Fac969 & hc27426

Q1)

a.
Each prisoner starts with one switch point. A switch point is spent if they turn the switch on, and received if they turn it off. So if they go into the room with one point, and the light is on, and they turn it off, they have 2 points. If they can keep track of this, when one of them gets P points, everyone must've been in there at some point to 'deposit' their point

b.
When the initial state is unknown, the above strategy cannot work. This is because if the switch starts on, there will be P+1 points in total and P points could be reached with one prisoner still never entering the room.
The strategy is to give each prisoner 2 points to start instead of 1.
Then, when one prisoner has 2P total points, everyone must've entered the room.
If the switch was on initially, then 2P-1 points came from prisoners. Every prisoner must've been in the room at least once in that case, because only 2P-2 points are possible if a prisoner never went into the room to spend any.

Q2)

a.
Raw spin output:

```
63:     proc  0 (user:1) q2a.pml:13 (state 1)    [flag[_pid] = 1]
                flag[0] = 1
                flag[1] = 1
64:     proc  1 (user:1) q2a.pml:21 (state 7)    [flag[_pid] = 0]
                flag[0] = 1
                flag[1] = 0
65:     proc  1 (user:1) q2a.pml:22 (state 8)    [goto again]
66:     proc  1 (user:1) q2a.pml:13 (state 1)    [flag[_pid] = 1]
                flag[0] = 1
                flag[1] = 1
67:     proc  1 (user:1) q2a.pml:14 (state 2)    [turn = _pid]
68:     proc  1 (user:1) q2a.pml:15 (state 3)    [(((flag[(1-_pid)]==0)||(turn==_pid)))]
69:     proc  1 (user:1) q2a.pml:17 (state 4)    [ncrit = (ncrit+1)]
                ncrit = 1
70:     proc  0 (user:1) q2a.pml:14 (state 2)    [turn = _pid]
                turn = 0
71:     proc  0 (user:1) q2a.pml:15 (state 3)    [(((flag[(1-_pid)]==0)||(turn==_pid)))]
72:     proc  0 (user:1) q2a.pml:17 (state 4)    [ncrit = (ncrit+1)]
                ncrit = 2
spin: q2a.pml:18, Error: assertion violated
```

Fac969 & hc27426

Explanation:

1. P0 sets wantCS[0] to true
2. P1 sets wantCS[1] to true
3. P1 sets turn = 1
4. P1 passes guard condition b/c turn == 1
5. P1 enters crit section
6. P0 sets turn = 0
7. P0 passes guard condition b/c turn == 0
8. P0 enters critical section
9. Assertion violated

b.

Raw spin output:

```
5384:   proc  1 (user:1) q2b.pml:21 (state 8)    [goto again]
5385:   proc  0 (user:1) q2b.pml:12 (state 1)    [turn = (1-_pid)]
                turn = 1
5386:   proc  1 (user:1) q2b.pml:12 (state 1)    [turn = (1-_pid)]
                turn = 0
5387:   proc  1 (user:1) q2b.pml:13 (state 2)    [flag[_pid] = 1]
                flag[0] = 0
                flag[1] = 1
5388:   proc  1 (user:1) q2b.pml:14 (state 3)    [(((flag[(1-_pid)]==0)||(turn==_pid)))]
5389:   proc  1 (user:1) q2b.pml:16 (state 4)    [ncrit = (ncrit+1)]
                ncrit = 1
5390:   proc  0 (user:1) q2b.pml:13 (state 2)    [flag[_pid] = 1]
                flag[0] = 1
                flag[1] = 1
5391:   proc  0 (user:1) q2b.pml:14 (state 3)    [(((flag[(1-_pid)]==0)||(turn==_pid)))]
5392:   proc  1 (user:1) q2b.pml:17 (state 5)    [assert((ncrit==1))]
5393:   proc  0 (user:1) q2b.pml:16 (state 4)    [ncrit = (ncrit+1)]
                ncrit = 2
spin: q2b.pml:17, Error: assertion violated
```

Explanation:

1. P0 sets turn = 1
2. P1 sets turn = 0
3. P1 sets wantCS[1] to true
4. P1 passes guard b/c wantCS[0] = 0
5. P1 enters critical section
6. P0 sets wantCS[0] to true
7. P0 passes guard b/c turn == 0
8. P0 enters critical section
9. Assertion violated

Q3)

The filter algorithm works by having a series of gates that threads must progress through to reach the critical section (CS). Each gate will stop one thread V from progressing as long as there are other threads ahead and V was the last to arrive at that gate. Using this design, we can employ N-1 gates to ensure that only one thread at a time can make it through all of the gates, even if all N are trying to enter, because all but one will be stopped at lower levels.

Modifying this for L-exclusion is relatively simple, we just need to reduce the number of gates from N-1 to N-L, where L is the maximum number of threads allowed in the critical section. This way, only N - (N-L) = 0+L = L threads can make it through the gates.

There is one more modification necessary, which is that the busy wait while loop must not keep waiting if there is *any* thread at the same or a higher level, but if there are *L* threads waiting at the same or a higher level. This way threads can move on as long as there are < L threads ahead, and L threads will be able to make it through all the gates at once.

Pseudocode:

```
For k in range(1,N-L+1) {
      Gate[i] = k;
      Last[k] = i;
      Int ahead = 0;
      Do {
            Ahead = 0;
            For (b in range(0,N)) {
                  If (b!=i && gate[b] >= k) ahead++;
            }
      } while (ahead >= L) && (last[k] == i);

// we update the # of threads ahead or at same level within the busy
// wait loop. We reduce gates from N-1 to N-L. otherwise the same
```

Proofs:

L-exclusion:

Suppose there are L threads in the critical section. WLOG, assume some thread T_n now is able to enter the critical section, and there are now L+1 threads in the CS. T_n must have been waiting on the last gate level, and there are 2 cases for how it continued:

Case 1: (last[k] == pid) no longer true

If this is the case, some other thread T_{n-1} entered the same level as T_n, causing T_n to no longer be the last thread to have entered. Given the supposition that there were already L threads in the CS at this time, there are only N-L-1 threads remaining besides T_n. Because T_n was at the highest level, T_{n-1} could not have moved onto the highest level because there was no thread waiting at a higher level, since T_n was at a higher level. It must have moved onto the highest level because some other thread T_{n-2} entered its level and wrote to the last variable. The same reasoning shows that T_{n-2} could only have moved on if some thread T_{n-3} wrote to the last variable and allowed T_{n-2} to move on. We cascade this reasoning all the way to the bottom level.

The thread at the bottom level T_1 can only have moved on if some other thread entered the first level and became the last thread to write to the last variable. However, there were L threads in the critical section to begin with, and N-L threads stuck in the gates, one at each level. This is a total of N threads, meaning there cannot have been another thread that enters at the bottom level and causes T_1 to move on because there are not enough threads to do so. This is a contradiction, and thus this condition cannot exist.

Case 2: (gate[j] >= k for L j's != pid) no longer true

If there are no longer L or more threads ahead of T_n , then there are no longer L threads in the CS, because the only way for a thread to be ahead of T_n is if it is in the critical section, and the only way for the thread to fall to a lower gate than T_n is by releasing the lock and leaving the critical section. This is a contradiction of the supposition that L+1 threads entered the critical section, thus this condition cannot occur.

L-Starvation-Freedom:

Suppose there are L-1 threads in the critical section, and some thread T is at the first gate and was the last thread to enter (set the last variable). Suppose that L-2 of the L-1 threads in the critical section are halted, so only 1 thread in the critical section is alive. This thread will eventually finish its work and leave the critical section. There are 2 cases:

Case 1: There are no other threads at higher gate levels waiting to enter the CS.

In this case, as soon as the active thread in the CS finishes, T will advance through each gate with ease as there will be fewer than L threads ahead of it and so it will skip each busy wait loop until it is in the CS.

Case 2: There are threads at higher gate levels that are waiting to enter the CS.

In this case, as soon as the active thread in the CS finishes, some other thread ahead of T will enter the CS because there will no longer be L or more threads ahead of it. Then some other active thread in the CS will eventually finish, and let another thread into the CS. As long as there is at least one active thread in the CS (which is given), this will keep occurring until there are no longer L or more threads ahead of T. As soon as this is the case, T passes its guard condition at each gate until it reaches the last gate. Note that it is not possible for a thread to pass T, because as it enters the same level as T, it writes to the last variable and T will be allowed to move on. Because no thread can pass T and the number of threads ahead of it will decrease with time, T will eventually be allowed to enter the critical section.