

#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() {
    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
}

```

```

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)
}

```

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).

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

#5 What is the Banker's Algorithm?

#6 Deadlock Avoidance

#7 Linux/Windows strategy for deadlock avoidance?

#8 Acquiring resources in same rank