Assignment - 3 (Process Synchronization)

1. The following pair of processes share a common variable X.

```
Process A
int Y;
A1: Y = X*2;
A2: X = Y;

Process B
int Z;
B1: Z = X+1;
B2: X = Z;
```

X is set to 5 before either process begins execution. As usual, statements within a process are executed sequentially, but statements in process A may execute in any order with respect to statements in process B.

How many different values of X are possible after both processes finish executing?

a) two

b) three

c) four

d) eight

Answer: c
Explanation: Here are the possible ways in which statements from A and B can be interleaved.
A1 A2 B1 B2: X = 11
A1 B1 A2 B2: X = 6
A1 B1 B2 A2: X = 10
B1 A1 B2 A2: X = 10
B1 A1 A2 B2: X = 6
B1 B2 A1 A2: X = 12

2. The program follows to use a shared binary semaphore T.

```
Process A
```

```
int Y;
A1: Y = X*2;
A2: X = Y;
signal(T);

Process B
int Z;
B1: wait(T);
B2: Z = X+1;
X = Z;
```

T is set to 0 before either process begins execution and, as before, X is set to 5.

Now, how many different values of X are possible after both processes finish executing?

Answer: a

Explanation: The semaphore T ensures that all the statements from A finish execution before B begins. So now there is only one way in which statements from A and B can be interleaved:

A1 A2 B1 B2: X = 11.

3. A counting semaphore was initialized to 10. Then 6 P (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

Answer: (B)

Explanation: Initially we have semaphore value = 10

Now we have to perform 6 p operation means when we perform one p operation it decreases the semaphore values to one.

So after performing 6 p operation we get, semaphore values = 10 – 6 = 4 and now we have to perform 4 v operation means when we perform one V operation it increases the semaphore values to one. So after performing 4 v operation we get, semaphore values = 4 + 4 = 8.

4. 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 .

```
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

Answer (A)
Let us talk about the operation P(). It stores the value of s in x, then it fetches the old value of x, stores it in y and sets x as 1. The while loop of a process will continue forever if some other process doesn't execute V() and sets the value of s as 0. If context switching is disabled in P, the while loop will run forever as no other process will be able to execute V().

5. 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?
(a) I only
(b) I and II
(c) II and III
(d) IV only

Answer: (A)

Explanation: The above solution is a simple test-and-set solution that makes sure that deadlock doesn't occur, but it doesn't use any queue to avoid starvation or to have FIFO order.

6. 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 deadlockfree order of invoking the P operations by the processes? (GATE CS 2013)
   (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)

Answer: (B)

Explanation: Option A can cause deadlock. Imagine a situation process X has acquired a, process Y has acquired b and process Z has acquired c and d. There is circular wait now.

Option C can also cause deadlock. Imagine a situation process X has acquired b, process Y has acquired c and process Z has acquired a. There is circular wait now.

7. 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? (GATE CS 2013)

   (A) -2

   (B) -1

   (C) 1

   (D) 2

Answer: (D)

Explanation:
Background Explanation:

A critical section in which the process may be changing common variables, updating table, writing a file and perform another function. The important problem is that if one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Each process must request permission to enter its critical section. A semaphore is a tool for synchronization and it is used to remove the critical section problem which is that no two processes can run simultaneously together so to remove this two signal operations are used

named as wait and signal which is used to remove the mutual exclusion of the critical section. as an unsigned one of the most important synchronization primitives, because you can build many other Decrementing the semaphore is called acquiring or locking it, incrementing is called releasing or unlocking.

Since initial value of semaphore is 2, two processes can enter critical section at a time- this is bad and we can see why.

Say, X and Y be the processes.X increments x by 1 and Z decrements x by 2. Now, Z stores back and after this X stores back. So, final value of x is 1 and not -1 and two Signal operations make the semaphore value 2 again. So, now W and Z can also execute like this and the value of x can be 2 which is the maximum possible in any order of execution of the processes. (If the semaphore is initialized to 1, processed would execute correctly and we get the final

value of x as -2.)

Option (D) is the correct answer.

8. 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

Answer: (A) (B) (D)

Explanation: First of all, since the value of counter is 5, and if there are two wait() and two signal() for each thread. So, if when these threads will be completed, final counter value can never be 0.

So, option (C) is false.

9. 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 exclusion but not progress
(b) Progress but not mutual exclusion
(c) Neither mutual exclusion nor progress
(d) Both mutual exclusion and progress

Answer: (A)

Explanation:
Mutual Exclusion:
A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing. while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished

executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

Progress Requirement:

If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Solution:

It can be easily observed that the Mutual Exclusion requirement is satisfied by the above solution, P1 can enter critical section only if S1 is not equal to S2, and P2 can enter critical section only if S1 is equal to S2. But here Progress Requirement is not satisfied. Suppose when s1=1 and s2=0 and process p1 is not interested to enter into critical section but p2 want to enter critical section. P2 is not able to enter critical section in this as only when p1 finishes execution, then only p2 can enter (then only s1 = s2 condition be satisfied). Progress will not be satisfied when any process which is not interested to enter into the critical section will not allow other interested process to enter into the critical section.