

Challenge 1: "Make a barrier using only one mutex lock() and unlock()"

"Impossible Line 2 is a Critical Section, if a thread has locked the mutex..."

But here is an awful solution. (Why is this a 'poor' solution?)

```
01 void barrier() {
02     count ++
03     while( count != N) _____
04
05 }
```

2. When is disabling interrupts a solution to the Critical Section Problem?

```
pthread_mutex_lock() => { disable interrupts on the CPU }
pthread_mutex_unlock => { enable interrupts on the CPU }
```

Are there other limitations to this approach?

3. Challenge II: Create a barrier using each of the following lines once.
All 5 threads must call barrier before they all continue.

```
int remain =5;
void barrier() {
/* Rearrange the following */
    sem_wait(&s);
    sem_post(&s)
    remain --;
    pthread_mutex_lock(&m);
    pthread_mutex_unlock(&m);
    if(remain)
}
```

4. Challenge III. What is the largest value printed by the following?

```
pthread_cond_t cv = P_COND_INITIALIZER;
pthread_mutex_t m = P_MUTEX_INITIALIZER;
int fireworks=0;
pthread_t tids[5];
int main(argc,argv) {
    for(int i=0;i<5;i++) pthread_create( tids+i , NULL, firework, NULL);
    fireworks = 1;
    p_cond_signal(&cv);
    _____; // wait for all threads to finish
    return 0;
}
void* firework(void*param) {
    p_mutex_lock(&m);
    while(fireworks ==0) {p_cond_wait(&cv, &m); }
    p_cond_broadcast(&cv);
    fireworks ++;
    printf("Oooh ahh %d\n", fireworks);
    fireworks --;
    p_mutex_unlock(&m);
    return NULL;
}
```

5. Deadlock: " _____ "

Use two mutex locks and two threads to create an example of deadlock

Thread1:	Thread 2:
----------	-----------

Use three counting semaphores and three threads to deadlock 3 threads

thread #1:	thread #2:	thread #3:

Must deadlock involve threads? What about single-threaded processes?

6. The Reader Writer problem

A common problem in many different system applications

read_database(table, query) {...}	update_row(table, id, value) {...}
-----------------------------------	------------------------------------

cache_lookup(id) {...}	cache_modify(id, value) {...}
------------------------	-------------------------------

7. ReaderWriter locks are useful primitives & included in the pthread library!

01 pthread_rwlock_t lock;	01 cache_lookup(id) {
02 p_rwlock_init	02 p...rdlock(...)
03 p_rwlock_wrlock	03 read from resource
04 p_rwlock_rdlock	04 p...unlock(...)
05 p_rwlock_unlock	05 return result
	06 }

The **pthread_rwlock_?_lock()** function acquires a ?_____ lock on lock provided that lock is not presently held for ?_____ and no ?_____ threads are presently blocked on the lock. If the read lock cannot be immediately acquired, the calling thread blocks until it can acquire the lock.

CS241: Have to skills and the ability to build these! Along the way, also learn to reason about, develop and fix multi-threaded code

8. ~~ Welcome to the *Reader Writer* Game Show! ~~

Contestant #1

p_mutex_t *readlock,*writelock readlock=malloc(sizeof p_mutex_t) writelock=malloc(sizeof p_mutex_t) p_m_init(readlock,NULL) P_m_init(writelock,NULL) read() { lock(readlock) // do read unlock(readlock) }	write() { lock(writelock) lock(readlock) // do writing unlock(readlock) unlock(writelock) }
---	---

Contestant #2

bool reading=0, writing=0

read() { while(writing) {} reading = true // do reading here reading = false }	write() { while(reading writing) { writing = true // do writing here writing = false }
---	---

Contestant #3

read(){ lock(&m) while (writing) cond_wait(cv,m) reading++ /* Read here! */ reading-- cond_signal(cv) unlock(&m)	write(){ lock(&m) while (reading writing) cond_wait(cv,m) writing++ /* Write here! */ writing--; cond_signal(cv) unlock(&m)
--	--

Challenge: Sketch a better solution.