**Portfolio Milestone - Analysis**

Andrew Barnes

Colorado State University Global

CSC450-1

Reginald Haseltine

6/2/2024

**Concurrency Performance Awareness**

Using *std::mutex* creates mutual exclusion within my program, but lock contention can occur when many locking implementations exist. Specifically, the mutex is locked each time the count variable is updated and printed to the console within each loop iteration of the *countUp* function. However, the *countDown* function unlocks the mutex for the sleep duration to assist with lock contention mitigation. Lock contention is when more than one thread or process tries to acquire the same lock simultaneously. The outcome of that can lead to threads having to wait for the lock to be released, resulting in latency. Lock contention is simply a complexity that programmers must be aware of during the design phase of software.

**String Vulnerabilities**

Aside from printing to the console *“Count: “*, I had no string usage within my program. In other programs that implement strings with concurrency, erroneous use of synchronization techniques may cause various threads to read and write a string simultaneously, otherwise known as data racing. A solution for data racing is to use locking mechanisms, which I did use in my program, as previously mentioned.

**Data Type Security**

The data types I used were *int* and *bool*, which are considered primitive datatypes and they are usually secure because they won’t fall victim to buffer overflows like arrays or strings. Again, avoiding data races is a must, so that’s where *std::mutex* and *std::condition\_variable* come into play to make sure *count* is accessed and modified by one thread at a time and that the *countDown* function waits until *countUp* has completed. This creates an order of operations critical to the execution of any program implementing concurrency. When it comes to data types and user input, dynamically allocated memory must be managed properly, otherwise data leaks may occur. If I made my program allocate a user-defined input integer, as opposed to the static integer 20, I would need to provide many constraints to ensure the input provided by the user is cooperative with the counting application.