Security and Concurrency in Java

Stephan Peters

Colorado State University Global

CSC450-1 23WD: Programming III

Reginald Haseltine

5 April 2024

Security and Concurrency in Java

This paper will briefly cover security concepts introduced throughout the course and will demonstrate a Java program that utilizes threads sequentially rather than concurrently. Two topics that will be covered include performance issues with concurrency and vulnerabilities exhibited with use of strings. Finally, I will discuss vulnerabilities with the data types used in the application.

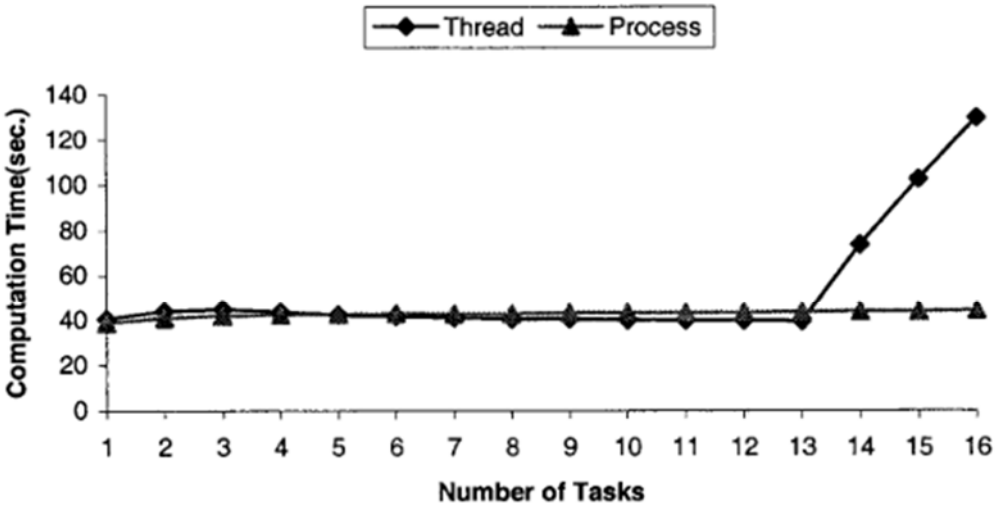
Performance Issues with Concurrency

Concurrency refers to executing several computations simultaneously. Running multiple simultaneous threads is a simple task if they do not interact with mutable shared objects. Concurrency can allow your application to accomplish more work in less time, though this is not always the case. Outside of the normal issues that may be caused from incorrectly programming for concurrency, such as unsynchronized collections, race conditions, memory inconsistencies (especially stemming from use of memory caches by the OS), non-atomic variables, deadlock, etc. may actually decrease the performance of your application (see Burcea, 2019).

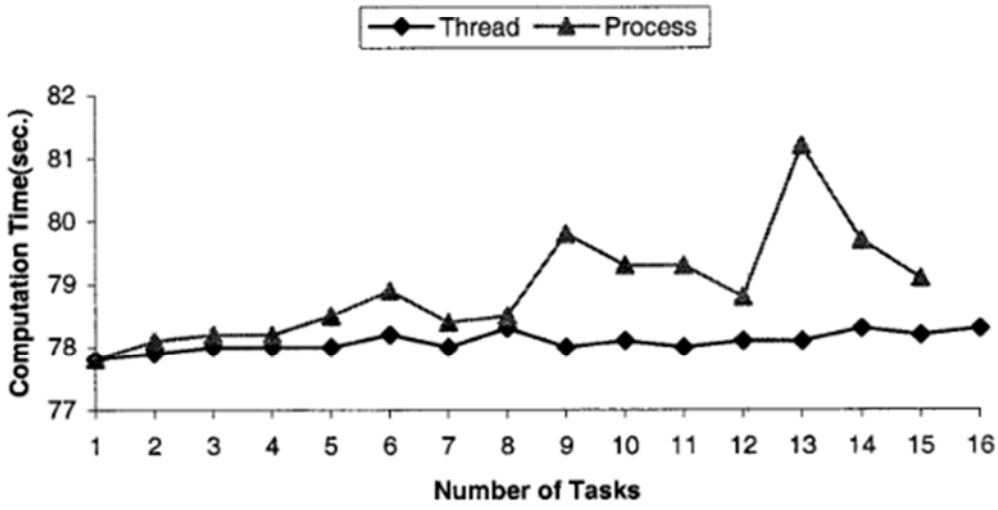
In a study performed in 1999 by Yan Gu, B.S. Lee and Wentong Cai entitled *Evaluation of Java thread performance on two different multithreaded kernels*, they found processing time actually increases using Java threads under Windows NT (1999). It must be noted this experiment was performed on much older computers and operating systems than are available now. But it still emphasizes the need to benchmark under expected load with similar hardware to what is expected to determine which methods of multithreading under Java will increase computation time rather than increase performance. Figure 1 shows some of the results of their experiments.

Figure 1

Performance of a multithreaded Java Application under light load, Windows NT vs. Sun Solaris (Gu et al., 1999)



(a) EP Computation Time under Light Load on Windows NT



(b) EP Computation Time under Light Load on Solaris

Figure 4: EP Computation Time under Light Load

While this study is primarily historical, the need to study different methods of multithreading in a modern setting with multiple multi-core processors as the norm still should be performed (See Sartor & Eeckhout, 2012).

Vulnerabilities in Strings

While Java does not have the egregious vulnerabilities caused by "C-Style" strings that may be found in C/C++ programs, there are still string vulnerabilities that exist within Java. One common vulnerability with strings in Java is the String concatenation vulnerability, where a string is concatenated with another string that contains user input. User input can contain malicious code. If a user-entered parameter contains malicious code, it can be concatenated with a mutable String variable, potentially executing malicious code. Another common string vulnerability is the reflection vulnerability. This is caused by using the reflection API to access private or protected fields of a string. For example:

```
String str = "Hello World";
Field field = str.getClass().getDeclaredField("value");
field.setAccessible(true);
String value = (String) field.get(str);
```

In this example, the reflection API is used to access the value field of the string. This field is protected, so it can only be accessed by code in the same package. However, if the code is malicious, it can access this field and potentially read or modify sensitive data.

Other vulnerabilities with strings in Java include (but are not limited to) SQL injection, XSS (cross-site scripting), LDAP injection, and Cross-Site Request Forgery (CSRF) (Dannarapu, 2023).

Vulnerability of Data Types in the Program

One possible vulnerability of this Java program is in the Count class, as the integer input is not validated. As the program stands, this is not an issue, but were this program to go into production for some reason, validating and sanitizing the integer input from a calling class would be a requirement. This includes validating the minimum is less than the maximum, and the interval is less than the minimum subtracted from the maximum.

Conclusion

Java is less likely to suffer from performance issues due to using concurrency for small operations than some other languages, but load testing may still be required to determine if there are areas that may be slower using concurrent threads over processes in today's multi-core / multi-CPU environment. Strings in Java may become vulnerable when mixing them with user entered content, or when using the reflection API. Any mutable object that includes user entered content should be validated and sanitized before introducing the content to an application.

References

Burcea, C. (2019, November 23). *Common Concurrency Pitfalls in Java* (A. Frieze, Ed.). Baeldung. https://www.baeldung.com/java-common-concurrency-pitfalls

Dannarapu, S. (2023, April 9). *Security Vulnerabilities in Java application*. Javarevisited. https://medium.com/javarevisited/security-vulnerabilities-in-java-application-e844bd281ff2

Gu, Y., Lee, B. S., & Cai, W. (1999). Evaluation of Java thread performance on two different multithreaded kernels. *ACM SIGOPS Operating Systems Review*, *33*(1), 34–46. https://doi.org/10.1145/309829.309841

Sartor, J. B., & Eeckhout, L. (2012). Exploring multi-threaded Java application performance on multicore hardware. *Ghent University Academic Bibliography (Ghent University)*, 281–296. https://doi.org/10.1145/2384616.2384638