1. There are race conditions on the c array because UpdateC3 could be updating the c array at the same time as UpdateC1 or UpdateC2 (it uses a different mutex variable than UpdateC1 and UpdateC2). So there are four possible outputs:

(1) **a a** (corresponding to c array’s final values 0 0)

(2) **a b** (corresponding to c array’s final values 0 1)

(3) **b a** (corresponding to c array’s final values 1 0)

(4) **b b** (corresponding to c array’s final values 1 1)

1. There are race conditions on the c array because two of the three created threads could be updating the c array at the same time (the third argument to sem\_init is 2). So there are four possible outputs:

(1) **a a** (corresponding to c array’s final values 0 0)

(2) **a b** (corresponding to c array’s final values 0 1)

(3) **b a** (corresponding to c array’s final values 1 0)

(4) **b b** (corresponding to c array’s final values 1 1)

1. Two threads are created to update the global c and x. Only one of these two threads can update the global c and x at a time (the third argument to sem\_init is 1). The global c is incremented 2000000 + 4999999 = 6999999 times and the global x is incremented 2000000 times. (The x in UpdateC2 is local and the global x will be hidden in it.) So the final output is **b a** because 6999999%2==1 and 2000000%2==0.
2. See A7p4.c and makefile
3. See slides 8\_Concurrent\_Programming.pptx or textbook Section 5.7 or online resources.