

# Operating Systems II : CS3523

## Spring 2019

### Theory Assignment 1: Chapter 6, 7 from the book

Instructor : Dr. Sathya Peri

Author : Puneet Mangla (CS17BTECH11029)

---

#### Solution 1 :

***Busy wait semaphore are used for implementation .***

- (a) The *full* semaphore is initialised as 0 so its minimum is 0 trivially . Semaphore *empty* is initialised to  $n$  and is decreased only by *producer* when it can produce item by calling *wait* on it. Since at max  $n$  items can be accommodated in buffer *producer* can call *wait* only  $n$  times so minimum value of *empty* is also 0.
- (b) The *empty* semaphore is initialised as  $n$  . It can only be signaled by *consumer* only when it consumes one item . For *empty* to be greater than  $n$  , *consumer* has to consume an item when buffer is empty which is not possible hence , its maximum value is  $n$ . Semaphore *full* is initialised to 0 and is only signaled by *producer* when it produces an item . Whenever it produces an item it calls *wait* on *empty* . Since minimum value of *empty* can be 0 . The *full* can only be signaled at max  $n$  times . Thus its max value is also  $n$ .
- (c)  $n - 1 \leq \text{full} + \text{empty} \leq n$   
The sum can be  $n-1$  when producer executes *wait(empty)* and has not executed the *signal(full)* yet.

#### Solution 2 :

Initialise semaphore queue as 1

*Writer*

```
while (true) {  
    wait(queue);  
    wait(rw mutex);  
    signal(queue);  
    . . .  
    /* writing is performed */  
    . . .
```

```

        signal(rw mutex);
    }

```

### *Reader*

```

while (true) {
    wait(queue);
    wait(mutex);
    read count++;
    if (read count == 1)
        wait(rw mutex);
    signal(queue);
    signal(mutex);
    . . .
    /* reading is performed */
    . . .
    wait(mutex);
    read count--;
    if (read count == 0)
        signal(rw mutex);
    signal(mutex);
}

```

### **Solution 3 :**

Compare and compare-and-swap works appropriately for implementing spinlocks . The reason is even if two or more threads enter the conditional statement the compare and swap is executed atomically . That impliest all threads can' t exit the while loop together . There can be at most one thread entering the critical section while other threads waiting in spinlock.

### **Solution 4 :**

The problem with such implementation is that it doesn' t ensure that one thread is not blocking while waiting for the semaphore. If semaphore value is 1 and two threads came and enter the if statement then wait will be executed for both but for the first thread it decreases the semaphore value to 0 and when second thread calls wait it blocks there.