**CTA 7 Concurrency C++**

Jordon Paynter

Department of Computer Science, Colorado State University Global

CSC 450: Computer Programming II

Professor Farhad Bari

October 29, 2023

**Introduction**

We’ll examine my C++ application that showcases fundamental concurrency concepts. The application creates two threads—one to count up and another to count down—while considering performance issues, potential vulnerabilities related to string usage, and the security of data types.

**Performance Issues with Concurrency**

It uses a mutex to protect the shared counter variable, preventing race conditions. Since both need to use count, it causes a bottleneck due to thread two waiting on thread one to finish counting. It could cause a waste of resources since the thread has to wait. This scenario allows one thread to count up and down while the other handles another resource.

The simulated work with sleep intervals demonstrates how threads can execute concurrently. The sleep delay also lets the human eye see the computer counting and not printing out the statements almost simultaneously. Doing this is using up resources, which is not normally advised.

The attempt to handle thread sequencing by checking the counter, although this approach may not be foolproof. Using mutex should prevent the deadlock from happening. There is a chance that a condition could be forever ongoing. Locking that resource forever and not allowing other threads to use the resource in a timely fashion.

**Vulnerabilities Exhibited with the Use of Strings**

This code doesn’t involve string manipulation or user input, so it doesn’t exhibit vulnerabilities related to strings. If there were string input from users, we would need to sanitize it for security purposes.

**Security of the Data Types Exhibited**

Using an integer data type for the counter is appropriate for this scenario. The program uses a MAX\_COUNTER to ensure it remains within the upper bound. We correctly use mutexes for secure synchronization, ensuring data integrity during concurrent access.

**Conclusion**

The C++ application demonstrates fundamental concurrency concepts by creating two threads that count up and down concurrently while addressing performance considerations, security of data types, and mutex-based synchronization.

# **Pseudocode:**

Declare a mutex called mtx

Declare an integer variable called counter and initialize it to 0

Define the maximum count value as maxCount (20)

Function CountUp:

Lock the mutex mtx to protect the shared variable

Print “Thread 1: Count is “followed by the current value of counter

For increment from 1 to maxCount:

Simulate some work

Increment counter by 1

Print “Thread 1: Counting up -” followed by the updated value of counter

For End

Unlock the mutex mtx

Function End

Function CountDown:

Declare an integer variable troubleTracker and initialize it to 0

While counter is not equal to maxCount:

Increment troubleTracker by 1

Sleep for a while to give Thread 1 a chance

While End

If troubleTracker is greater than 11:

Print “Thread 2: Trouble! Counter is” followed by the current value of counter

Print “Thread 1: Has not finished counting up yet to 20!”

Exit the loop

If End

Lock the mutex mtx to protect the shared variable

Print “Thread 2: Count is “followed by the current value of counter

While true:

Simulate some work

If counter is greater than or equal to 1:

Decrement counter by 1

Print “Thread 2: Counting down -” followed by the updated value of counter

If End

If counter is equal to 0:

Exit the loop

If End

While End

Function End

Main Function:

Create a thread thread1 that executes the CountUp function

Create a thread thread2 that executes the CountDown function

Wait for thread1 to finish

Wait for thread2 to finish

Return 0

Main Function End

# **Figures Section**

## **Figure 1.**

Output of the Concurrency program Counting up then down with two different threads.

A screen shot of a computer program

Description automatically generated

## **Figure 2**

GitHub Repo

A screenshot of a computer program

Description automatically generated