HW Group 4(Chris Worthley and Daniel Burns)

1. Lock Analysis

If the code about guard is removed, then the code would be incorrect. This is because, without the guard code, then mutual exclusion is violated. This is because the guard code is basically a lock for the lock's internal state, and it protects the lock's status and the wait queue. So, if the guard code lines are removed then multiple threads can concurrently read and write both the flag (lock status) and q (the wait queue). Because of this, mutual exclusion is violated, and the code is incorrect.

Example execution to show lock is incorrect:

Suppose two threads, T1 and T2, call lock(m) at the same time when the lock status is free (flag==0):

- T1 enters lock(m) and sees flag==0, and before it sets flag=1 the scheduler switches to T2
- T2 also sees the flag==0 and because of that, it also takes the lock
- So, both T1 and T2 write flag=1 and return from lock
- **Both threads believe they hold the lock at the same time, so mutual exclusion is violated

2. Lock Analysis II

If line 12 is replaced with lock->flag = lock->flag – 1, then the lock algorithm becomes incorrect. This is because it messes up unlocking, however not for all cases. If flag==1 then it works as intended as it sets the flag to 0, however it becomes incorrect when the flag does not equal 1. This would typically only happen when the developer does not know how to use locks properly however, as the case where it messes up only happens if either a double unlock is called or unlock is called without a lock in the first place.

Example execution explaining this:

- A thread is called and, in the thread, unlock is called when a lock is not held
- In this thread, if flag==0, the code will set the flag to -1(flag=-1)
- So, because of this, the lock state will get corrupted as the flag state can only be either 0 or 1
- If another thread runs lock after this happens, it will call the function LoadLinked(&flag). It will see -1 for flag, which does not equal 1, so it believes the lock is free

- So, it successfully calls the function StoreConditional(&flag,1), so 2 threads believe they can hold the lock at the same time
- Because of this, mutual exclusion is violated and the lock algorithm is incorrect