Eraser

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

当一个变量的状态为Shared-Modified且其对应的C(v)为空时，eraser会报告数据竞争。因为处于这个状态表明有多个线程读这个变量且有至少一个线程写这个变量或者有多个线程写这个变量，这两种情况下，都需要用锁来保护这个共享变量。而状态Shared--多个线程读这个共享变量，因为只读，所以可以不用锁；状态Exclusive--单一线程读写这个变量，无需用锁；状态Virgin，变量还没被初始化。所以我们选择了Shared-Modified状态。

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

False positive:

- 重用内存而没有重置shadow memory
- 用了私有的锁，Eraser不知道你用了，以为你没用
- Benign Race：一些人为的或有意或无意的，不影响结果正确性的情况。

False negitive：

若将Eraser改进为能检测multiple protecting locks情况下数据竞争的版本--只有在写的时候才更新C(v)，读和写都检测C(v)是否为空；这时就可能会出现假阴性：线程1拿着锁m1去读变量v，线程2拿着锁m2去写变量v，如果读发生在写之前，eraser就不会报告数据竞争（因为读不更新C(v)),而数据竞争确实发生了。

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?

Eraser是在算法层次上解决这个问题的。因为Eraser用的是改良的lockset算法，lockset不同于基于happen-before的算法，基于happen-before的算法根据不同线程的语句运行是否交叉来判断是否产生数据竞争，因此非常依赖于程序调度；而lockset算法是看访问某个变量时是否有锁来保护它，如果没有锁保护共享变量就会报告错误，即使当前的运行没有出bug，因此lockset是不依赖于调度的，减轻了这个问题。

Please raise at least one question of your own for the discussion.

- 文章中说在操作系统中检测数据竞争和在用户态的区别很大，因为系统中更多地是用信号量而不是锁，信号量和锁有啥区别吗?

嗯。我知道了，文章里说信号量没有owner，是这个意思：锁是这个进程拿了，做完事情就必须要放，而信号量的概念更像是一种资源，一个程序执行了P，之后它不一定会执行V（比如producer和consumer程序就是consumer只执行P，producer只执行V），只能说可以把信号量P、V操作当作锁来用，但是信号量还能干更多别的事情，所以就不好判断某一个信号量是用来保护哪个变量的。