Gate

Process Management
(PYQs)

Q17. 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.

TSL

Binary semaphore with TSL:

```
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;
}
```

atomic means no preemption

OS Kernel

If S=1 process will be in infinite loop.
No preemption → so only that process will execute forever in the loop.

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

(A) The implementation may not work if context switching is disabled in P → Right

(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 → but not right answer.

A, D both are right but A is more right.

A

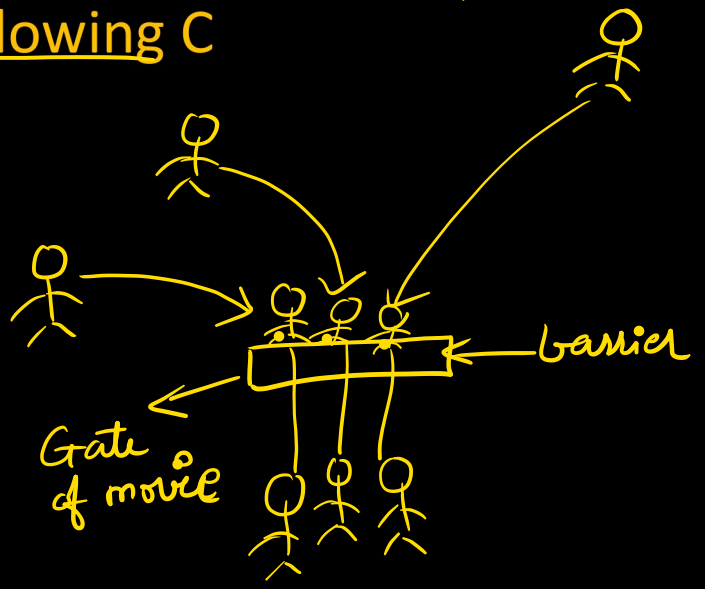
```
void P (binary_semaphore *s) {
    unsigned y;
    unsigned *x = &(s->value);
    do {
        fetch-and-set x, y;
    } while (y);
}

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);
}
```



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?

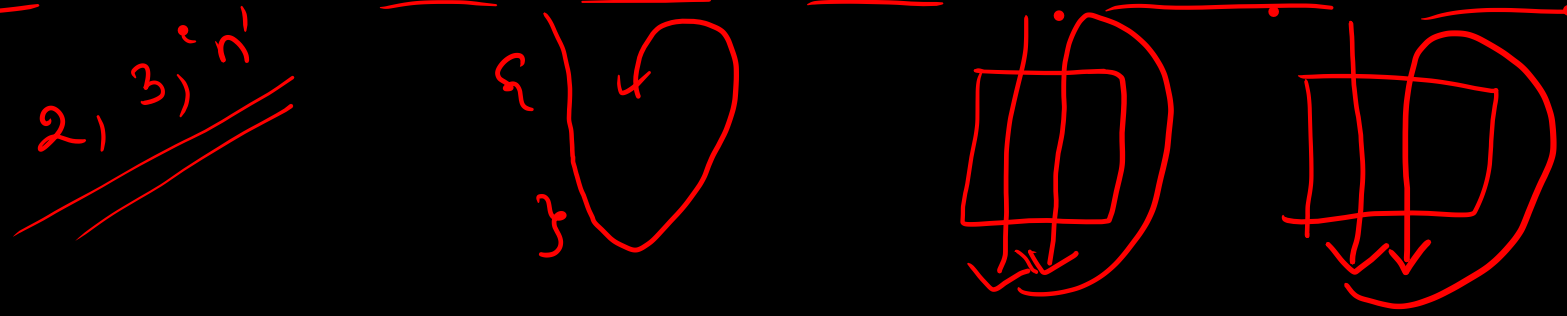
[GATE 2006 : 2Marks]

(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);
}
```

Normal flow: PA = ØX2340 PL = ØX2340

P_1: 1 2 3 4 | P_2: 1 2 3 4 | P_3: 1 2 3 4 5 6 7 11 |

P_1: 4 5 6 7 11 | P_2: 4 5 6 7 8 9 11 |

Barrier in working → 2, 3, 10 ..... 'n'

PA = ØX234 PL = Ø1

P_1: 1 2 3 4 | P_2: 1 2 3 4 | P_3: 1 2 3 4 5 6 7 11 1

2 3 4 | P_1: 4 | P_2: 4 | P_3: 4 | P_1: 4 | P_2: 4

PA = ØX2340.4

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–

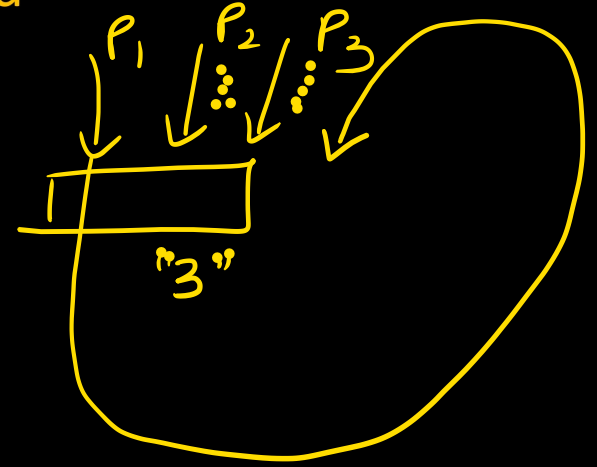
(B) At the ~~beginning of the~~ barrier the first process to enter the barrier waits until process_arrived becomes zero before proceeding to execute P(S).

(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

PA = 0

while (process_arrived != 3) not "3"!



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;
    }
}
```

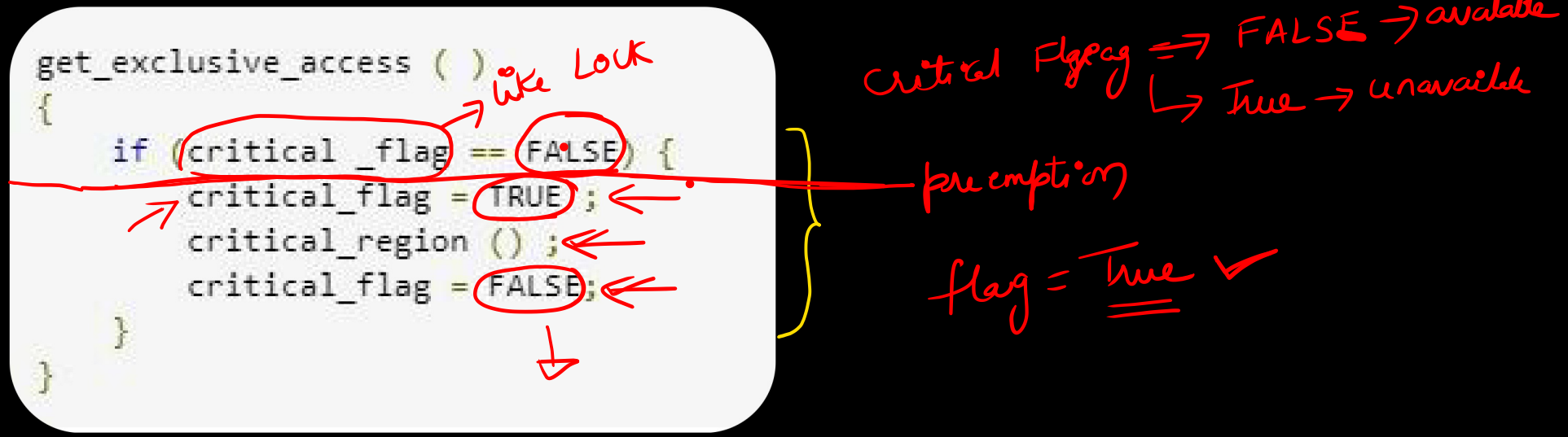
Consider the following statements.

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; → put in queue

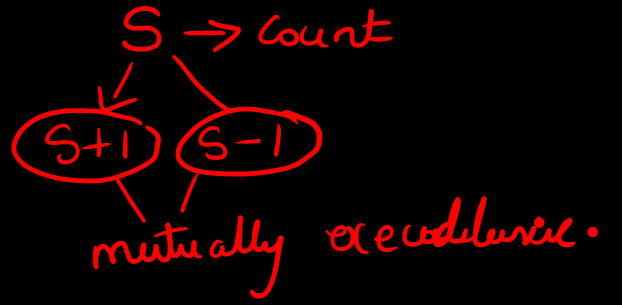
$V(s)$:
$s = s + 1;$ →
If $s \leq 0$ then wake up process waiting on s; → from the queue

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:

using binary semaphores → Counting semaphore is implement.

$P(s)$:
$P_b(x_b);$
$s = s - 1;$
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

$P(s)$:

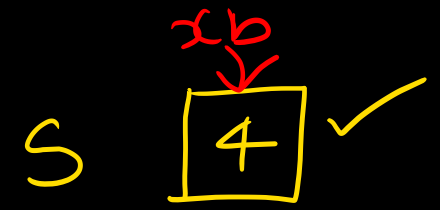
```
P_b(x_b);
s = s - 1;
if (s < 0)
{
    V_b(x_b);
    P_b(y_b);
}
else V_b(x_b);
```

$V(s)$:

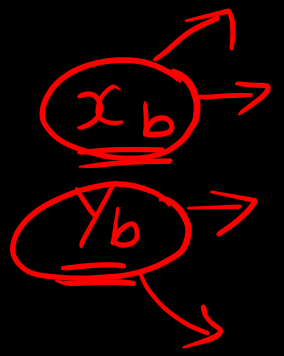
```
P_b(x_b);
s = s + 1;
if (s <= 0) V_b(y_b);
V_b(x_b);
```

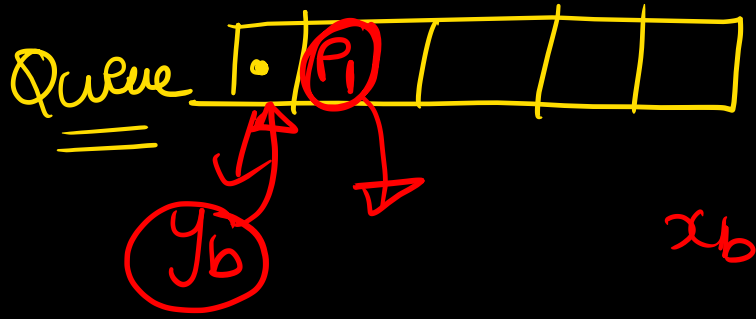
$P_b(x_b)$

x_b is used to provide mutual exclusion to 's'. only one process at a time can change 's'.



S ✓





$x_b = 1$ — no one can access

$y_b = 0$ — every process has to be blocked on $P_b(y_b)$

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

S0 = 1

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]**

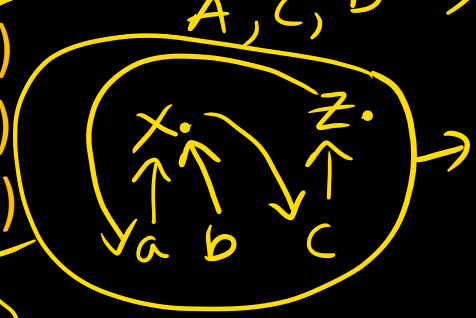
Q24. Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation 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]**

	X	Y	Z
(A)	P(a)P(b)P(c)	P(b)P(c)P(d)	P(c)P(d)P(a)
(B)	P(b)P(a)P(c)	P(b)P(c)P(d)	P(a)P(c)P(d)
(C)	P(b)P(a)P(c)	P(c)P(b)P(d)	P(a)P(c)P(d)
(D)	P(a)P(b)P(c)	P(c)P(b)P(d)	P(c)P(d)P(a)

deadlock

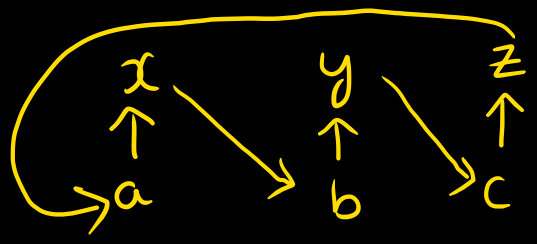
A, C, D → deadlock

dead lock.



Prove that DL is possible
Don't prove DL is not possible.

option D:



(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

$x = 0$

w, x:

$x = x + 1$

y: z:

$x = x - 2$

$S = 2$ ✓

w:

P(S)

1) load x, R0

2) inc R0

3) store R0, x

V(S)

x:

P(S)

1 same

2

3

V(S)

y:

P(S)

4 load x, R1

5 ~~inc R1~~ $R_1 = R_1 - 2$

6 store R1, x

V(S)

z

P(S)

4

5

6

V(S)

w: 1 2 3 | x: 1 2 ✓ | y: 4 5 6 | z: 4 5 6 | x: 3

$R_0 = x$

$x = 1 - 1 - 2$ ②

y: 4 | w: 123 | x: 123 | y: 56 | z: 456

② ✓ min = ④ ✓

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);
}
```

ExitX(R, S); → V(R) P(S) ✓

Semaphores

Process Y:

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

EntryY(R, S); → P(R) V(S)

b[i] → Can be computed only after a[i] is completed

a[0] | b a[1]

b[0]

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.

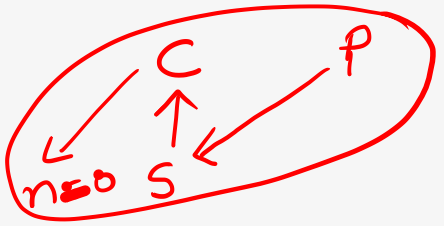
```
ExitX(R, S) {
 V(R);
 P(S);
}
EntryY(R, S) {
 V(S);
 P(R);
}
```

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

1) 2) 3) 4)

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

6 possibility

The number of distinct values that B can possibly take after the execution is 3

2 or 3 or 4

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 */

No progress.
No deadlock
ME X

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]

$S - 20 + 12 = -1$

$S = 7$

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().

empty → no of filled slots
full → no of empty slots

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);

P → full
R → empty
Q → empty
S → full

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

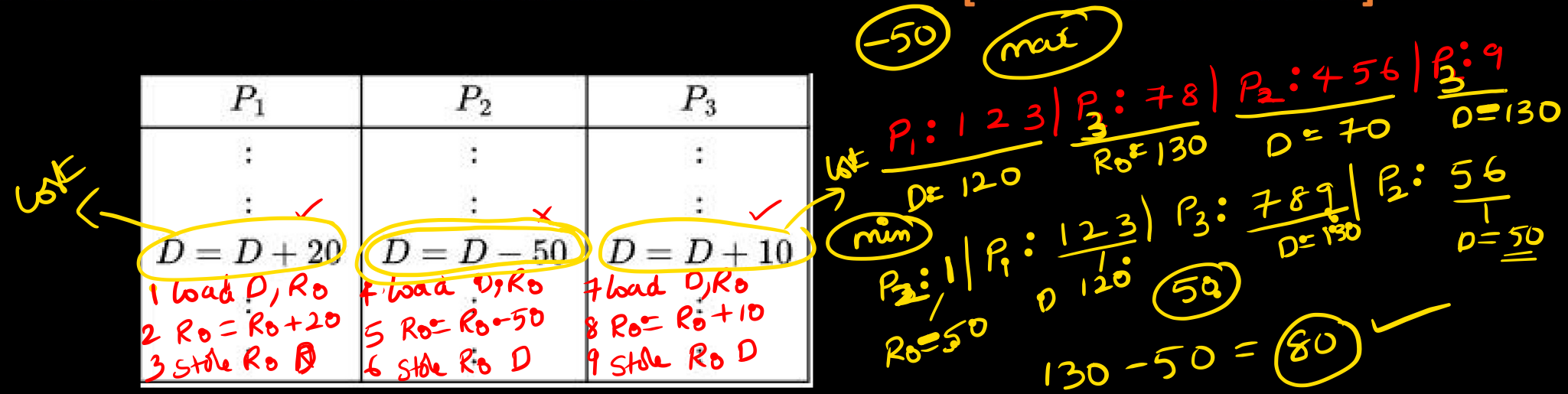
[GATE2016 : 2Marks]

(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
:	:	:
:	:	:
$D = D + 20$	$D = D - 50$	$D = D + 10$
1 load D, R_0	4 load D, R_0	7 load D, R_0
2 $R_0 = R_0 + 20$	5 $R_0 = R_0 - 50$	8 $R_0 = R_0 + 10$
3 store R_0 D	6 store R_0 D	9 store R_0 D



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