Justin McCartney

Colorado State University - Global

CSC 450-1

Professor Haseltine

October 6, 2024

Program Analysis

Performance Issues

Thread Overhead

In Java, the creation of threads has apparent overhead, similar to creating threads in C++. In Java, an abstraction is provided over system threads, which generally means that the creation of threads can be more resource-consuming compared to C++ thread creation. This is due to JVM, or *Java Virtual Machine*, which is the core of the Java ecosystem. This allows for the *write once, run anywhere* approach in Java programming. That being said, embedded in this ecosystem is also a concurrency package, which allows for optimization of multithreading.

Context Switching

Similar to C++, Java also struggles with the performance impact of context switching between threads. The thread scheduling system which Java uses typically operates within the JVM, and generally relies on the native devices' operating system thread scheduler. When context switching is involved in any operation, there is a deal of overhead introduced while multiple threads are active.

Thread Synchronization

The synchronization process is a contributor to overhead costs itself. In this particular example, the *Thread.join()* syntax is used to ensure that the second thread does not begin until the first thread has completely finished. Without this level of synchronization, both threads may end up running concurrently. For this example, both threads running concurrently would create an issue where both counters are *counteracting* one another, creating an infinite state.

Garbage Collection (JVM)

Once again, Java's garbage collection system also adds overhead when managing memory for threads. If the garbage collection were to kick in before, or during thread execution,

it may pause other threads. When other threads are paused, this will generate some issues regarding overall performance. That being said, when working with modern JVM architecture and implementations, this issue is significantly minimized.