Suggested answers to Week 4,5 assignments

Activity 4.3; Homework 4.1

Activity 4.3

- The semaphore's integer state variable (value) will increase by 1, i.e., the effect of release() is *remembered* even if no producers are yet asking for an empty slot.
  - That's why the race condition illustrated on Slide 6.36 won't happen for semaphores.
- The notify() will have no effects (since there are no producer threads in the wait set that can be waked up).
  - Hence, the race condition on Slide 6.36 could happen if we called wait()/notify() without holding a lock.
- The number of full slots is kept with a counting semaphore itself. So there's no need for a separate count variable.

Homework 4.1

- Attempt 1 is incorrect. It does not satisfy mutual exclusion. Initially, wantEnter[0] == wantEnter[1] == false. Consider this interleaving of P0's and P1's execution:
  - P0 tests wantEnter[1] == false, exits while loop;
  - P1 tests wantEnter[0] == false, exits while loop;
  - P0 sets wantEnter[0] to true and enters CS;
  - P1 sets wantEnter[1] to true and enters CS;

- Attempt 2 is incorrect. It doesn't satisfy progress (initially, turn == 0):
  - P0 does *not* want to enter CS, turn == 0 indefinitely;
  - P1 wants to enter, indefinitely stuck in while loop (although CS is available) because turn is 0;

Homework 4.1 (cont'd)

- According to Peterson's Algorithm, after *i* exits the critical section (CS), before it can enter again, it must set turn to *j*. So if *j* is waiting to enter (i.e., flag[*j*] == true), *i* must now wait at the while loop until after *j* got its own chance to enter the CS. Hence, *j* can't be beaten twice in a row, and bounded waiting is satisfied with a bound of 1.