Operating Systems Sample Questions

**Concurrency: Semaphores, Condition Variables, and the Producer-Consumer Problem**

1. Describe the behavior of (a) UP and DOWN operations on a semaphore, (b) WAIT and SIGNAL operations on a condition variable.

2. What is the main difference between a binary semaphore and a counting semaphore?

3. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. In the following skeleton of pseudo-code, demonstrate the use of *SEMAPHORES and MUTEXES* to complete the pseudo-code for producer and consumer functions. Your code should have *no race conditions* and *no busy loops*.

You can assume that the following functions are available to you. You shouldn’t need anything more than these functions in your pseudo-code.

- `produce_item()` produces and returns an item
- `insert_item(item)` inserts the item in the common buffer
- `remove_item()` removes and returns an item at the head of the buffer
- `consume_item(item)` consumes the item supplied
- `up(&semaphore)` and `down(&semaphore)` have their usual meanings

```c
#define N 100 /* Number of slots in the buffer */
typedef int semaphore; /* semaphores are a special kind of counter */
semaphore mutex = (initialize this); /* figure out the role of mutex */
semaphore empty = (initialize this); /* figure out the role of empty sem */
semaphore full = (initialize this); /* figure out the role of full sem */

void producer(void)
{
    /* complete this function */
}

void consumer(void)
{
```
4. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. Complete the following skeleton pseudo-code to explain how you can solve the producer-consumer problem using a monitor and condition variables.

```plaintext
procedure Producer
begin
    /* complete this procedure */
end

procedure Consumer
begin
    /* complete this procedure */
end

monitor ProducerConsumer
    condition /* declare the condition variables you need */
    integer /* declare any other variables you need */

    procedure insert(item)
    begin
        /* complete this procedure */
    end

    procedure item *remove()
    begin
        /* complete this procedure */
    end
end monitor
```

5. What is the producer-consumer problem (NOT the solution) and its three synchronization requirements?

6. When would you use a semaphore? When would you use a condition variable?

7. What are the tradeoffs in using semaphores versus monitors with condition variables?
8. How does the **Test-and-Set Lock (TSL)** instruction work? Why can’t we use separate LOAD and STORE instructions instead?

9. Explain how you can implement the UP and DOWN operations on a mutex (binary semaphore) using the TSL instruction.

10. Explain how you can implement the WAIT and SIGNAL operations on condition variable using the TSL instruction.

11. How does the **compare-and-set instruction** work? (b) How can you implement a DOWN operation on a mutex (binary semaphore) using a compare-and-set instruction (such as CMPXCHG in x86)?