The goal of this research is to compare linear execution, multithreading, and multiprocessing across different programming languages, different operating systems, and different systems. The programming languages used for comparison are Java and Python. In Java, only linear execution and multithreading are tested. Windows 10, Windows 11, and Raspbian Linux are the three operating systems used for comparison. The systems used in this research are Device 1: Dell Inspiron 16 Plus (Intel Core i7-11800H 24MB Cache, up to 4.6 GHz, 8 cores) (Windows 11) ​, Device 2: desktop PC (Intel Core i7-127700F 2.10 GHz) (Windows 10)​, Device 3: HP Pavilion Notebook (Intel Core i7-6500U 2.5 GHz) (Windows 10)​, Device 4: Raspberry Pi 4B (1.5 GHz) (Linux)​, and Device 5: Raspberry Pi 400 (1.8 GHZ) (Linux)​. Three random number generating algorithms are used in this research as well, each within their own thread when testing multithreading and multiprocessing. The three random number generating algorithms are the linear congruential, lagged Fibonacci, and the middle square algorithms. With linear execution, each random number generator executes one at a time. With multithreading, each random number generator runs within its own thread and each thread runs on one core. With multiprocessing, each random number generator runs with its own thread and each thread runs on a separate core. Our experiment uses three different PRNG algorithms each generating 10 million numbers each. The timed data result is the sum time of all three methods; The displayed result is the average time after fifty runs of the program. The memory usage graphs display usage over one run of the program. The results of this experiment are visualized in the following graphs:

Chart, bar chart
shows linear time is 1.887 seconds and Threaded time in 1.476 seconds.Device 1 data:

Chart, bar chart

Description automatically generated This graph shows that when using Java, it takes an average of 1.887 seconds when running the program linearly and an average of 1.476 seconds when implementing threads.

This graph shows that when using Python, it takes an average of 6.671 seconds when running the program linearly and an average of 6.322 seconds when implementing threads, and an average of 4.112 seconds with multiprocessing.

On this device, Java is 3.54x faster than Python linearly. When using Threading, Java is 4.28x faster than Python, and Java threading is also 2.79x faster than Pythons multiprocessing.

Chart, line chart

Description automatically generatedThis graph shows that Java linear memory maxes out just under 1600 MiB while threading uses slightly more, maxing out 1600.

Chart, line chart

Description automatically generated

This graph shows that Python is most efficient when using linear processing using about 250Mib during peak times. Threading is significantly more costly at just under 600 MiB, and multiprocessing has about the same peak use but is also more time efficient.

Chart, bar chart

Description automatically generatedDevice 2 data:

Chart, bar chart

Description automatically generated This graph shows that when using Java, it takes an average of 1.344 seconds when running the program linearly and an average of 0.953 seconds when implementing threads.

This graph shows that when using Python, it takes an average of 7.269 seconds when running the program linearly and an average of 7.229 seconds when implementing threads, and an average of 4.572 seconds with multiprocessing.

On this device, Java is 5.41x faster than Python linearly. When using Threading, Java is 7.59x faster than Python, and Java threading is also 4.8x faster than Pythons multiprocessing.

Chart, line chart

Description automatically generatedThis graph shows that Java linear memory maxes out just Over 1400 MiB while threading uses slightly less, maxing out just under 1400.

Chart, line chart

Description automatically generated

This graph shows that Python is most efficient when using linear processing using about 300Mib during peak times. Threading is more costly at just over 400 MiB, and multiprocessing Peaks a little under 500MiB but is also more time efficient.

Chart, bar chart

Description automatically generatedDevice 3 data:

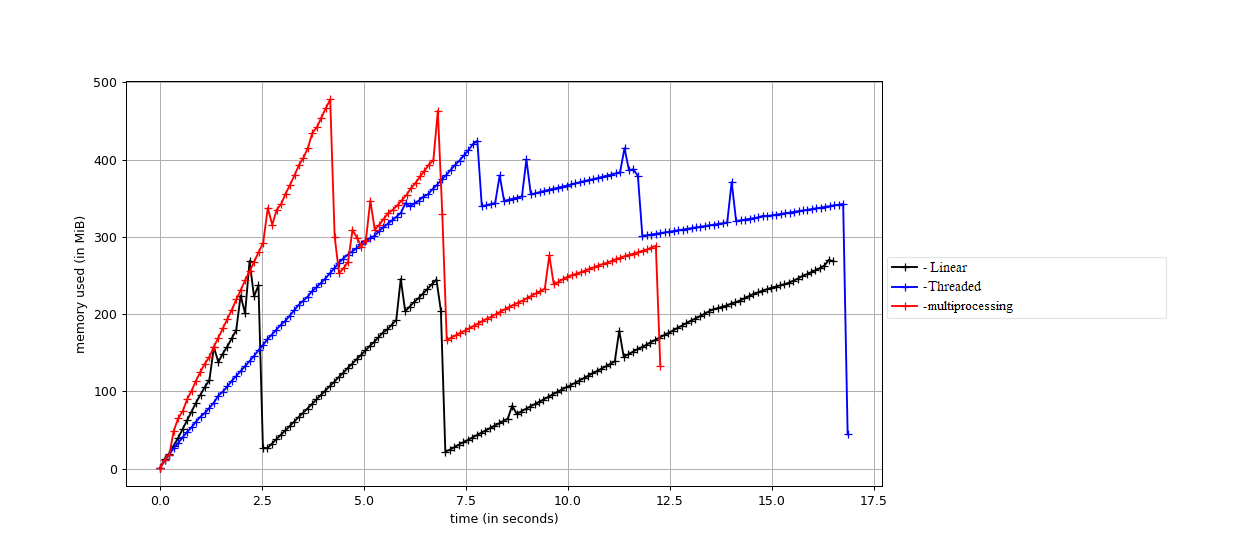
Chart, bar chart

Description automatically generatedThis graph shows that when using Java, it takes an average of 5.374 seconds when running the program linearly and an average of 4.49 seconds when implementing threads.

This graph shows that when using Python, it takes an average of 18.056 seconds when running the program linearly and an average of 17.967 seconds when implementing threads, and an average of 12.277 seconds with multiprocessing.

On this device, Java is 3.36x faster than Python linearly. When using Threading, Java is 4.00x faster than Python, and Java threading is also 2.73x faster than Pythons multiprocessing.

Chart, line chart

Description automatically generatedThis graph shows that Java linear memory maxes out at about 1300 MiB while threading is slightly more memory efficient, maxing out at just under 1200.

This graph shows that Python is most efficient when using linear processing using under 300Mib during peak times. Threading is more costly at Just over 400 MiB, and multiprocessing is the most costly at almost 500Mib peak use but is also more time efficient.

Chart, bar chart

Description automatically generatedDevice 4 data:

Chart, bar chart

Description automatically generated This graph shows that when using Java, it takes an average of 16.094 seconds when running the program linearly and an average of 12.75 seconds when implementing threads.

This graph shows that when using Python, it takes an average of 48.396 seconds when running the program linearly and an average of 51.266 seconds when implementing threads, and an average of 32.157 seconds with multiprocessing.

On this device, Java is 3.01x faster than Python linearly. When using Threading, Java is 4.02x faster than Python, and Java threading is also 2.52x faster than Pythons multiprocessing.

Chart, line chart

Description automatically generatedThis graph shows that Java linear memory maxes out just Under 400 MiB while threading uses significantly more, maxing out just under 800MiB.

Chart, line chart

Description automatically generated

This graph shows that Python is most efficient when using linear processing using about 350Mib during peak times. Threading is much more costly at 700 MiB, and multiprocessing takes even more at 800MiB but is also more time efficient.

Chart, bar chart

Description automatically generatedDevice 5 data:

Chart, bar chart

Description automatically generated This graph shows that when using Java, it takes an average of 14.277 seconds when running the program linearly and an average of 11.375 seconds when implementing threads.

This graph shows that when using Python, it takes an average of 41.165 seconds when running the program linearly and an average of 43.389 seconds when implementing threads, and an average of 26.802 seconds with multiprocessing.

On this device, Java is 2.88x faster than Python linearly. When using Threading, Java is 3.81x faster than Python, and Java threading is also 2.36x faster than Pythons multiprocessing.

Chart

Description automatically generatedThis graph shows that Java linear memory maxes out just over 300 MiB while threading uses significantly more, maxing out at 700MiB.

Chart, line chart

Description automatically generated

This graph shows that Python is most efficient when using linear processing using about 150Mib during peak times. Threading is more costly at about 225MiB, and multiprocessing takes even more at 275MiB but is also more time efficient.

Summary:

On average, Java code is about 3x faster than Python when running the PRNG functions linearly. Device two is an outlier as a custom-built PC and runs the Java code 5x faster. When running each PRNG function in its own thread, the Java code is about 4x faster than python threads. However, when using python multiprocessing which is more comparable to how java threading works, Java’s advantage is a bit smaller, averaging about 2.6x faster. Device two runs java threads 7.59x faster than Python threads and 4.8x faster than Python multiprocessing.

When looking at the relationship between time and memory usage In general the more memory being use the faster the program completes. The outlier to this is the Raspbery Pi 4B is slow at floting point operations and the operations bottle neck causing slow times with high usage for threaded and multiprocessing operations.

References

Delporte, F. (2021, December 10). *Getting started with Java 17 on the Raspberry Pi: Foojay.io Today*. foojay. Retrieved 2022, from https://foojay.io/today/java-17-on-the-raspberry-pi/

*How to calculate memory usage of a Java program?* Stack Overflow. (1963, August 1). Retrieved 2022, from https://stackoverflow.com/questions/37916136/how-to-calculate-memory-usage-of-a-java-program#:~:text=Calculate%20the%20memory%20used%20after%20your%20code%20execution%3A-,best%20bet%20is%20any%20profiler%20tool%20like%20jvisualvm

*How to do multiprocessing in Java, and what speed gains to expect?* Stack Overflow. (1958, December 1). Retrieved 2022, from https://stackoverflow.com/questions/8001966/how-to-do-multiprocessing-in-java-and-what-speed-gains-to-expect

*Memory-profiler*. PyPI. (n.d.). Retrieved 2022, from https://pypi.org/project/memory-profiler/

Real Python. (2021, May 15). *What is the python global interpreter lock (gil)?* Real Python. Retrieved 2022, from https://realpython.com/python-gil/

*Startup profiler*. VisualVM. (n.d.). Retrieved 2022, from https://visualvm.github.io/startupprofiler.html

Xu, L. (2019, April 20). *Summary of multithreading and multiprocessing in Java and python*. Just Chillin'. Retrieved 2022, from https://liyanxu.blog/2019/04/19/summary-of-multithreading-and-multiprocessing-in-java-python/

|  |  |
| --- | --- |
| Individual Times Each Member Spent Working on This Project | |
| Thomas Kyte | Michael Emory |
| Aprox. 35 hours | Aprox. 35 hours |