**Portfolio Project: Part 1**

Jeremy Matthews

Colorado State University Global

CSC450 – Programming III

Dr. Reginald Haseltine

Due Date: 07/28/2024

**Portfolio Project: Part 1**

Concurrency in programming is when the program is executing multiple sequences of operations at the same time to improve the performance and responsiveness of the application. This can be a great thing for our programs if done correctly, but can cause issues if implemented poorly. The performance of our applications can be impacted by the overhead of the threads and the synchronization, as well as the context switching involved. Vulnerabilities can also become present when we use strings in our application, which can allow buffer overflows and race conditions if they aren’t handled correctly. So, it is important to take care when designing our programs to make sure that the security of our data types is done correctly in a concurrent environment. To control thread operations and protect the shared resources we can employ mechanisms, such as mutexes and condition variables to help ensure our programs run smoothly in a concurrent environment.

**Performance Issues with Concurrency**

When dealing with concurrency we must make sure that we design our programs in a way that everything works together to provide a smooth experience for the users. Issues like the overhead involved with creating and managing threads and using mutexes and condition variables to synchronize them can cause our programs to perform poorly if managed incorrectly. Frequent context switching can also lead to performance degradation. So it is important to implement these aspects of our programs in a correct way to ensure a smooth experience for our users.

**Thread Overhead**

Creating and managing threads produces overhead because each thread requires its own stack space and the system needs to manage context switching between the threads. This overhead can cause the performance of our programs to degrade if not managed properly. Threads carry their own set of resources, such as register states and stack memory, which adds to the overhead (CodeGuru, n.d.). Additionally. Some systems handle thread creation more efficiently than others; for instance, VMS (Virtual Memory System) is known to be less efficient in this regard when compared to other systems, like Linux, which goes to show that some systems perform these types of operations better than others (Stack Overflow, n.d.).

**Synchronization Overhead**

Using mutexes and condition variables to synchronize threads can create performance bottlenecks by causing the threads to have to wait for locks to become available, which can make our programs less efficient. It is best practice to make sure that we optimize the performance of single threads before introducing multithreading into our applications to help avoid unnecessary synchronization overhead (Stack Overflow, n.d.a).

**Context Switching**

Switching contexts frequently between multiple threads can cause performance degradation, especially if the threads perform minimal work before switching. So, it is important to minimize the number of times that this switching happens, and maximize the amount of work that happens in between switches. Having too many threads in the thread pool can also lead to performance decreases, as additional threads can lead to idle states and unnecessary context switches (Stack Overflow, n.d.b).

**Vulnerabilities Exhibited with Using Strings**

**String Handling**

Using strings in your applications can introduce vulnerabilities such as buffer overflows if they aren’t handled correctly. In a multithreaded environment, improper string handling can lead to data corruption caused by the need for synchronized access to dynamic memory, which creates overhead (Stack Overflow, n.d.a).

**Concurrency Issues with Strings**

If two or more threads access and modify the same string without proper synchronization, it can lead to race conditions and data corruption. Using thread synchronization mechanisms like mutexes are vital to make sure that strings are manipulated safely in concurrent environments.

**Security of the Data Types Exhibited**

**Primitive Data Types**

The application primarily uses primitive data types like integers, which are generally safe in a concurrent environment if proper synchronization is used. However, if shared primitive variables are used without synchronization, it can lead to race conditions. To prevent this a lock can be used before accessing shared resources in order to prevent unexpected behavior (Stack Overflow, n.d.)

**Thread Safely**

The use of **std::mutex** and **std::condition\_variable** makes sure that threads operate in a controlled manner, which prevents data races. Efficient use of these primitives can help to maintain thread safety without significant performance penalties (Stack Overflow, n.d.b).

**Data Integrity**

When dealing with concurrency in our applications we have to carefully design them so that the integrity of the data is ensured and all race conditions are avoided and the data is updated consistently. For example, in my program I used a boolean flag that is protected by a mutex to help make sure that the second thread only starts after the first thread finishes counting up, which maintains the data integrity.

**Conclusion**

Concurrency can significantly improve the performance and responsiveness of an application by allowing multiple operations to run in parallel. Although it also introduces complexities such as performance overhead from context switching and synchronization, and vulnerabilities related to data sharing and race conditions. Proper design, including careful use of synchronization mechanisms and attention to data safety, is essential when building concurrent applications.

**References**

Code Guru (2007 April 6) Why Too Many Threads Hurts Performance, and What To Do About It. Code Guru.  
<https://www.codeguru.com/cplusplus/why-too-many-threads-hurts-performance-and-what-to-do-about-it/>

Stack Overflow (2010 October) How Much Overhead is There When Creating a Thread?. Stack Overflow.  
<https://stackoverflow.com/questions/3929774/how-much-overhead-is-there-when-creating-a-thread>

Stack Overflow (2018 June, a) Why is the C++ thread/future Overhead So Big?. Stack Overflow.  
<https://stackoverflow.com/questions/50636652/why-is-the-c-thread-future-overhead-so-big>

Stack Overflow (2015 October, b) What is the Performance Impact of Having More OpenMP Threads Than Work?. Stack Overflow  
<https://stackoverflow.com/questions/33129488/what-is-the-performance-impact-of-having-more-openmp-threads-than-work>