**Module 7 Portfolio Project Part 1**

Alex Stampfl

Colorado State University Global

CSC450 Programming III

Farhad Bari

February 27th, 2025

**Analysis**

**Concurrency Performance Issues:**

My program is pretty simple as I’m using two threads sequentially, one after another. If, for example, I were to try running them simultaneously in parallel, it would indeed make the program run faster. However, unless there’s a real reason to do so, it’s better to keep things simple. If I try to get fancy and run two threads simultaneously, I might run into issues such as deadlocks, where two threads are stuck waiting for each other. I could also encounter data races, where two threads share the same memory, leading to improper synchronization.

**Vulnerabilities with Strings:**

In my code, I’m not using any string data types, so this isn’t an issue for me. However, if I were to use strings, I would need to ensure I’m either using `std::string`, which automatically includes the null terminator, or using a `char[]` array, and manually including the null terminator. Failing to handle strings properly could lead to buffer overflows or memory corruption, especially if the strings are used in user input handling.

**Data Types Security:**

As I mentioned above, I would need to be careful with using strings. Another data type vulnerability is with integers. One example of an integer data type issue is the integer overflow, which occurs when a value exceeds the maximum storage capacity of the data type. Similarly, unsigned integers can cause unexpected behavior if negative values are mistakenly assigned. My program is very simple, so there is no need for immediate need for additional data types. However, if I were to include some additional data types, I would need to ensure they are secure and that there is enough memory space allocated for each of them.

**Pseudocode**

1. Start with basic setup (create main() method)
2. Declare the two methods that do the actual iterations of the threads
3. In the main method, create the two threads. Create the 1st thread and employ the `.join` to wait until thread1 is complete. Create the 2nd thread, and do the same thing, use the join() to wait until it is complete
4. The join() ensures that the thread output is controlled and sequential. Do not start the next thread until the previous thread is fully complete.
5. End main method
6. In the 1st method, `countUp` (thread1), use a for-loop to iterate and display the numbers 1-20.
7. In the 2nd method, countdown(thread2), use a for-loop to iterate and display the counting down of the numbers 20-1

**Source Code**

#include <iostream>

#include <thread>

using namespace std;

// Function declarations

void countUp();

void countDown();

int main() {

    cout << "\n";

    // Create thread #1

    thread thread1(countUp);

    thread1.join(); // wait till thread1 is done to start thread2

    // Create thread #2

    thread thread2(countDown);

    thread2.join();

    cout << "\n-- Threads have finished execution --\n";

    return 0;

}

void countUp() {

    // Thread #1 - Count up to 20

    cout << "\*\* Thread #1 has started \*\*\n";

    for (int i = 1; i <= 20; i++) {

        cout << "CountUp: " << i << endl;

    }

    cout << "\*\* Thread #1 has ended \*\*\n\n";

}

void countDown() {

    // Thread #2 - Count down to 0

    cout << "\*\* Thread #2 has started \*\*\n";

    for (int i = 20; i >= 1; --i) {

        cout << "CountDown: " << i << endl;

    }

    cout << "\*\* Thread #2 has ended \*\*\n";

}

**Screenshots (output)**

A screen shot of a computer program

AI-generated content may be incorrect.

**Results (Output)**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

\*I observed something interesting: when I adjust the size of the terminal window in VSC, it would alter the output, and the output would be random and chaotic even after the code had terminated.

**Git Repository**

A screenshot of a computer

AI-generated content may be incorrect.

**References**

*Concurrency in C++*. (2023, December 9). GeeksforGeeks. <https://www.geeksforgeeks.org/cpp-concurrency/>

Ricciardi, A. S. (2024, November 10). *Concurrency in C++: Mitigating Risks - Level Up Coding*. Medium; Level Up Coding. <https://levelup.gitconnected.com/concurrency-in-c-mitigating-risks-5891fba59842>