SEI Insights

SEI Blog

The Latest Research in Software Engineering and Cybersecurity

Multicore and Virtualization Recommendations

POSTED ON OCTOBER 30, 2017 BY **DONALD FIRESMITH** [/AUTHOR/DONALD-FIRESMITH] IN **MULTICORE PROCESSING AND VIRTUALIZATION** [HTTPS://INSIGHTS.SEI.CMU.EDU/SEI_BLOG/MULTICORE-PROCESSING-AND-VIRTUALIZATION/]

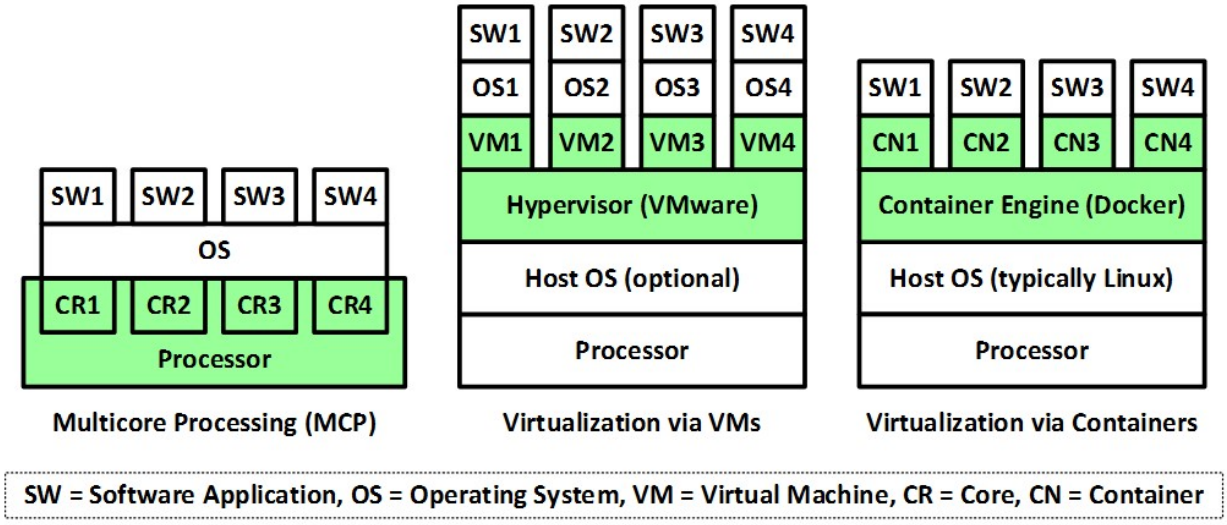
By Donald Firesmith
Principal Engineer
Software Solutions Division

The **first post** [https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-and-virtualization-an-introduction.html] in this series introduced the basic concepts of multicore processing and virtualization, highlighted their benefits, and outlined the challenges these technologies present. The **second post** [https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-processing.html] addressed multicore processing, whereas the **third** [https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-virtual-machines.html] and fourth posts concentrated on virtualization via virtual machines (VMs) and containers (containerization), respectively. This fifth and final post in the series provides general recommendations for the use of these three technologies--multicore processing, virtualization via VMs, and virtualization via containers--including mitigating their associated challenges.

The Three Technologies--A Brief Reminder

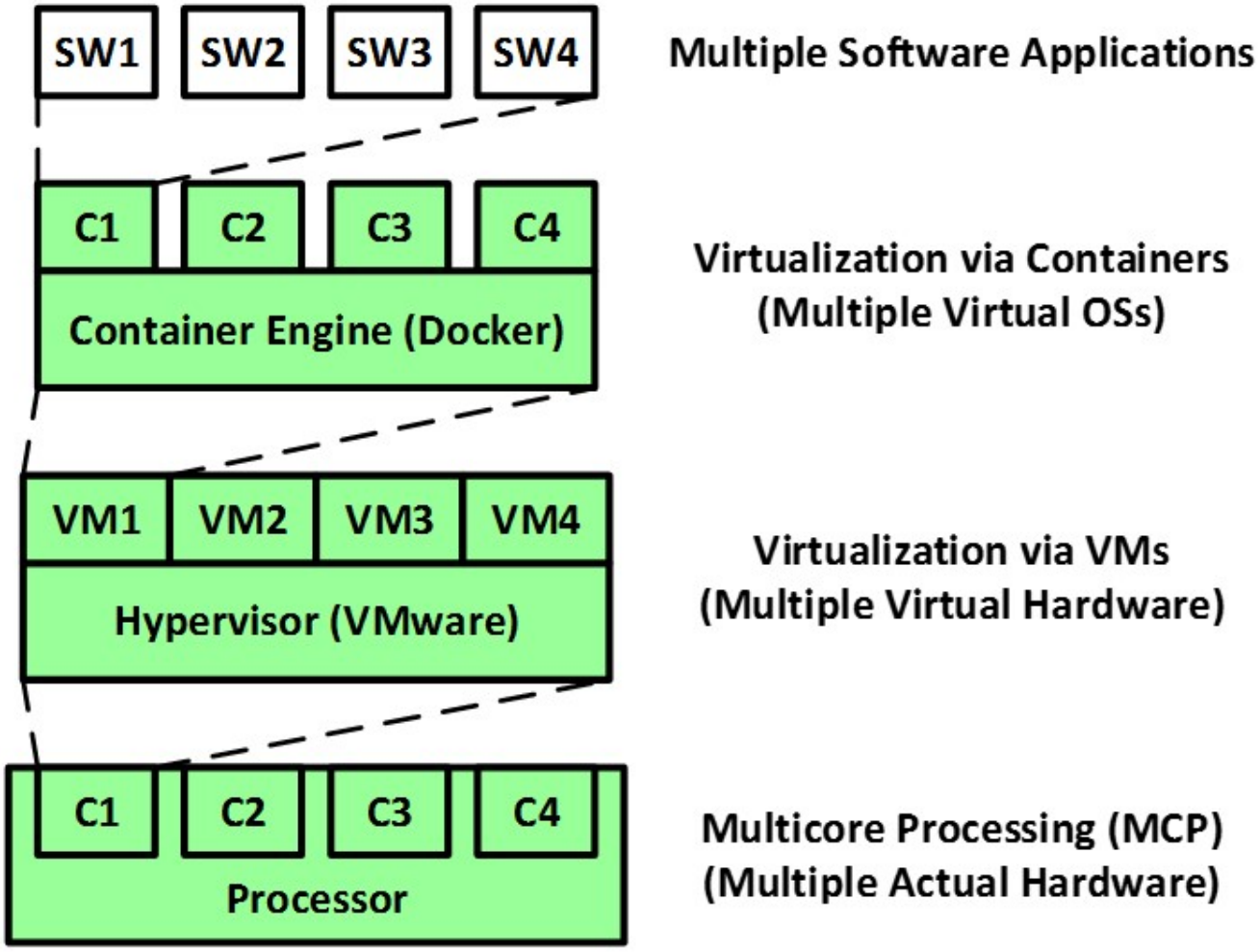
The following figure highlights the primary similarities of the three technologies. Each one provides a

separate real or virtual execution platform for different software applications, an environment that enables real or virtual concurrent execution of applications that are isolated both spatially and temporally from each other.



The following figure highlights the main differences among these three technologies, which exist in different layers of the architecture. Multicore processing provides multiple actual hardware environments for the software applications. On the other hand, virtualization via VMs provides multiple virtual hardware environments by emulating the actual underlying hardware. Finally, virtualization via containers provides virtual software environments.

As noted in posts **three** [https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-virtual-machines.html] and **four** [https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html] of this series, an additional significant difference between virtualization via VMs and virtualization via containers is that virtual machines are relatively heavyweight because they support different guest operating systems. In contrast, containers are relatively lightweight because they are implemented using only parts of the kernel of a single operating system (typically Linux).



While these three technologies can be used individually, they are increasingly being used together in hybrid architectures to achieve each technology's associated benefits, which were enumerated in the previous three blog posts. However, each technology also has its associated challenges. The remainder of this blog post provide general recommendations for using these technologies, including approaches to address and mitigate these challenges.

When to Use

Use *multicore processors* when regular hardware technology refreshes are important and when software can be decomposed to benefit from parallelism. Use *virtualization* when isolation and configurability (flexible deployment) are important. The following table lists criteria that can be used to decide whether to use virtualization via VMs and/or containers.

Criteria	VMs	Containers
Portability—number of operating systems	One or more	One
Portability—number of OS versions	One or more	One
Portability—number of OS types	One or more	Primarily Linux
Size of Applications	Medium or large	Small or medium
Security	More isolation	Less isolation
Number of applications per server	Lower	Higher
Number of copies of single application	One	Many
Performance (throughput, not response time)	Lower	Higher
Overhead—administration	Higher	Lower
Overhead—resource usage	Much higher	Much lower
Readily share resources (devices, services)	No	Yes
Robustness via failover and restart	Not supported	Supported
Scalability and load balancing via dynamic deployment	Slower and harder	Faster and easier
Application runs on bare metal	Not supported	May be supported

Allocation

Unless auto-scaling in dynamic cloud computing environments necessitates the dynamic allocation of VMs and containers to hardware, make this allocation static and minimize the use of hybrid virtualization (allocating containers to VMs) to simplify the architecture and reduce the number of test cases required.

Documentation

Document the deployment of application software to containers to virtual machines to processors to cores in the software architecture models and documentation. Specifically, document (where appropriate)

- software applications deployed to containers (if any)

- containers to container engine (if any)
- container engine to guest operating system (OS) or host OS
- containers or container engine to VM (if any)
- VM to hypervisor (if any)
- VM or hypervisor to host OS (if any)
- VM or host OS to core in a multicore processor
- multicore processors to computers (e.g., blades in racks) and processor-external shared resources
- containers and/or VM to data partitions in memory

Due to the complexity of deployment when using multicore processing and virtualization, deployment diagrams will likely be notional rather than specific. Therefore, consider using tables or simple relational databases supported by widely-available tools (e.g., Excel spreadsheets or Access databases). You should also automate the build and configuration process (e.g., via the associated tools and/or integration and configuration scripts or files).

Document the following VM-related information in the system and/or software architecture models and/or documentation: vendor and model number, hypervisor type, hypervisor configuration information, and (where appropriate) successful usage on real-time, safety-critical systems.

Similarly, document the following container-related information in the system and/or software architecture models and/or documentation: vendor and model number, container engine configuration information, and (where appropriate) successful usage on real-time, safety-critical systems.

Document the techniques used to eliminate, reduce, and mitigate significant container-, VM-, and core-interference penalties as well as associated non-deterministic behavior.

Document the analysis and testing performed to verify that performance deadlines are met, including analysis and test method(s) used, interference paths analyzed and path selection criteria, analysis and test results, and any known limitations of the analysis/test results.

Interference Analysis

Require the performance of multicore processing and virtualization interference analysis. The following process can be used to perform interference analysis:

1. Identify the relevant software (e.g., hard real-time and safety-critical) that can be significantly

impacted by interference.
2. Determine the deployment of relevant software to cores and data paths.
3. Adequately identify the processor's important interference paths
   based on the behavior of deployed software. Note that exhaustive identification and analysis is likely infeasible.
4. Determine the potential consequences based on lack of spatial isolation (e.g., race conditions involving data access) and lack of temporal isolation (e.g., due to the interference penalties of the important interference paths).
5. Categorize interference paths as acceptable or unacceptable (having either bound or unknown interference penalties).
6. Use interference elimination and mitigation (bounding and reducing) techniques for unacceptable interference paths.
7. Repeat steps 2 through 6 until all interface paths are acceptable.
8. Document the analysis results and limitations of the analysis.
9. Review the analysis results and associated limitations.
10. Use the analysis results when verifying performance (e.g., response time and throughput), reliability, robustness (e.g., fault and failure tolerance), and safety requirements.

Interference Analysis Challenges

Multicore processing and virtualization makes interference analysis significantly harder due to the

- increased number and type of shared resources where interference can take place (Note that the number of interference paths greatly increases with the number of components that can form the path.)
- corresponding increased number of interference paths
- increased length and complexity of the interference paths
- increased complexity of the multicore processors
- lack of architecture and design documentation of blackbox components (e.g., due to their proprietary nature)
- multiple "types" of interferences along same path based on the operations involved (e.g., read vs. write), the sequences of these operations, memory locations accessed, and CPU usage (stress)

Due to the large number of potential interference paths, it is important to identify a representative subset to analyze. Equivalence classes of paths and path redundancy can be used to limit the number of cases to analyze.

Interference Management Techniques

After interference paths and significant sources of interference are identified, the interference needs to be eliminated, reduced, or at least bounded. Techniques for eliminating, bounding, and reducing interference include

- allocation of software to cores
- interference-related fault, failure, and/or health monitoring (e.g., performance, missed deadlines) backed up by exception handling
- configuration of the hardware and operating system
- cache coloring or partitioning to reduce cache conflicts
- interference-free scheduling (only one critical task per timeslot)
- deterministic execution scheduling (tasks accessing the same shared resource execute in different timeslots)

There is no single silver bullet when it comes to managing interference. Multiple techniques are typically required.

Some interference-management techniques are completely up to the software architect to implement. Other techniques require the software architect to configure the virtualization software. Finally, some techniques require selection and implementation by the processor vendor.

Interference analysis alone is typically too hard and incomplete to be relied on. Interference analysis should be augmented with a significant amount of relevant types of testing, expert feedback on the multicore processor and virtualization technologies, and information shared by the manufacturer or vendor.

Testing

Multicore and virtualization defects are rare and non-deterministic. Testing, therefore, typically requires the creation of very large suites of test cases to uncover rare faults and failures. Use long-duration **soak testing** [https://en.wikipedia.org/wiki/Soak_testing], **modeling and simulation (M&S)** [https://en.wikipedia.org/wiki/Modeling_and_simulation], and **combinatorial testing** [http://mse.isri.cmu.edu/software-engineering/documents/faculty-publications/miranda/kuhnintroductioncombinatorialtesting.pdf]
to achieve adequate coverage of combinations of inputs and conditions and to adequately address edge and corner cases. Because these defects occur stochastically, probability and statistics are required to determine the minimum number of test cases required to achieve a desired level of

confidence that the system and software have been adequately tested.

Clearly simple demonstrations of functional requirements (with only a few **sunny day test cases** [https://www.faa.gov/.../common_test_problems_presentation_2012-10-11.ppsx]) will be insufficient to identify defects related to multicore processing and virtualization, especially those resulting in performance, reliability, robustness, and safety faults and failures. Unit testing and low-level integration testing are also unlikely to uncover these defects. Instead, the testing will require that the system or at least major subsystems be integrated. Final testing will also require a high-fidelity test environment that is properly built and configured.

To uncover relevant defects after so much integration has occurred, adequate testability (the combination of observability and controllability) will often require instrumenting the software so that logs can be scrutinized for rare timing and other anomalies. You should also consider incorporating **built-in-test (BIT)** [https://en.wikipedia.org/wiki/Built-in_self-test], especially continuous BIT (CBIT), interrupt-driven BIT (IBIT), and periodic BIT (PBIT).

Finally, because it is hard and expensive to uncover multicore- and virtualization-related defects, you should probably concentrate your limited testing resources on testing the most important software (e.g., mission- and safety-critical software).

Safety

Software safety standards, policies, and **accreditation and certification (C&A)** [https://www.nist.gov/nvlap/accreditation-vs-certification] often assume traditional architectures. Therefore, it is necessary to tailor them for multicore processing and virtualized architectures. This tailoring includes new architecture documentation, analysis, and testing requirements and guidelines.

When categorizing real-time software as safety-critical and safety-relevant, take the impact of multicore processing and virtualization into account. Take associated potential faults and failures into account when performing safety (hazard) analysis.

Conclusion

Multicore processing and virtualization via VMs and containers are quite similar and subject to similar challenges. Many recommendations for addressing these challenges are applicable to all three technologies. Although all three technologies improve (but do not guarantee) spatial and temporal isolation, they also involve shared resources that are the source of both spatial and

temporal interference. Analyzing this interference is hard and cannot be exhaustive. Analysis must therefore be augmented with large amounts of specialized testing backed up by expert opinion and past experience with the technologies. Safety standards, guidelines, policies, and accreditation and/or certification processes will likely require tailoring.

Additional Resources

Read earlier posts in the **multicore processing and virtualization** [https://insights.sei.cmu.edu/sei_blog/multicore-processing-and-virtualization/] series.

Read **all blog posts by Don Firesmith** [http://insights.sei.cmu.edu/author/donald-firesmith/].

About the Author

Donald Firesmith

Contact Donald Firesmith [https://www.sei.cmu.edu/contact.cfm]
Visit the SEI Digital Library for **other publications by Donald** [http://resources.sei.cmu.edu/library/author.cfm?authorID=4637]
View **other blog posts by Donald Firesmith** [/author/donald-firesmith]