Implementing Mutexes & Condition Variables

CS520
Dept. of Computer Science
Univ. of New Hampshire

Mutex & Condition Ver. 66 Primitives

```
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
```

5/21/13 This should be done in unlock!

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.v.'s wait queue

record mutex in queue entry *

unlock mutex **

call csm_yield

lock mutex

Do this in signal?

* perhaps add field to TCB

** return failure if thread does not have mutex locked

Sig. 1

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 cmpxchg to do low-level lock on lock and condition variable objects before we worked upon them.