Implementing Mutexes & Condition Variables

CS520
Dept. of Computer Science
Univ. of New Hampshire
Mutex & Condition Variable Primitives

```c
int thread_mutex_init (thread_mutex_t *mutex);
int thread_mutex_lock (thread_mutex_t *mutex);
int thread_mutex_unlock (thread_mutex_t *mutex);

int thread_cond_init (thread_cond_t *cond);
int thread_cond_wait (thread_cond_t *cond, thread_mutex_t *mutex);
int thread_cond_signal (thread_cond_t *cond);
```
Mutex lock / Unlock

lock object must contain:

- Flag — is lock locked?
- If locked, which thread owns it?
- If locked, queue of threads waiting on lock
Mutex Lock

if lock is not locked
  set FLAG to LOCKED
  set owner to current thread's ID
else
  if the owner is the current thread
    return FAILURE
  else
    remove current thread's TCB from ready list
    put it on the end of the lock's queue
    call asm-yield
    set FLAG to LOCKED
    set owner to current thread's ID
return success
Mutex Unlock

See if you can write the pseudo code yourself.
Condition Variable Wait/Signal

Condition variable must contain:

queue of threads waiting

For each thread in queue, the mutex it held when it called wait
Wait

- Remove current thread's TCB from ready list
- Place TCB on end of c.u.'s wait queue
- Record mutex in queue entry
- Unlock mutex
- Call csm_yield
- Lock mutex

* Perhaps add field to TCB

** Return failure if thread does not have mutex locked
See if you can write pseudo code yourself.
This discussion assumed there was only one processor.

If our threads were running on multiple processors, then we would use semaphores to do low-level lock or lock and condition variable objects before we worked upon them.