National University of Computer and Emerging Sciences
Islamabad.
Operating System, Spring 09
Homework 5

These are some of the hardest synchronization problems that have appeared on 162 midterms. If you can do these, you're probably in good shape.

For each of the following problems …

a. Identify the correctness constraints of the problem
b. Specify the conditions that each method must wait for
c. Write down the shared state that you will use to check these conditions
d. Implement a solution using a single **Mesa-style monitor** (only one lock)

1. Building H_2O
   Create a monitor with methods `Hydrogen()` and `Oxygen()`, which wait until a water molecule can be formed and then return. Don't worry about explicitly creating the water molecule; just wait until two hydrogen threads and one oxygen thread can be grouped together. For example, if two threads call `Hydrogen`, and then a third thread calls `Oxygen`, the third thread should wake up the first two threads and they should then all return.
2. FIFO Semaphores
   The Nachos semaphore implementation has a problem: threads that call `P()` do not necessarily return in the same order. Implement semaphores that do not have this problem. Assume that locks and condition variables are already FIFO (Nachos condition variables are FIFO, as are the lock implementations given in lecture). See the synchronization hand-out for a non-FIFO implementation of semaphores. Hint: no thread in `P` should have to wait on a condition variable more than once, since otherwise FIFO order is lost. Another hint: the fundamental problem with the existing implementation of `P` is that it doesn't check to see if any other threads arrived first.
3. River Crossing
   A particular river crossing is shared by both Linux hackers and Microsoft employees. A boat is used to cross the river, but it only seats **four** people, and must always carry a full load. In order to guarantee the safety of the hackers, you cannot put three employees and one hacker in the same boat; similarly, you cannot put three hackers in the same boat as an employee. To further complicate matters, there is room to board only one boat at a time; a boat must be taken across the river in order to start boarding the next boat. All other combination are safe. Two procedures are needed, HackerArrives and EmployeeArrives, called by a hacker or employee when he/she arrives at the river bank. The procedures arrange the arriving hackers and employees into safe boatloads. To get into a boat, a thread calls BoardBoat(); once the boat is full, *one* thread calls RowBoat(). RowBoat() does not return until the boat has left the dock. Assume BoardBoat() and RowBoat() are already written. Implement HackerArrives() and EmployeeArrives(). These methods should not return until after RowBoat() has been called for the boatload. Any order is acceptable (again, don't worry about starvation), and there should be no busy-waiting and no undue waiting (hackers and employees should not wait if there are enough of them for a safe boatload).