50.530, Year 2015: Problem Set 6

Question 1 (15%): Exercise 1.

Answer: No, there is no race condition according to the happens-before approach. Yes, there is a race condition on variable count (the first and last line).

Question 2 (15%): Exercise 2.

Answer: The lockset algorithm finds the race condition at line 1 since count is a shared variable and it is accessed without any lock held (so that the intersection is {}).

Question 3 (20%): Exercise 3.

For variable count, after the first count++, it moves from state Virgin to state Exclusive

After the second count++, it moves from state Exclusive to Shared-Modified state since it is accessed by a new thread with write privilege.

The checking then happens and since there is no lock held, a race condition is reported.

Question 4 (20%): Exercise 4.

Answer: There is no race. The race on childThread is dismissed by lockset (i.e. the common lock on main is applied) first. Then, the race on globalFlag is dismissed by a happens-before relation from thread creation to any action of that thread.

Question 5 (20%): Exercise 5.

After applying the lockset-based algorithm, we get the following race. Notice that since line 10 is not executed in this test case, the pair 1 and 10 is not reported.

- Race on z: line 7 and 5

After applying happens-before filtering, we get

- no race on z since there is happens-before from unlock(L) to lock(L)

As a result, no race is reported.

Question 6 (30%): Exercise 6.

Assume that thread 1 is scheduled first, after executing line 1, it is postponed; Thread 2 is scheduled to run line 7 and then is blocked at line 8. Thread 1 is then removed from postponed. Unfortunately, it will be picked again (since it is the only one enabled). Then no race is reported.

If thread 1 is postponed before executing line 1, thread 2 execution all the way until it terminates and therefore no race is reported either.