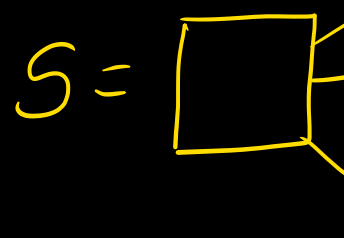
Counting semaphore:



+ve → How many processes can do down successfully

0 → 0 processes can do down successfully
(0 processes are blocked)

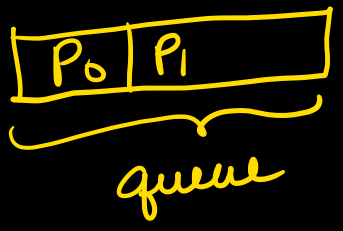
-ve. → How many processes are blocked.

S = 3 → 2 → 1 → 0 → -1 → -2

down(S)
down(S)
down(S)
down(S)

down(S) → Process is blocked. → P_0

one more process is blocked → P_1



Binary semaphore:

```
Struct B_semaphore
{
    enum value (0,1);
    Queue type L;
}
```

L contains all processes that are blocked while performing down operation unsuccessfully.

→ Binary semaphore can only take two value

If counting semaphore

if S = 3

↓

3 processes can enter CS simultaneously

S = 1 / 0

down (S)

down (S) → blocked.

S = 1

down (S)

CS → only one process can enter CS.

up (S)

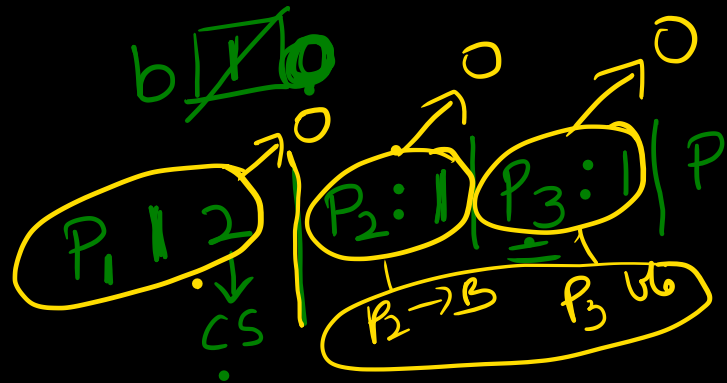
∴ Binary semaphore is also called mutex

mutex

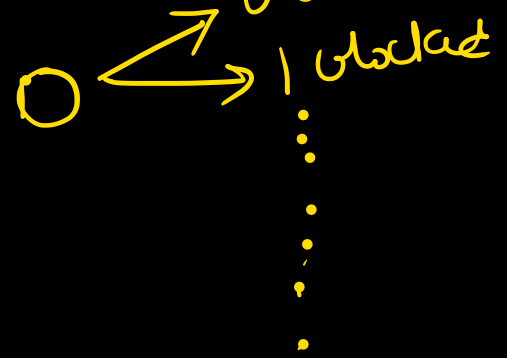
mutual exclusion

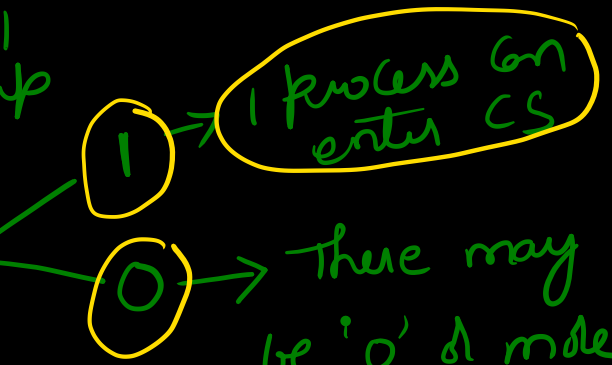
Binary semaphore b=1

1) down(b)
2) CS
3) up(b)



L P2 → unblocked





Semaphores are implemented in OS (Kernel)

∴ mutual exclusion is guaranteed.

Kernel → core → most important part of OS.

Down (Binary semaphore S)

{

if (S.value == 1) → success ✓

{ S.value = 0; ✓

}

else → failure (semaphore value is zero)

{

Put the process in S.L (queue);

sleep() → Process will go to block state

}

}

S = 1 → Down(S) → CS

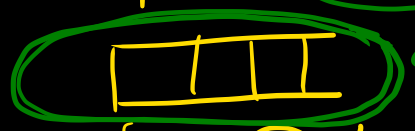
S = 0 → Down() → process will get blocked

→ Kernel (OS) → no preemption

```
up (Binary Semaphore S)
{
    if (S.L is empty) → no process is blocked
        S.value = 1;
    else
    {
        Select a process from S.L;
        wake up ();
    }
}
```

S = 0 or 1

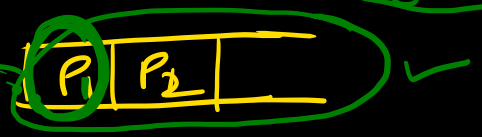
up → S = 1 ✓



Q is empty.

no process is blocked

S = 0 → S will be 1 to 0



up.

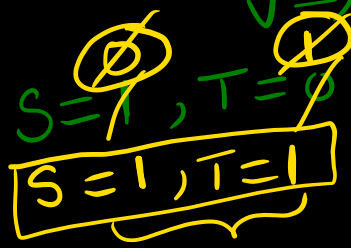
wake up

unblocked.

Binary semaphore & mutex can be used to order the instructions.

order → what is the order in which we can execute instructions.

P → down
V → up
→ initially



P0:
```
while (1)
{
    P(S);
    Pf(1);
    V(T);
}
```

P1:
```
while (1)
{
    P(T);
    Pf(2);
    V(S);
}
```

P(S) → success

P(T) → blocked → unblocked

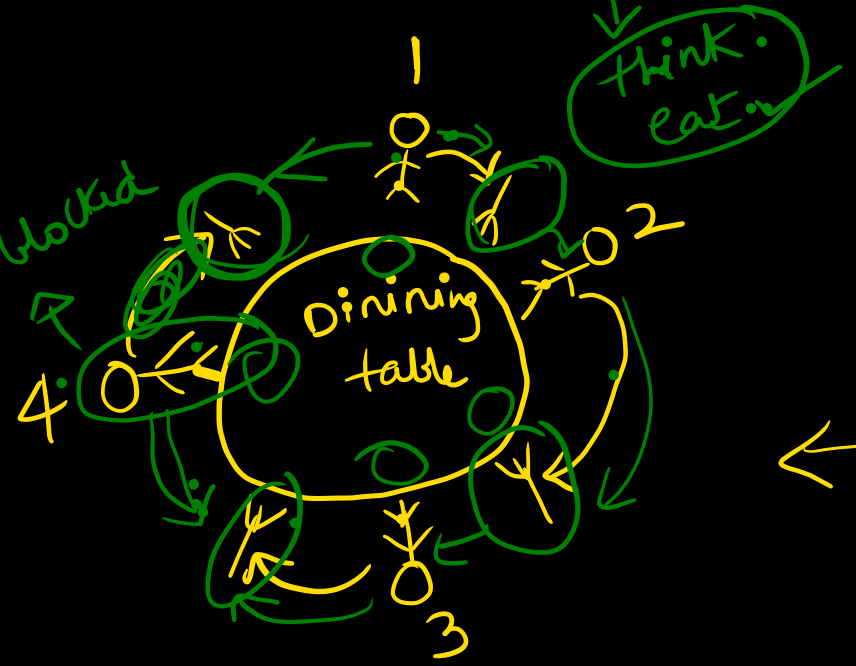
1 2 1 2 1 2 1 . . . .
2 1 2 1 2 1 2 . . . . . .

What will be printed?

1 2 1 2 1 2 1 2

2 1 2 1 2 1 2 1

Dining philosophers problem:



n - people and n forks.
Each one need two forks to eat.

Program:

```
Pi (i = 1,2,3,4)
{
    Take left fork;
    Take right fork;
}
```

→ deadlock.

→ preemption

Solution: At least one person should change order.

```
P4 {
    Take right;
    Take left;
}
```

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go next

Go net

Go net

Go next

5 min Break

Gate questions

Process Management
(PYQs)

Q1. A critical region is [GATE1987 : 2Marks]

Q1. A critical region is [GATE1987 : 2Marks]

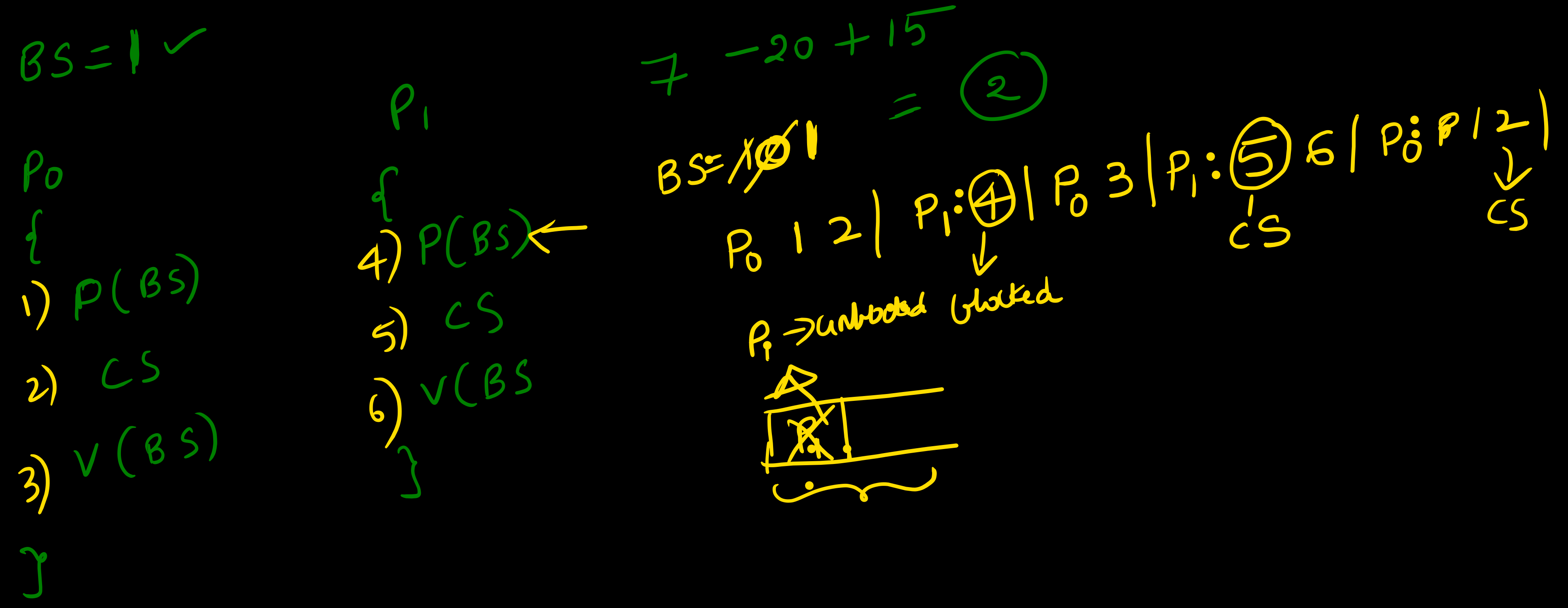
(a) One which is enclosed by a pair of P and V operations on semaphores
(b) A program segment that has not been proved bug-free
(c) A program segment that often causes unexpected system crashes
(d) A program segment where shared resources are accessed

Q2. Semaphore oprations are atomic because they are implemented within the OS.....

[GATE1990 : 2Marks]

no preemption Kernel

Q3. 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: **[GATE1992 : 2Marks]**



Q3. 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: **[GATE1992 : 2Marks]**

(a) 42
(b) 2 ✓
(c) 7
(d) 12

Q4. A critical section is a program segment [GATE 1996 : 1Mark]

Q4. A critical section is a program segment [GATE 1996 : 1Mark]

a) Which should run in a certain specified amount of time
b) where shared resources are accessed
c) which avoids dead lock
d) which must be enclosed by a pair of semaphore operations P and V

Q5. A solution to the Dining Philosophers Problem which avoids deadlock is:

[GATE1996 : 1Mark]

Q5. A solution to the Dining Philosophers Problem which avoids deadlock is:

[GATE1996 : 1Mark]

(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

Q6. Each process Pi, i = 1,2, 3,...., 9 is coded as follows:

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

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

[GATE1997 : 2Mark]

Q6. Each process Pi, i = 1,2, 3,....., 9 is coded as follows:

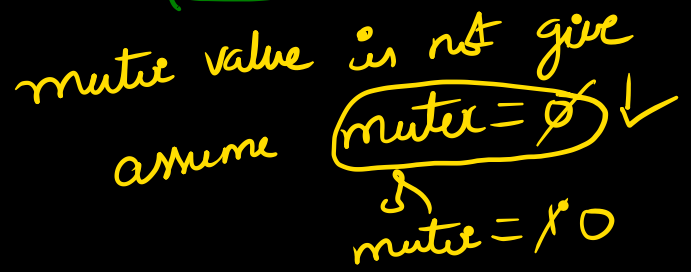
```
repeat
   1) P (mutex)
   2) { critical section }
   3) V (mutex)
forever
```

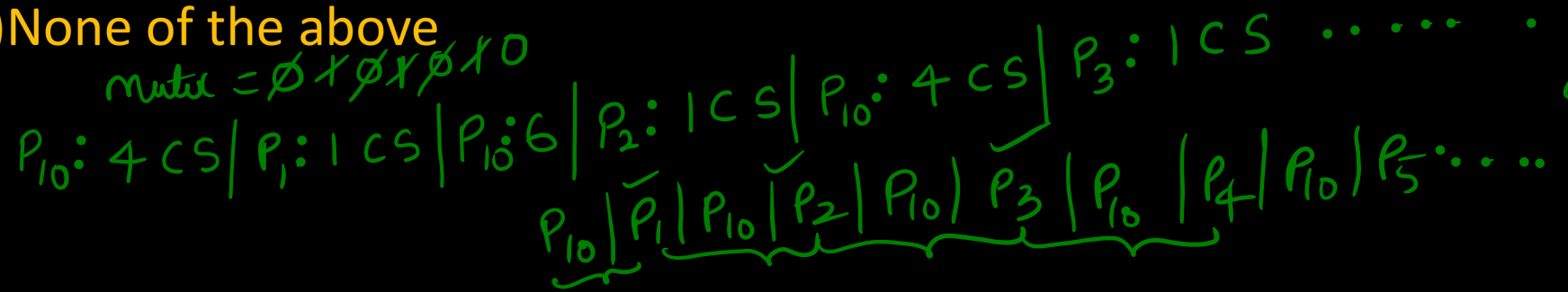
P10:
```
repeat
4) V(mutex)
5) CS
6) V(mutex)
```

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

[GATE1997 : 2Mark]

a)1
b)2
c)3
d)None of the above





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

Q7. When the result of a computation depends on the speed of the processes involved there is said to be [**GATE1998 : 1Mark**]

Q7. When the result of a computation depends on the speed of the processes involved there is said to be **[GATE1998 : 1Mark]**

a) Cycle stealing
b) Race condition
c) A time lock
d) A deadlock

Q8. 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

[GATE 1998 : 1 Mark]

Q8. 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

[GATE 1998 : 1 Mark]

a) 0
b) 8
c) 10
d) 12

Q9. 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]);
```

Q9. 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:

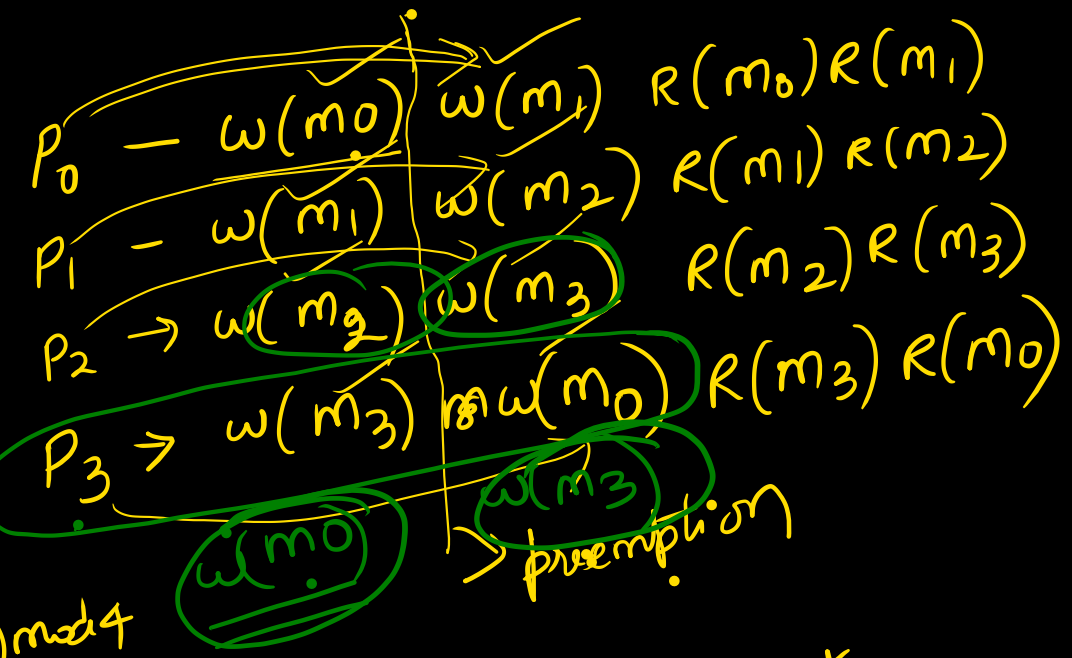
(A) Thrashing

(B) Deadlock

(C) Starvation, but not deadlock

(D) None of the above

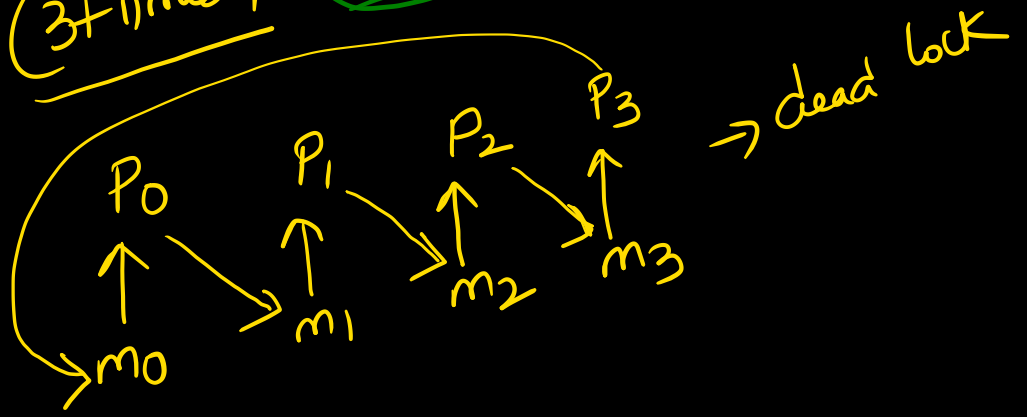
```
wait (m[i]); wait(m[(i+1) mode 4]);

------

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



P0
{
wait(m(0))
wait(m(0+1 mod 4)) — wait(m(1)
⋮
⋮
Release (m[0]
Relase m(1)
}

during while

(3+1)mod4



Q10. Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below. **[GATE 2001 : 1 Mark]**

```
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

Common data for Q11 and Q12

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.

Q11. 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? **[GATE 2003 : 2 Marks]**

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

Process P:	Process Q:
while(1) {	while(1) {
W: P(S)	Y: P(T)
print '0';	print '1';
print '0';	print '1';
X: V(S)	Z: V(T)
}	}

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

output should never contain a substring 01^n0 or 10^n1

option A: S=1 T=1

So both can enter simultaneously

010

101

Lidia

ans in C

not b

Process P:	Process Q:
while(1) {	while(1) {
W: P(S)	Y: P(T)
print '0';	print '1';
print '0';	print '1';
X: V(T)	Z: V(S)
}	}

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

Option B:

S=1 and T=1

So both of them can enter simultaneously.

0 1 0

1 0 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)
}	}

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

H/W

S=1 and T=0

either P will enter & Q will enter

P: 1 2 3 4

if P enters

(00 + 11)

00 11 11 00

$\frac{00}{P}\ \frac{11}{Q}\ \frac{00}{P}\ \frac{00}{P}\ \frac{00}{P}\ \frac{11}{Q}$

$\frac{11}{Q}\ \frac{00}{P}\ \frac{11}{Q}\ \frac{11}{Q}$

01^n0 → n is odd

10^n1 → n is odd

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

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

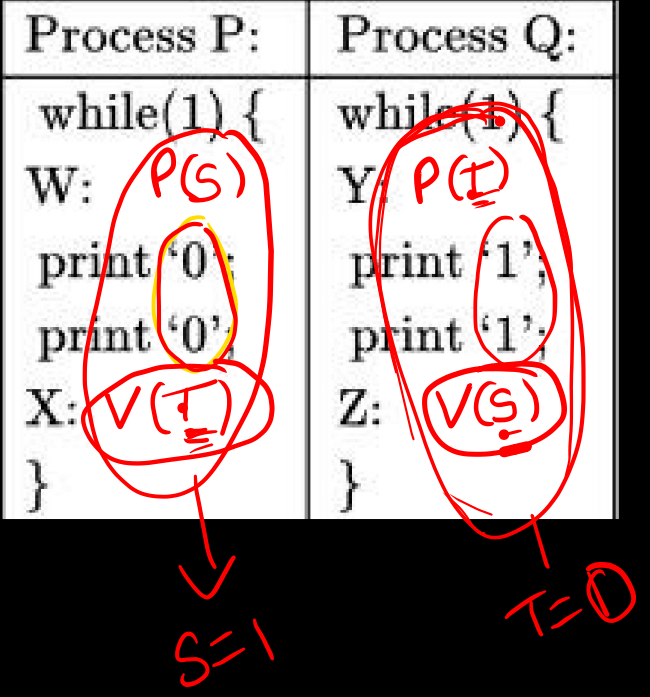
Q12. Which of the following will always lead to an output starting with ‘001100110011’ ?

[GATE 2003 : 2 Marks]

(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

Process P:	Process Q:

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



(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

00110011001100.....

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

(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

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

(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

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

(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

Q13. Consider two processes P1 and P2 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 P1 and P2 is as follows

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

Q13. Consider two processes P1 and P2 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 P1 and P2 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;
$V(S_X)$;	$V(S_Y)$;
$V(S_Y)$;	$V(S_X)$;
}	}

Break 5 min

In order to avoid deadlock, the correct operators at L1, L2, L3 and L4 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) $P(S_X)$, $P(S_Y)$; $P(S_X)$, $P(S_Y)$

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;
$V(S_X)$;	$V(S_Y)$;
$V(S_Y)$;	$V(S_X)$;
}	}

P(Sy)

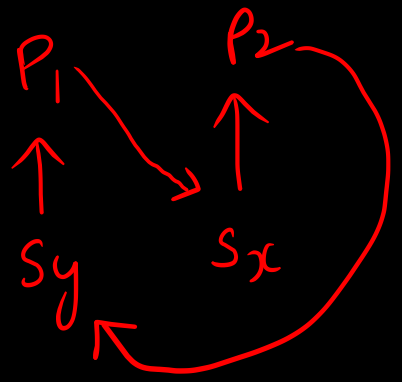
P(Sx)

P(Sx)

P(Sy)

preemption

option A



dead lock.

(A) P(SY), P(SX); P(SX), P(SY)
(B) P(SX), P(SY); P(SY), P(SX)
(C) P(SX), P(SX); P(SY), P(SY)
(D) P(SX), P(SY); P(SX), P(SY)

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

Same problem deadlock.

(A) P(SY), P(SX); P(SX), P(SY)
(B) P(SX), P(SY); P(SY), P(SX)
(C) P(SX), P(SX); P(SY), P(SY)
(D) P(SX), P(SY); P(SX), P(SY)

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

(A) P(SY), P(SX); P(SX), P(SY)
(B) P(SX), P(SY); P(SY), P(SX)
(C) P(SX), P(SX); P(SY), P(SY)
(D) P(SX), P(SY); P(SX), P(SY)

optional:

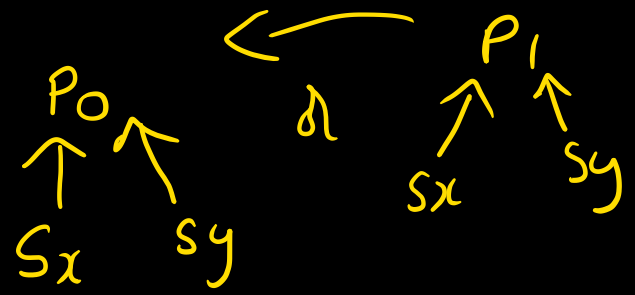
P_1: L_1 L_2 ↓ ↓ Success block | P_2: L_3 L_4 ↓ ↓ Success block

dead lock

P_1:	P_2:
While true do {	While true do {
L_1 : P(Sx) ←	L_3 : P(Sx) ←
L_2 : P(Sy)	L_4 : P(Sy)
X = X + 1;	Y = Y + 1;
Y = Y − 1;	X = Y − 1;
V(S_X);	V(S_Y);
V(S_Y);	V(S_X);
}	}

(A) P(SY), P(SX); P(SX), P(SY)
(B) P(SX), P(SY); P(SY), P(SX)
(C) P(SX), P(SX); P(SY), P(SY)
(D) P(SX), P(SY); P(SX), P(SY)

only one process will get Sx and that process will only get Sy. So no deadlock.



Q14. The semaphore variables full, empty and mutex are initialized to 0, n and 1, respectively. Process P1 repeatedly adds one item at a time to a buffer of size n, and process P2 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.

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

P1

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

P2

The statements K, L, M and N are respectively [GATE 2004 : 2Marks]

(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)

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

K → when to sleep Producer

L → when to wake up Consumer

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

M → Consumer sleep

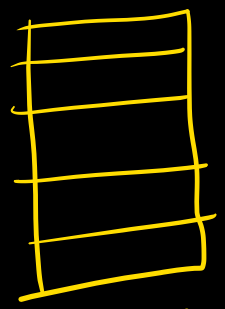
N → Producer wakeup

(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)

full = 0 | empty = N | mutex = 1

P1 → Producer

P2 Consumer.



Buffer.

Initially full slots = 0

empty slots = N

Producer sleep → full == → NO empty slots → empty = 0

∴ P(empty) — (K)

Consumer sleep → NO data → full == 0

P(full) → (M)

Consumer wake up → when full = 1

V(full)

producer → V(empty

Q15. 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 forever
$V(X)$;	$P(X)$;
Compute;	Compute;
$P(X)$;	$V(X)$;

Q15. 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 forever
$V(X)$;	$P(X)$;
Compute;	Compute;
$P(X)$;	$V(X)$;

Consider the following statements about processes P1 and P2:
I: It is possbile for process P1 to starve.
II. It is possible for process P2 to starve

Q15. 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 forever
$V(X)$;	$P(X)$;
Compute;	Compute;
$P(X)$;	$V(X)$;

Consider the following statements about processes P1 and P2:
I: It is possbile 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

$P1$	$P2$
repeat forever	repeat forever
1) $V(X)$;	4) $P(X)$;
2) Compute;	5) Compute;
3) $P(X)$;	6) $V(X)$;

$x = 0/1/0/1$

P_1: 1 2 | P_2: 4 5 | P_1: 3 | P_2: 6 | P_1: 1

3 → B, 6 → UB

Both P_1 and P_2 will enter CS whenever they want

No starvation for both.

No sequence where one process will wait forever

Q16. 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 2005 : 2Marks]**

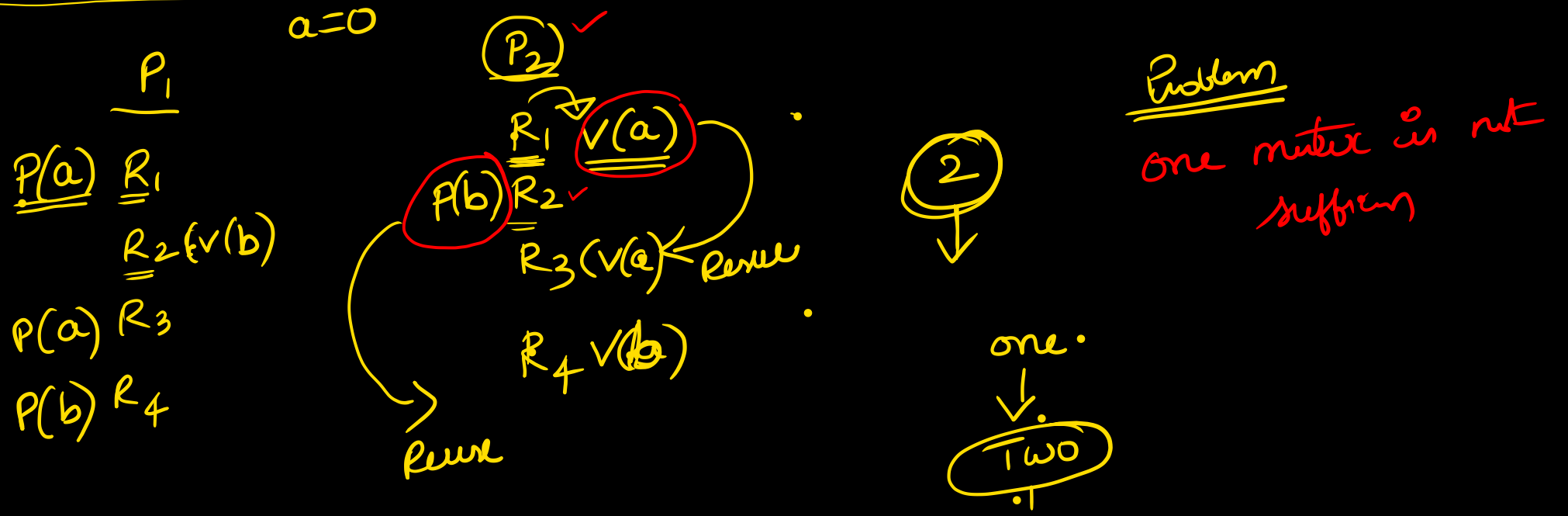
(A) 1
(B) 2
(C) 3
(D) 4

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.



Problem

One mutex is not sufficient

Q17. The atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x n 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;
}
```

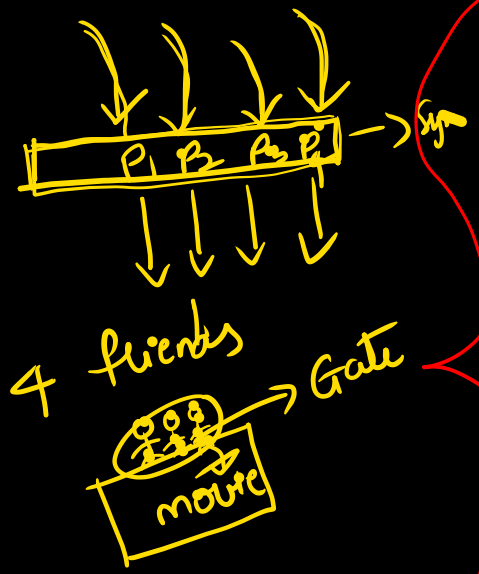
Which one of the following is true? [GATE 2006 : 2Marks]

(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

```
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;
}
```

Common data for Q18 and Q19

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) {
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);
}
```

wait in loop
barrier.
PA = 3
→ any

Vidia please dont chat unnecessarily
Vidia Stop

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.

Q.18 The above implementation of barrier is incorrect. Which one of the following is true?

[GATE 2006 : 2Marks]

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.

Q.18 The above implementation of barrier is incorrect. Which one of the following is true?

[GATE 2006 : 2Marks]

(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.

```
void barrier (void) {
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);
}
```

Q19. Which one of the following rectifies the problem in the implementation?

[GATE 2006 : 2Marks]

Q19. Which one of the following rectifies the problem in the implementation?

[GATE 2006 : 2Marks]

(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

Q20. 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;
    }
}
```

Q20. 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.

Q20. 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 false and (ii) is true

(B) Both (i) and (ii) are false

(C) (i) is true and (ii) is false

(D) Both (i) and (ii) are true

Q21. Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:Here, wants1 and wants2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct? **[GATE2007 : 2Marks]**

```
/*  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 */
```

Q21. Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:Here, wants1 and wants2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct? **[GATE2007 : 2Marks]**

```
/*  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 */
```

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

Q22. 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;

Q22. 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 Pb and Vb the wait and signal operations on binary semaphores are provided. Two binary semaphores Xb and Yb 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 $(s \leq 0) V_b(y_b);$
$V_b(x_b);$

The initial values of Xb and Yb are respectively [GATE 2008 : 2Mark]

(A) 0 and 0
(B) 0 and 1
(C) 1 and 0
(D) 1 and 1

$$
\begin{array}{ll}
 & P_b(x_b); \\
 & s = s - 1; \\
 & \text{if } (s < 0) \\
 & \{ \\
P(s): & V_b(x_b); \\
 & P_b(y_b); \\
 & \} \\
 & \text{else } V_b(x_b);
\end{array}
$$

$$
\begin{array}{ll}
 & P_b(x_b); \\
 & s = s + 1; \\
V(s): & \text{if } (s \leq 0) V_b(y_b); \\
 & V_b(x_b);
\end{array}
$$

Q23. The following program consists of 3 concurrent processes and 3 binary semaphores.The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0. **[GATE2010 : 2Marks]**

Process P0	Process P1	Process P2

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

Q23. The following program consists of 3 concurrent processes and 3 binary semaphores.The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0. **[GATE2010 : 2Marks]**

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

Process P0	Process P1	Process P2

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

Q24. 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? **[GATE2013 : 1Mark]**

Q24. 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? **[GATE2013 : 1Mark]**

(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)

(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)

Q25. 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? **[GATE2013 : 2Marks]**

Q25. 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? **[GATE2013 : 2Marks]**

(A) -2
(B) -1
(C) 1
(D) 2

Q26. A certain computation generates two arrays a and b such that a[i]=f(i) for $0 \le i < n$ and b[i]=g(a[i]) for $0 \le i < n$. 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.

[GATE2013 : 2Marks]

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);
}
```

```
private i;
for (i=0; i< n; i++) {
 a[i] = f(i);
 ExitX(R, S);
}
```

Process X:

```
private i;
for (i=0; i< n; i++) {
  EntryY(R, S);
  b[i] = g(a[i]);
}
```

Process y:

A.

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

```
private i;
for (i=0; i< n; i++) {
 a[i] = f(i);
 ExitX(R, S);
}
```

Process X:

```
private i;
for (i=0; i< n; i++) {
  EntryY(R, S);
  b[i] = g(a[i]);
}
```

Process y:

B.

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

```
private i;
for (i=0; i< n; i++) {
 a[i] = f(i);
 ExitX(R, S);
}
```

Process X:

```
private i;
for (i=0; i< n; i++) {
  EntryY(R, S);
  b[i] = g(a[i]);
}
```

Process y:

C.

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

```
private i;
for (i=0; i< n; i++) {
 a[i] = f(i);
 ExitX(R, S);
}
```

Process X:

```
private i;
for (i=0; i< n; i++) {
  EntryY(R, S);
  b[i] = g(a[i]);
}
```

Process y:

D.

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

Q27. Consider the procedure below for the Producer-Consumer problem which uses semaphores: **[GATE2014 : 2Marks]**

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

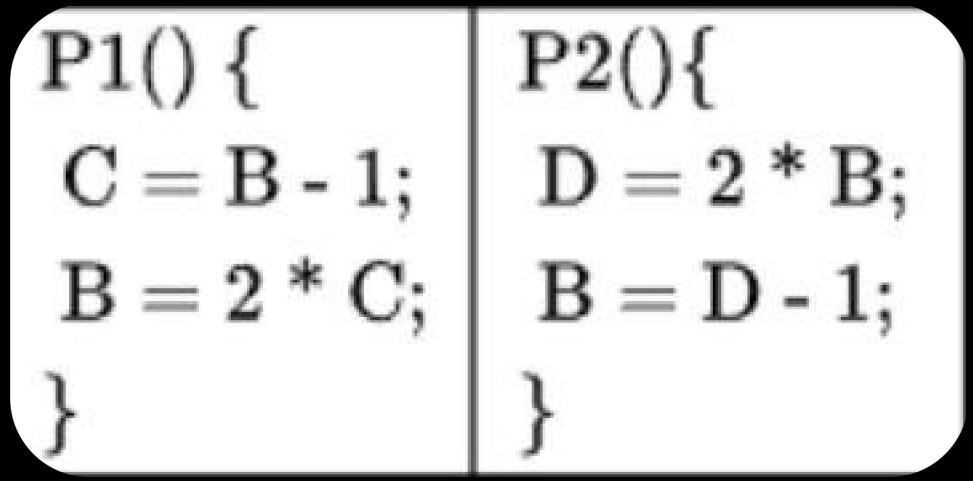
```
void producer()
{
    while(true)
    {
        produce();
        semWait(s);
        addToBuffer();
        semSignal(s);
        semSignal(n);
    }
}
```

```
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.

Q28 .The following two functions P1 and P2 that share a variable B with an initial value of 2 execute concurrently. **[GATE2015 : 1Mark]**



```
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 _____

Q29. Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes. **[GATE2015 : 1Mark]**

Process X	Process Y

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

Q30. Consider the following two-process synchronization solution. **[GATE2016 : 2Marks]**

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

Q30. Consider the following two-process synchronization solution. **[GATE2016 : 2Marks]**

PROCESS 0	Process 1
Entry: loop while (turn == 1); (critical section) Exit: turn = 1;	Entry: loop while (turn == 0); (critical section) 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.

Q.31 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 _____. **[GATE2016 : 2Marks]**

Q.31 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 _____. **[GATE2016 : 2Marks]**

(A) 7
(B) 8
(C) 9
(D) 10

Q32. 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 {	do {
wait (P);	wait (R);
wait (mutex);	wait (mutex);
//Add item to buffer	//consume item from buffer
signal (mutex);	signal (mutex);
signal (Q);	signal (S);
}while (1);	}while (1);

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

[GATE2016 : 2Marks]

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

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

Q33. Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100. **[GATE2019 : 1Mark]**

P_1	P_2	P_3
:	:	:
:	:	:
$D = D + 20$	$D = D - 50$	$D = D + 10$
:	:	:
:	:	:

The process 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 _____.

Q34. Consider the following threads, T1, T2, and T3 executing on a single processor, synchronized using three binary semaphore variables, S1, S2, and S3, 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){	while(true){	while(true){
wait(S_3);	wait(S_1);	wait(S_2);
print("C");	print("B");	print("A");
signal(S_2); }	signal(S_3); }	signal(S_1); }

Q34. Consider the following threads, T1, T2, and T3 executing on a single processor, synchronized using three binary semaphore variables, S1, S2, and S3, 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….?

[GATE2022: 1Mark]

(A) S1 = 1; S2 = 1; S3 = 1
(B) S1 = 1; S2 = 1; S3 = 0
(C) S1 = 1; S2 = 0; S3 = 0
(D) S1 = 0; S2 = 1; S3 = 1

Q35. 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;
}
```

Q35. 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 is TRUE ? **[GATE2009 : 2Marks]**

(A) I only
(B) I and II
(C) II and III
(D) IV only