Parallel Computing Theory Assignment 1

Rachel Hwang

January 21, 2014

- Exercise 2
  1. Safety. The "bad thing" avoided is patrons being served out of order.
  2. Liveness. The "good thing" ensured is that things will eventually come back down.
  3. Liveness. The "good thing" ensured is that one process enters its critical section, and both are not simply waiting forever.
  4. Safety. The "bad thing" avoided is that an interrrupt will never go more than a second unacknowledged.
  5. Liveness. The "good thing" ensured is that a message will be printed eventually.
  6. cost of living
  7. Safety. The "bad things" avoided are immmortality and government bankruptcy.
  8. Liveness. The "good thing" ensured is that you will eventually indentify man from harvard.

- Exercise 3
  Bob and Alice can use a two can system in which they each have a can on their window sil. Alice and Bob begin by setting his can upright. If Bob's can is upright, he pulls Alice's string and enters the yard. When he's finished, he crosses the yard and resets Alice's can and leaves the yard. If Alice wants to use the yard, she must first check if her own can is down. If not, she is free to use to yard. She enters the yard, pull's Bob's can and reset's it when she is finished. Essentially, each person's can represents their right to claim the yard. Each person basically locks the other while using the yard. and unlocks the other person's can, but not their own, when they are down. This ensures that each person uses the yard only when the other person has signaled that is time for them to do so by resetting their can.

- Exercise 4
  If the initial state of the switch is known to be off, the prisoners select a leader. This leader only ever turns the switch on. If the switch is already on, he does nothing. Each other prisoner turns the switch off if it is on and does nothing if it is off already. The leader merely counts the nubmer of times he has flipped the light on. After the (p-1)th switch on, where p is the number of prisoners, he knows that every other prisoner must have visited the room and flipped the switch once. and can claim as such.

  If the initial state of the switch is unknown, the strategy is the same except everyone (minus the leader) must flip the switch on twice. The leader waits until she has flipped the switch off 2p-2 times. At 2p-2 on lights, either every (non-leader) person has turned on the light twice or every (non-leader) - 1 has touched the light and it was initially on.

- Exercise 5
  The prisoner at the back of the line cannot be saved. His job is to transmit binary information to the rest of the group. Beforehand, the prisoners can agree that "blue" will mean that there are an odd number of hats, and "red" will mean an even number.

  Let us show that this method works by induction. Base case: the penultimate man in line will recieve information, even or odd blue hats, and since he can see all the hats before him, can derive the color of his own hat.

  Assuming that for any man, all the hat colors before him have been stated correctly by the time his turn is reached, the man will know the color of every hat apart from his own (and excluding the last man), since he can also see all the hats before him. If he knows whether there are an odd or even number of blue hats in total, he can add the number of blue hats already called to the number of blue hats he sees. If this sum has the property (odd/even) that the last man named, then this man's hat must be red. If the sum does not, his hat must be blue. By induction, this strategy allows all but the last prisoner to know his hat color, so long as P¿2.

- Exercise 7

$$\frac{1}{1-p+p/2} = S_2$$
$$1 - 1.5p = \frac{1}{S_2}$$
$$p = \frac{2(1-1/S_2)}{3}$$

This p value is the proportion of parallizable program. So this can be plugged back into Amdahl's law like so:

$$p = \frac{2(1-1/S_2)}{3}$$
$$S_n = \frac{1}{1-(2(1-1/S_2)/3) + \frac{(2(1-1/S_2)/3)}{n}}$$

- Exercise 8
  We want a speedup for at least 5x in order for the multiprocessor to be worth it. Using Amdahl's law, for S = 5 and n = 10:

$$5 = \frac{1}{1-p+p/10}$$
$$p = 8/9$$

Therefore the portional of an application that is parallizable must be at least $\approx$ 89% to be worth the multiprocessor machine.

- Exercise 11
  Mutual Exclusion: satisfied. Since the outer loop invariant only allows threads to exit the loop if it is their turn, and turn can only have one value, only one thread may exit the lock at one time. All other threads are placed in the inner loop to wait until the lock is not busy. Thus, only one thread can take its turn at a time and so thread critical sections can be never allowed to overlap if lock is called before and after.

  Starvation-free: not satisfied. Consider the case where several threads call lock at about the same time. Thread 1 gets the turn and threads 2 and 3 are left waiting in the inner loop. Once thread 1 calls unlock, thread 2 jumps and grabs the turn. When the lock is next free, thread 1 gets back into the waiting loop and if thread 3 is unlucky, could once again grab the turn before thread 3 has an opportunity. There is nothing preventing thread 3 from never getting a turn, and so if conditions are unfavorable, it may be starved out.

  Deadlock-free: satisfied. So long as unlock is called exactly as many times as lock is called, there will never be any dead lock. This code has no competing conditions which would allow that to occur. As in our discussion of Mutual Exclusion, only one thread may have its turn at once; the remaining threads are left waiting in the inner loop. Thus, as soon as the lock is unlocked, the next thread may proceed to grab control and the process continues. So any thread which calls lock must either make it through or be stalled in the inner loop, a waiting process that has a definite endpoint so clong as unlock is eventually called.

- Exercise 14
  sdf

- Exercise 15
  This lock does not satisfy mutual exclusion. Consider the case where the lock is locked and both threads 1 and 2 have called it, and so are waiting in the while loop. Once the lock is unlocked, threads 1 and 2 are both free to proceed. The lock will be locked by one of them, but both will succesfully exit the lock function with the lock locked. This means that mutual exclusion is not preserved since multiple threads may be operating freely while the lock is locked.

- Exercise 16
  At least one thread gets the value STOP: This must be true because if there are n threads currently in the visit method, the i value must be set to one of their indices, so even if all n threads proceed to line 14, (last == i) must be true for one of them, so one must get the value STOP.

  At most $n-1$ threads get the value DOWN: Like the complement of the previous proof. At least one value must be STOP, so if n threads make it to line 14, at most n-1 threads can continue to the else statement and return DOWN.

  At most $n-1$ threads get the value RIGHT: this must be the case since the goRight variable is initialized to false. One thread must be the one to change its value to true, which may only be accomplished by going past the point where recieving the value RIGHT would be possible.