Lab 7: The Dining Philosopher's Problem

In this lab, we consider an instance of the dining philosopher's problem implemented using PThreads. We propose and implement two solutions to this problem using the synchronization mechnanisms we have discussed in previous labs.

Issues addressed by this lab include:

- Practice reading code using Pthreads
- Practice using Pthread mutexes
- Practice using Pthread condition variables
- Practice analyzing concurrency control problems

Lab Materials

1. Slides
2. Lab Files

Assignment

Modify the starter code to prevent deadlock using two methods: (1) asymmetric locking order in different philosophers and (2) use of a Pthreads condition variable to simulate the presence of a waiter that will give both or neither of the chopsticks to a philosopher in an atomic operation.

What to Hand in

In addition to your implementation, you may want to go over the following questions, as you may be quizzed over them:

1. Describe the asymmetric solution. How does the asymmetric solution guarantee the philosophers never enter a deadlocked state?
2. Does the asymmetric solution prevent starvation? Explain.
3. Describe the waiter's solution. How does the waiter's solution guarantee the philosophers never enter a deadlocked state?
4. Does the waiter's solution prevent starvation? Explain.
5. Consider a scenario under a condition variable based solution where a philosopher determines at the time it frees its chopsticks that both chopsticks of another philosopher (Phil) it shares with are free, and so it sends the (possibly) waiting Phil a signal. Under what circumstances may Phil find that both of its chopsticks are NOT free when it checks?

You need to zip up your lab for submission. For this step, you should use the 'zip' target included in the lab's Makefile. Change the STUDENT_ID variable in the Makefile to your student ID and type:

```
make zip
```

Submit this zip file to Blackboard.

< Back to the Lab Home Page