Programming Assignment:
Dining UPC Philosophers

CS 377: Parallel Programming
Fall 2006
October 26, 2006

1 Administrative Details

Due: Thursday, Nov. 2, 2006

To be handed in: Your printed lab report, and printed source code for all three versions.

Comments: Be sure to update the comments to indicate your name and the location of your source file, as well as which version is implemented.

Report: Your lab report will answer the questions posed below, in addition to the standard elements required in all lab reports: your overall experience solving problems, problems encountered, how solved, lessons learned, etc.

Starting Code: Copy your starting code from my upc directory on the cluster: mlsmith/upc/philosophers.upc

2 Description

The given implementation of a solution to the dining philosophers problem "suffers" from several problems. First, the shared resources are forks, instead of chopsticks! But in the interest of shorter variable names, we’ll let this go. Second, more seriously, it *cheats*: it doesn’t block while waiting to acquire the locks (forks)! Instead, it uses the UPC function: upc_lock_attempt(). What Linda primitive is this similar to? In Linda, substituting blocking reads for predicate reads introduces the tradeoff...
between deadlock and livelock. This is not the case here. Why? Third, even though this solution doesn’t suffer from deadlock, it does suffer from something else. What?

3 Assignment

Your mission is to replace the `upc_lock_attempt()` statements with `upc_lock()` statements. Unfortunately, this will introduce the possibility of deadlock into the program. Why? (Note: if your program deadlocks, you can break out of it by pressing the CTRL+C keys.)

We discussed several strategies for avoiding the possibility of deadlock in the Dining Philosophers problem – for semaphore-based solutions:

1. One of the philosophers picks up her chopsticks in a different order from the other four.
2. Even numbered philosophers pick up chopsticks left-right; odd numbered philosophers pick up chopsticks right-left.
3. A "waiter" seats at most four philosophers at a time at the table.

UPC supports implementing two of these strategies directly by using the MYTHREAD and THREADS values. The third strategy, which uses a waiter to seat the philosophers, will use these two values, as well, but is a little trickier to pull off. (Hint: you may want to consider adding two additional states: `STANDING` and `SITTING`. Implement all three solutions. Remember, you must use `upc_lock()`, and not `upc_lock_attempt()`.)