EECS 678 Lab 08 Report

Author: Jace Kline 2881618

Questions

1. Describe the asymmetric solution. How does the asymmetric solution guarantee the philosophers never enter a deadlocked state?

The asymmetric solution to the dining philosophers problem is achieved by instructing the even-indexed philosophers to pickup their forks in left-right order, while the odd-indexed philosophers are instructed to pickup their forks in right-left order. The original (not asymmetric) solution will deadlock because, essentially, each philosopher will be waiting on their right neighbor to give up their shared chopstick and therefore is not giving up their own left chopstick to their left neighbor who is waiting on the shared chopstick between them. This continues all the way around the table until nobody is sharing any chopsticks. When we deploy the asymmetric solution, we remove the deadlock possiblity by ensuring that all the philosophers are never all waiting on their left-hand chopstick while not giving up their right-hand chopstick to their partner. By having alternating pick-up patterns, the whole table cannot all be waiting in a deadlock loop.

2. Does the asymmetric solution prevent starvation? Explain.

The asymmetric solution does not prevent starvation. The reason for this is as follows: Suppose philosopher A has a fork that philosopher B wants. Depending on the scheduling of the threads, there is no guarantee that the free fork will be picked up by philosopher B (due to it waiting on its other fork) before philosopher A's thread is rescheduled and attempts to pick up that fork again. Hence, although starvation is relatively unlikely, the particular scheduling mechanism or sheer probability could induce it.

3. Describe the waiter's solution. How does the waiter's solution guarantee the philosophers never enter a deadlocked state?

The general idea behind the waiter's solution is that a philosopher must ensure that both of his chopsticks are available before picking up the chopsticks to eat. In this solution, we use a 'waiter' mutex that provides mutual exclusion to the state of the forks (an array). With access to the waiter, a philosopher will continuously (in a loop) inquire whether or not both of his forks are available and invoke a 'wait' call on his condition variable 'can_eat' where control is given to other threads until a signal is acquired. This means that when a neighbor philosopher thread provides a signal that he has given up the chopstick, this thread will wake up and check its condition again. Once it finally receives the signal and the condition is true, the status of his chopsticks will be updated to taken, and then he will release the waiter mutex and begin eating. When done eating, the philosopher will wait to get the waiter mutex again and his forks will be changed to the status available. Then the philosopher will send a signal to his neighbor philosopher threads' can_eat condition variables and release the waiter mutex. His neighbors, in turn, will be woken up and this general process continues. This prevents deadlocking because it must be the case that a philosopher knows that he can pick up both chopsticks simultaneously before doing so. Deadlocking can only occur if two threads are mutually waiting on a resource that the other one has. In this case, this is not possible because a philosopher thread can never only have one chopstick.

4. Does the waiter's solution prevent starvation? Explain.

The waiter's solution does not prevent starvation because there is never a guarantee that when one a philosopher's thread is signaled that both chopsticks will be simultaneously available. Hence, a philosopher could, in theory, keep waiting forever for the condition to hold based on the timing of the scheduling and other factors.

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?

This is possible in the case that between the time Phil is signaled by the first philospher and Phil checks the condition again, that Phil's other neighbor philosopher (Bob) was also signaled because both of his chopsticks were also available. In this case, it could occur that Bob's thread wakes up first, picks up the shared chopstick between Phil and Bob (along with the other one from his other neighbor), and then mutates the shared chopstick between Phil and Bob to be in 'taken' status. Then, when Phil finally checks his condition (when his thread is scheduled), the chopstick that was available between Phil and Bob is now actually taken by Bob, and therefore the condition for Phil to pick up his chopsticks will be false.