Tutorial-6

1. 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

2. 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

3. 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

4. Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

/* P1 */	/* P2 */
while (true) {	while (true) {
wants1 = true;	wants2 = true;
while (wants2 == true);	while (wants1==true);
/* Critical	/* Critical
Section */	Section */
wants1=false;	wants2 = false;
}	}
/* Remainder section */	/* Remainder section */

Here, wants1 and wants2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above 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 alternation.
(d) It does not prevent deadlocks but ensures mutual exclusion.

5. 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