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

unacademy

GATE-2008

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 *apriori*..

unacademy

GATE-2008

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

unacademy

GATE-2009

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

GATE-2010

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

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

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

In which of the following situations is a deadlock possible?

A. $n = 40, k = 26$
B. $n = 21, k = 12$
C. $n = 20, k = 10$
D. $n = 41, k = 19$

unacademy

GATE-2013

Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., $wait$) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d, and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the V operation (i.e., $signal$) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes?

A. $X : P(a)P(b)P(c)\ Y : P(b)P(c)P(d)\ Z : P(c)P(d)P(a)$
B. $X : P(b)P(a)P(c)\ Y : P(b)P(c)P(d)\ Z : P(a)P(c)P(d)$
C. $X : P(b)P(a)P(c)\ Y : P(c)P(b)P(d)\ Z : P(a)P(c)P(d)$
D. $X : P(a)P(b)P(c)\ Y : P(c)P(b)P(d)\ Z : P(c)P(d)P(a)$

GATE-2014

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.

unacademy

Solution

	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

unacademy

Solution

	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

GATE-2014

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 _____.

GATE-2015

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

A. 1
B. 2
C. 3
D. 4

unacademy

GATE-2015

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

I. Process 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 deccreasing resource numbers
IV. The resources are numbered uniquely. A processes is allowed to request for resources 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

A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for that 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

unacademy

GATE-2018

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 _____ .

unacademy

GATE-2018

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 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

unacademy

Happy Learning.!