|  |
| --- |
| Comparison of C++ and Java Concurrency Programs |

The programs in C++ and Java were intended to accomplish the same task. We needed to have a program that created two threads. The first thread should count up to 20, while the second thread should count down to 0 only after the first thread has completed counting to 20. Both will require a similar solution. We will also determine which one is the better solution of the two.

The C++ version of the concurrency application consists of 2 functions that are invoked form 2 threads in main, one function counts up to 20, and the other counts down once the first is executed. Due to the sequential nature of this application we will encounter performance issues with concurrency since we are consuming more of the CPU’s cores to execute code that could be executed from a single thread using a single core. In this application there are no vulnerabilities exhibited with use of strings because we only use hard coded strings in the applications output. We do not accept any inputs, so there is no need to sanitize the inputs of this application. The primitive data types we use are an int. Since we only set the value within the range 0 → 20 which will never surpass the limits of an int value.(hard to be specific since the range varies depending on C++ compiler.) We also use thread. Threads pose the risk of issues due to concurrent code execution like race conditions or unpredictable output if some sequence of events are expected to occur. This can pose a security risk if a user is able to take advantage of this and exploit it to bypass a step in sequence. This code is a great example of this because increasing the max number can exhibit output that is out of sequence and strings will get mutilated in the process. This can be solved by making the functions atomic by using mutexes and semaphores to block the other thread from executing before the first thread is done with that task.

The Java program is sequential in nature and therefore does not need to be executed on multiple threads. This will cause the JVM to hog up more CPU on the users device un-neccessarily meaning this program is highly inefficient. Therefor it would make more sense to execute the logic on of this application from a single thread. There are no vulnerabilities exhibited with the use of strings in this application. The reason being all strings are hard coded, and have no easy means to manipulate. The data type we use is an int. It is iterated through the means of a for loop, one incrementing and the other decrementing. This program accepts no inputs and only writes the count to the std out. Therefor its data is not vulnerable to user input, and it would be impractical to attack this program. We also create two threads to execute each count thread. The iteration of the for loops in each thread use locks to synchronize the execution of the counts, so there are no issues with concurrency in this application.

Both programs are sequential in nature, and could have been solved using a single thread. Both programs also only have int data types, and all strings are static in nature, so do not pose much of a threat in this scenario. Both programs also create two threads with for loops to count up and down. Neither program has any inputs and only print to Std out, so cannot be manipulated into miss behaving via IO.

The question still remains, which program is superior to the other. A major difference between the Java and C++ implementations is that the C++ implementation runs as a native binary, and is compiled specific to the system it will run on while the Java implementation compiles down into Java byte code, which is interpreted by the JVM. With this fact in mind it is easy to conclude that the C++ implementation has the potential to be more efficient since it does not require a virtual machine to run. However, the JVM provides the Java application with many advantages. The first being potability. The compiled Java byte code can be ran on any system that has a JVM installed. The second being security. The JVM helps to enforce security of the application. While it is possible to find and exploit a vulnerability at the interpreter level, this would an issue with the interpreter itself and not the code. The JVM will also manage memory for the programmer, so will pose less risk of a programmer creating a vulnerability due to a resource leak.

With these points in mind, the question of which program is better really depends on the implementation that the developer came up with. In the case of these programs, the C++ program poses a major flaw. The C++ program has an issue with synchronization, due to not using mutexes or semaphores, while the Java implementation is automic and will not concurrently count down before beginning to count up. With this being the case, the Java version of this code is more reliable, and would be the better choice since the C++ program is not guaranteed to function as intended.