4.8. If a program uses more than one mutex, and the mutexes can be acquired in different orders, the program can **deadlock**. That is, threads may block forever waiting to acquire one of the mutexes. As an example, suppose that a program has two shared data structures–for example, two arrays or two linked lists–each of which has an associated mutex. Further suppose that each data structure can be accessed (read or modified) after acquiring the data structure's associated mutex.

a. Suppose the program is run with two threads. Further suppose that the following sequence of events occurs:

Time	Thread 0	Thread 1
0	`pthread_mutex_lock(&mut0)`	`pthread_mutex_lock(&mut1)`
1	`pthread_mutex_lock(&mut1)`	`pthread_mutex_lock(&mut0)`

What happens?

b. Would this be a problem if the program used busy-waiting (with two flag variables) instead of mutexes?

c. Would this be a problem if the program used semaphores instead of mutexes?

4.9. Some implementations of Pthreads define barriers. The function

```
int pthread_barrier_init(
      pthread_barrier_t*             barrier_p  /* out */,
      const pthread_barrierattr_t*   attr_p     /* in  */,
      unsigned                       count      /* in  */);
```

initializes a barrier object, `barrier_p`. As usual, we'll ignore the second argument and just pass in `NULL`. The last argument indicates the number of threads that must reach the barrier before they can continue. The barrier itself is a call to the function

```
int pthread_barrier_wait(
      pthread_barrier_t*  barrier_p  /* in/out */);
```

As with most other Pthreads objects, there is a destroy function

```
int pthread_barrier_destroy(
      pthread_barrier_t*  barrier_p  /* in/out */);
```

Modify one of the barrier programs from the book's website so that it uses a Pthreads barrier. Find a system with a Pthreads implementation that includes barrier and run your program with various numbers of threads. How does its performance compare to the other implementations?

4.10. Modify one of the programs you wrote in the Programming Assignments that follow so that it uses the scheme outlined in Section 4.8 to time itself. In order to get the time that has elapsed since some point in the past, you can use the macro `GET_TIME` defined in the header file `timer.h` on the book's website.