#1 Reader Writer (Writers priority implementation)

```
int writers; // # writer threads that want to enter the critical section (some or all of these may be blocked)
int writing; // Number of threads that are actually writing inside the C.S. (can only be zero or one - can you see why?)
int reading; // Number of threads that are reading inside the C.S.
int readers; // Number of threads that are or want to read

// if writing !=0 then reading must be zero (and vice versa)
```

reader	writer

<pre>reader() {
    lock(&m)
    readers ++
    while (______________________)
        cond_wait(&r_cv, &m)
 	
    Do we need to wait for
    both 'writers' and 'writing'?
 	
    reading++
    unlock(&m)
 	
  // perform reading here
 	
    lock(&m)
    reading--
    readers--
    wake up who here? (and how many)
 	
    unlock(&m)
    return result
}</pre>
	<pre>writer(){
	    lock(&m)
	    writers++
	    while (______________________)
	        cond_wait(&w_cv, &m)
	
	    writing++
	    unlock(&m)
	
	    // perform writing here
	
	    lock(&m)
	    writing--
	    writers--
	    wake up who here? (and how many)
	
	    unlock(&m)
	}</pre>

DEADLOCK

#2 Deadlock Definition:

#3 Coffman Conditions

Necessary? Y/N
Sufficient? Y/N

1

2

3

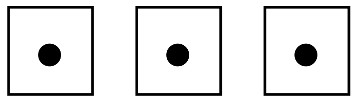
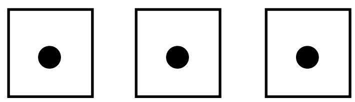
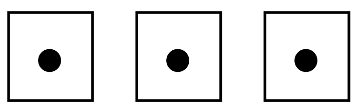
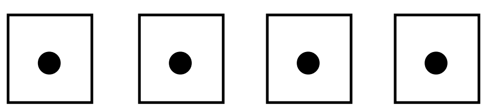
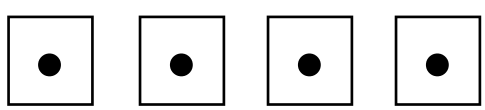
4

#4 Resource Allocation Graphs

Figure 1. Deadlock do not confuse it with dreadlocks.

Assume processes acquire locks in the order specified and release resources only when finished. Create a *resource allocation graph* to determine if and when there is deadlock.

When a process waits for a resource it will acquire an exclusive lock on resource as soon as no other process has an exclusive lock. Assume locks are fair (earliest waiting process obtains the lock).

Q1 Process 1 ("P1") requests (and obtains) Resource A and then Resource B Process 2 requests C and then B. Deadlock for P1? P2?	
Q2 P1 requests (and obtains?) A P2 requests (and obtains?) B P3 requests (and obtains?) C P2 requests (and obtains?) C P3 requests (and obtains?) A P1 requests (and obtains?) C	
Q3 P1 requests A then B P2 requests C then B P3 requests B P4 requests C then B Deadlock for P1? P2? P3? P4?	
Q4 P1 requests A then B P2 requests C, D then B P4 requests D P3 requests B P1 requests C Deadlock for P1? P2? P3? P4?	
Q5 P1 requests A and B P2 requests C and D then B P4 requests D P3 requests B P1 *releases* B (thus P2 acquires B) P1 requests C Deadlock for P1? P2? P3? P4?	

#5 What is the Banker's Algorithm?

#6 Deadlock Avoidance

#7 Linux/Windows strategy for deadlock avoidance?

#8 Acquiring resources in same rank