CMSC 23000	**Operating Systems**	**Homework 1**
Winter 2006		**Due January 25**

The Intel IA32 provides an atomic *compare and swap* instruction with the following semantics:

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

Note: in the actual hardware instruction, the in-out parameter `key` is held in the `%eax` register. In the questions below, you should use the `cmpxchg` function. Furthermore, you may assume the existence of a function

```
thread_id_t get_tid();
```

and a constant `NO_TID` of type **`thread_id_t`** that is distinct from any thread's ID.

1. Using the `cmpxchg` function, implement a spinlock. Your implementation should include the following definitions:

   ```
   typedef ... SpinLock_t;
   void sl_init (SpinLock_t *);
   void sl_lock (SpinLock_t *);
   void sl_unlock (SpinLock_t *);
   ```

2. Using `cmpxchg`, implement a mutex lock that satisfies the following properties:

   - *mutual exclusion* — no two threads can hold the lock at the same time.
   - *progress* — if one or more threads is attempting to acquire the lock, then one of them will do so.
   - *fairness* — is a thread is waiting on the lock, then there is a bound on the number of times that other threads are allowed to acquire the lock before it does so.

   Justify that your implementation satisfies these properties. For this question, you may assume the existence of a FIFO queue abstraction and the `block` and `wakeup` operations used in the text.

3. For extra credit, change the implementation of the lock to support *reentrancy*. *I.e.,* allow nested locking by the same thread.