CMSC 23000	**Operating Systems**	**Homework 2**
Autumn 2006		**Due October 24**

The C functions `setjmp` and `longjmp` provide a portable way to save and restore machine state. Assume that you are given the following C types to represent threads:

```
#include <setjmp.h>

typedef enum { READY, BLOCKED } status_t;

typedef struct {
    jmp_buf     state;
    status_t    status;
} *tid_t;

#define NO_TID ((tid_t *)0)
```

and the following interface to support queues of threads:

```
typedef ... queue_t;

void queue_init (queue_t *);
void enqueue (queue_t *, tid_t *);
tid_t *dequeue (queue_t *);

extern queue_t *ReadyQ;
```

The `dequeue` function returns a nil pointer when the queue is empty. You may assume that the ready queue (`ReadyQ`) is never empty. Also assume that you have the following atomic *compare and exchange* operation:

```
bool cmpxchg (void **word, void **key, void *val)
{
    if (*word == *key) {
        *word = val;
        return true;
    }
    else {
        *key = *word;
        return false;
    }
}
```

There are two parts to the assignment:

1. Implement *reentrant* mutex locks with the following interface:

```
typedef ... mutex_t;
void mutex_init (mutex_t *mu);
void mutex_lock (mutex_t *mu);
void mutex_unlock (mutex_t *mu);
```

Your solution should include the representation of locks as well as the implementation of the operations.

2. Add condition variables to your lock implementation with the following interface:

```
typedef ... cond_t;
void cond_init (cond_t *cv);
void cond_wait (mutex_t *mu, cond_t *cv);
void cond_signal (cond_t *cv);
```

Your solution should include the representation of condition variables as well as the implementation of the operations.