

Homework 1 (due 2/3/2009 at 12:05pm)
via email to ada@cc or hardcopy in class

1. Regular mutexes block when they are already locked and a thread A tries to lock them, even if the current holder is the same thread. In contrast, a *recursive mutex* is one that can be locked multiple times by the same thread and needs to be unlocked the same number of times before other threads can enter the critical section. Using the primitives shown in class (e.g., regular mutexes and condition variables), or any equivalent set you are familiar with, implement a recursive mutex. That is, with pseudo code (or pthreads, if you want to) show the implementation of:

- a type `RecMutex` (i.e., what does this data structure look like);
- three routines `rec_mutex_lock(rec_m)`, `rec_mutex_unlock(rec_m)`, and `rec_mutex_init(rec_m)` that operate on entities of type `RecMutex` and implement the entry and exit from the critical section protected by a recursive mutex, as well as the initialization of the recursive mutex data.

You may assume the availability of a function `current_thread()`, which returns the unique identifier of the currently executing thread.

2. Consider the Stein-Shah paper on the implementation of the Solaris user-level threads library. Answer the following question:

When user-level threads and their masks are not visible to the kernel, how are signals delivered to the appropriate user-level thread?