SE-223::CSE

Reading #5::Eraser

Question #1

According to the lockset algorithm, when does eraser signal a data race? Why is this condition chosen?

Lockset Algorithm (initial version) checks every memory address accessing. Any same-addressed requests will be forced by a lock, which ensures the address sequences.

This condition is quite very strict, even those addressing that shares similar addresses that's too far away to come up with a race, will also be protected by a arbitrary lock, which costs a lot.

Lockset Algorithm (improved version) provides a much lesser general algorithm: Initially nobody gets a lock. And reading operations doesn't require a lock (We allow dirty readings.), only writing needs that. Under that, we also allows one and only one reader when a writer is writing. That could ensure the "consistency" of reading (will not read different values at same transaction) but not the "freshness" (not ensure that's the latest value).

Question #2

Under what conditions does Eraser report a false positive? What conditions does it produce false negatives?

According to the article, when no other thread could reference my memory addresses, we don't have to lock them at all. The typical time would be the Large Memory Initializing time, at that time we will quickly writes a lot chunk of memory. If at that time we can omit the lock requesting operation, we might save lots of wasting time.

Question #3

Typically, instrumenting a program changes the intra-thread timing (the paper calls it interleaving). This can cause bugs to disappear when you start trying to find them. What aspect of the Eraser design mitigates this problem?

The original article says,

> In contrast, the programming error in Figure 2 will be detected by Eraser with any test case that exercises the two code paths, because the paths violate the locking discipline for y regardless of the interleaving produced by the scheduler. While Eraser is a testing tool and therefore cannot guarantee that a program is free from races, it can detect more races than tools based on happens-before.

Since the interleaving problem is not a problem caused by the program itself (just like some minor error on the CPU instruction layer), but it randomly happens when debugging. So it's impossible to detect it when this error happens.

So the problem transits to: when such situation happens (happens-before error), how should we deal with that and tries our best to avoid incorrectness.

The basic thoughts is: when we tries to acquire a lock, but it isn't actually exist, what should we do? When a new variable v is initialized, its candidate set $C(v)$ is considered to hold all possible locks. When the variable is accessed, Eraser updates $C(v)$ with the intersection of $C(v)$ and the set of locks held by the current thread. This process, called lockset refinement, ensures that any lock that consistently protects v is contained in $C(v)$. If some lock l consistently protects v, it will remain in $C(v)$ as $C(v)$ is refined. If $C(v)$ becomes empty this indicates that there is no lock that consistently protects v.

Simply saying, the lockset refinement could ensure any lock will exist in the lock list. Thus even though the "happens-before" stuff will not bring a difference.