Question 9: Mutual Exclusion

The answer is B. The algorithm guarantees mutual exclusion but not deadlock freedom. The annotated code for the algorithm appears in Figure 1. On an intuitive level, the algorithm guarantees mutual exclusion because once some thread passes Line 2, all threads that excute Line L later will get stuck at Line 1, and among any collection of threads that pass the test at Line 1 concurrently and race to get to Line 3, only the one that wrote last in Line L can get past Line 3, and all later threads will never get past Line 1 until this winning thread leaves the critical section and executes Line 5.

However, proving mutual exclusion rigorously requires to state an invariant about the system state because we must argue about the possible races that might occur, including ones in which perhaps a thread might execute Line 5 while others are possibly executing Line 1. It is perhaps educational to see how one goes about proving formally that such races cannot occur.

---

x, y: *shared locations, initially* $y = 0$;

0b: $exit - remainder$
while True:
L: $x := i$;
1: **if** $y \neq 0$ **continue**;
2: $y := 1$;
3: **if** $x \neq i$ **continue**;
4a: $enter - critical - section$
4b: $exit - critical - section$
5: $y := 0$;
0a: $enter - remainder$

Figure 1: Code for Process i.

We describe the program states for n threads with ids $\{1 \ldots n\}$ as a combination of *program counter* values as above, local variable values and shared variable values. The algorithm execution can be thought of as a sequence of transitions from one state to the next, where in each step some thread executed an operation.

In any state of the algorithm, we define:

$F1$**:** the set of thread ids i such that $x = i$ and $pc(i) = 2$,

$F2$**:** the set of thread ids i such that $x = i$ and $pc(i) = 3$,

CR**:** the set of thread ids i in the critical section, having $pc(i) \in \{4a, 4b, 5\}$. In other words, we consider a thread in the critical region until it completes executing Line 5.

RM**:** the set of thread ids i in the remainder (that is, the thread is not in acquire, release, or critical section), having $pc(i) \in \{0a, 0b\}$, .

We can prove the following lemma by induction on the length of the execution, a sequence of reads, writes, and advancements of the program counters.

Lemma 0.1 *The following invariants are true in any reachable state of the algorithm, where $|S|$ denotes the cardinality of a set S.*

$I1:\ |F1| + |F2| + |CR| \leq 1.$

$I2:$ *If* $|F2| + |CR| > 0$ *then* $y = 1.$

The mutual exclusion property will follows from invariant $I1$ of Lemma 0.1 but we need both invariants to make the inductive argument work.

Proof: By induction on the length of an execution. Initially $I1$ and $I2$ hold vacuously since all threads are in the remainder ($pc(i) = 0$), and $y = 0$. We assume that the invariants hold in some state s, and for the inductive step, go over all the possible operations (or operation and pc changes) of a thread i, leading from s to a new state s' in a given execution. We refer to these operations by their relative line numbers in the code. First note that in any state, there is at most one thread i for which $x = i$, so $|F1|$ + $|F2|$ is always less than or equal to 1.

Line L CR remains the same, and $|F1|$ and $|F2|$ can only decrease, so $I1$ holds in s'. Since y remains unchanged, and $pc(i) = 1$ following the operation, $I2$ holds.

Line 1 If $x \neq i$ both $I1$ and $I2$ hold immediately. The only state change is in $pc(i)$. If $x = i$ in s then it was the case that $|F1|+|F2| = 0$. From $I2$ we thus have that in s if $|CR| > 0$ then $y = 1$. If y was 1, then i reads $y = 1$ and and continues so in s' $pc(i) = L$, implying that $I1$ and $I2$ continue to hold since $F1$ and $F2$ remain unchanged. If y was 0, process i will read $y = 0$ and in s' $pc(i) = 2$, implying $|F1| = 1$ and $|F2| = 0$. Since it read $y = 0$ by $I2$ this implies $|CR| = 0$ in s, so in s' $|F1| + |F2| + |CR| \leq 1$ and $I1$ and $I2$ hold.

Line 2 $I2$ holds since in s' $y = 1$. If $x \neq i$ in s, $I1$ also holds immediately. If $x = i$, then in s $|F1| = 1$ and $|F2| = 0$, and in s' $|F1| = 0$ and $|F2| = 1$, and $I1$ holds.

Line 3 If $x \neq i$ in s, then $F1$ and $F2$ and CR remain unchanged implying that $I1$ and $I2$ hold. If $x = i$, then in s $|F1| = 0$, $|F2| = 1$, and $|CR| = 0$, and in s' $|F1| = 0$, $|F2| = 0$ and $|CR| = 1$, so $I1$ holds. By $I2$ and $|F2| = 1$ it must be that $y = 1$ in s and remains so in s', implying $I2$.

Line 4 (a and b) Both invariants hold immediately.

Line 5 Since $pc(i) = 5$ in s, it follows that $|CR| = 1$ and by $I1$ that $|F1| + |F2| = 0$. Thus, since in s' $pc(i) = 0a$, $F1$ and $F2$ remain unchanged, and $|CR| = 0$, implying that both $I1$ and $I2$ hold.

Line 0a All invariants hold immediately.

Line 0b Since in s', CR is unchanged and $|F1|$ and $|F2|$ can only decrease, both the invariants hold.

■

Finally, the reason this algorithm does not guarantee deadlock freedom is that when a thread reaches Line 3 and the check fails because some other later arriving thread executed Line L, the value of y will remain 1, and no thread will ever enter the critical section again.