

# Implementing Mutexes & Condition Variables

CS520

Dept. of Computer Science  
Univ. of New Hampshire

## Mutex & Condition Variable 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

## 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<sup>\*</sup>

unlock mutex<sup>\*\*</sup>

call csu\_yield

lock mutex

\* perhaps add field to TCB

\*\* return failure if thread does not have  
mutex locked

Signal

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 `mutex` to do low-level lock or lock and condition variable objects before we worked upon them.