Lock Guide

What is the purpose of the spinlock used by a lock?

- The lock structure has a field which indicates if the lock is available or not.

```
struct Lock
{
   // … other lock fields …
   bool held;  // Note:  there are other solutions
   // … other lock fields …
};

void LockInit( Lock * lock )
{
   // … lock initialization code …
   lock->held = false;  // no one owns the lock yet
   // … other lock initialization code …
}
```

- To take the lock, a thread must test-and-set the field which holds the lock availability

```
Acquire( Lock * lock )
{
   // … lock code here …
   while ( lock->held ) { … }   // this is the TEST
   lock->held = true;    // this is the SET
   // … lock code here …
}
```

- As written above, we would have a race condition because the test-and-set operation requires mutual exclusion.
- We can use a spinlock, instead of assembly, to protect the critical section.

```
Acquire( Lock * lock )
{
   // … non-critical section lock code here …
   Acquire( lock->spinlock )
      // … critical section lock code here …
      while ( lock->held ) { … }   // this is the TEST
      lock->held = true;    // this is the SET
      // … critical section lock code here …
   Release( lock->spinlock )
}
```

- The spinlock is only held for a brief amount of time --- only for the test-and-set.
- **DO NOT** block a thread (put it to sleep) while it owns the spinlock!

Why a loop?

- Suppose there was no test-and-set loop.

```
Acquire( Lock * lock )
{
   // … non-critical section lock code here …
   Acquire( lock->spinlock )
      // … critical section lock code here …
      if ( lock->held ) { … block … }   // this is the TEST
      lock->held = true;    // this is the SET
      // … critical section lock code here …
   Release( lock->spinlock )
}
```

- Given three threads, A, B and C and a lock L, which is initially available.

A	B	C
Acquire(L) // succeeds, **A owns L**		
Context Switch		

	Acquire(L)
	if (lock->held)
	// blocks, L is unavailable	

Context Switch		

Release(L)
// **L is available**
// B is unblocked
// B placed in ready queue		

Context Switch		
		Acquire(L) // succeeds, **C owns L**
Context Switch		
	Exits lock->held branch lock->held = true // succeeds **// B ALSO owns L!!**	

- When B is unblocked and continues execution there is a possibility that the lock was taken by another thread that was scheduled to run before B. Before B can take the lock after being unblocked it needs to ensure that lock is still available. Hence, the test is in a loop.

```
Acquire( Lock * lock )
{
    // … non-critical section lock code here …
    Acquire( lock->spinlock )
        // … critical section lock code here …
        while ( lock->held ) { … block … }   // this is the TEST
        lock->held = true;    // this is the SET
        // … critical section lock code here …
    Release( lock->spinlock )
}
```

Why must you release the spinlock prior to calling wchan_sleep?

- A thread that calls wchan_sleep will block.
- If that thread owns the spinlock when it blocks, then no other threads may acquire that spinlock. Those threads will end up spinning --- instead of blocking.

```
Acquire( Lock * lock )
{
    // … non-critical section lock code here …
    Acquire( lock->spinlock )  // other threads attempting to Acquire spin here
    // until lock blocked thread releases spinlock!
        // … critical section lock code here …
        while ( lock->held )
        {
            … code here …
            wchan_sleep( lock->wchan )  // spinlock not released before calling sleep
            … code here …
        }
        lock->held = true;
        // … critical section lock code here …
    Release( lock->spinlock )
}
```

- Must release spinlock prior to blocking to ensure that threads attempting to acquire the Lock do not spin on the spinlock protecting Acquires critical section

```
Acquire( Lock * lock )
{
    // … non-critical section lock code here …
    Acquire( lock->spinlock )  // threads will only spin waiting for critical section
    // and never for a blocked thread to release spinlock
        // … critical section lock code here …
        while ( lock->held )
        {
            … code here …
            Release( lock->spinlock )
            wchan_sleep( lock->wchan )  // spinlock not owned by blocked thread
            … code here …
        }
        lock->held = true;
        // … critical section lock code here …
    Release( lock->spinlock )
}
```

Why must you lock the wait channel prior to releasing the spinlock?

- Suppose we locked the wait channel after releasing the spinlock.

```
Acquire( Lock * lock )
{
    // … non-critical section lock code here …
    Acquire( lock->spinlock )
        // … critical section lock code here …
        while ( lock->held )
        {
            … code here …
            Release( lock->spinlock )    // release spinlock
            wchan_lock( lock->wchan )  // lock wait channel
            wchan_sleep( lock->wchan )  // block
            … code here …
        }
        lock->held = true;
        // … critical section lock code here …
    Release( lock->spinlock )
}
```

- Two threads, A and B and one lock L.

A	**B**
Acquire(L) // succeeds, **A owns L**	
Context Switch	

	Acquire(L)
	  // ...
	  while(L->held)
	  {
	    // ...
	    Release(L->spinlock)

Context Switch	
Release(L) // **L is available**	
Context Switch	
	wchan_lock(L->wchan) wchan_sleep(L->wchan) **// B is asleep on L's wait channel** **// BUT L is AVAILABLE** **// Who wakes B?**

- You must acquire the wait channel lock prior to releasing the spinlock to prevent this.

Exercises

1. What happens if a thread tries to acquire a lock it already owns?
2. What happens if a thread tries to release a lock it does not own?
3. Suppose N threads are blocked waiting to acquire lock L. Suppose that Release calls wchan_wakeall. What happens to the N blocked threads when Release is called?
    a. Which of the N threads acquires L?
    b. What happens to threads that do not acquire L?