C++ Program Analysis

Matthew S McCormick

CSU Global

Section C

CSC450

Farhad Bari

The most prominent vulnerability of using concurrency in my program is the sharing of the count variable between threads. This is because both threads modify the count variable, leading to a race condition. The solution to this problem is to use a mutex which locks itself when entering the critical region of modifying the count variable. The mutex stays locked until the count is fully incremented to 20 or decremented to 1. This prevents the modification of the count variable when it is in use by the countUp or countDown functions. I initialize the countUp thread first, so this is the first one that locks the mutex and thus the count up to 20 is printed first. Without mutual exclusion, the output of my program becomes unpredictable. This is because one function ticks up count while the other ticks it down, so it bounces between numbers instead of only going up or down.

My code is also designed to avoid deadlocks. It is not possible for the program to get stuck in the loop of upticking or downticking count, as the loop conditions will always be met by changing count, and they are both in critical regions. There is also only one way of entering and exiting each of the functions, so the mutex is always locked and unlocked when a function is run. I only lock the mutex when entering critical regions, and unlock it as soon as I leave them to prevent overlapping regions.

My program doesn't use many strings, but it is still designed to use them securely. My program doesn't take in any user input, so I don't need to worry about buffer overflows involving strings. I do print the string " " to the console to put spaces in between numbers. This string is automatically null terminated with \0, as it's not a manually built array of characters. The printf() function can also cause buffer overflows if not used carefully, so I decided to use the more secure cout instead to be safe.

The primary data types used in my program are shorts, string literals, classes, and functions. I decided to use a short for the count. This is because it is not possible for it to get below 0 or above 20 due to the loop conditions surrounding incrementing or decrementing it, preventing an integer overflow. The count variable is declared globally so that all functions

within the program can access it. The string literal I use is “ “ for printing out the count, which is automatically given a null terminator. The class I use is mutex, which it is not recommended to delete while it is locked so I avoid that. My functions are outside of a class, as they don’t need to be accessed from anywhere outside of the file they are declared in.