CSE P505, Winter 2009, Assignment 5
Due: Thursday March 12, 2009, 5:00PM

Last updated: February 28

- This assignment covers Concurrent ML.
- We have provided a Makefile. Compiling a Caml program to use threads requires some carefully placed flags. We have tested the Makefile on Windows with cygwin and on Linux; if you have trouble compiling on another platform let us know. See also the comments at the beginning of `hw5.ml`.
- Turn in your solution via the "Turn-in" link on the course website. Include `hw5.ml` and a file with your answer to problem 3.
- Understand the course policies on academic integrity (see the syllabus) and challenge problems.

1. In `hw5.ml`, use Concurrent ML to reimplement the bank account example from lecture 9, which implements the interface in `hw5.mli`. Still use Concurrent ML (the `Event` library), but use 4 channels (2 input and 2 output) and do *not* use a datatype. `get` operations should use different channels than `put` operations. Clients should not be able to distinguish your implementation from the one in lecture.

   Hints:

   (a) You need to use `choose` and `wrap` in a straightforward way.

   (b) The sample solution is a total of about 25 lines.

2. In `hw5.ml`, use Concurrent ML to complete an implementation shared/exclusive locks (better known as readers/writer locks), which have the interface in `hw5.mli`. Function `new_selock` creates a new lock. Functions `shared_do` and `exclusive_do` take a lock and a thunk and run the thunk. An implementation is correct if for each lock lk:

   - A thunk passed to `exclusive_do` lk runs while no other thunk passed to `exclusive_do` lk or `shared_do` lk runs.
   - If there are thunks that have not completed, then at least one of them gets to run.
   - If there are no uncompleted exclusive thunks, then the uncompleted shared thunks run at the same time.

   A correct solution to this problem has been given to you, but it is much longer and more complicated than necessary. This solution is for type `se_lock1`. In it, the "server" maintains explicit lists of waiting thunks.

   You need to complete the implementation for `se_lock2` by writing `new_selock2`. The sample solution is about 25 lines. This implementation uses a simpler protocol that relies on the fact that CML allows multiple threads to block on the same channel, effectively forming an implicit queue. You still need `choose` and `wrap`, but you need fewer code cases and fewer total messages.

3. In answering this question, remember `choose` is nondeterministic.

   (a) Consider the complicated implementation of shared/exclusive locks from problem 2 (the one given to you). Suppose a call to `shared_do1` lk blocks before a call to `exclusive_do1` lk. Describe the states, if any, under which we are *sure* the call to `shared_do` lk will unblock before the call to `exclusive_do` lk. Explain your answer.

   (b) Repeat the previous question for the implementation you completed. That is, suppose a call to `shared_do2` lk blocks before a call to `exclusive_do2` lk. Describe the states, if any, under which we are *sure* the call to `shared_do` lk will unblock before the call to `exclusive_do` lk. Explain your answer.

4. **(Challenge Problem)** Implement the commented-out interface for barriers in `hw5.mli`. Define type `barrier` and functions `new_barrier` and `wait`. If a barrier is created by `new_barrier` i and a thread makes one of the first $i - 1$ calls to `wait` with that barrier, then the thread should block until the i^{th} call, at which point all i threads should proceed. A thread that calls `wait` with a barrier after there have been i calls with that barrier should block forever.