Virtualization for embedded industrial systems

Texas Instruments

Ellen Kou
Product marketing manager
Texas Instruments

Introduction

"Virtualization" is a widely used term that encompasses a wide range of technologies and can mean very different things in different contexts. A virtualized cloud system will take on vastly different tasks than a virtualized enterprise system and does things that are even further removed from what a virtualized embedded system would do.

Although methods and end applications for different types of virtualization differ, they do have one thing in common: taking a bounded set of physical hardware and making that set act like multiple virtual environments by creating partitions inside the hardware. Virtualization can work at a server level, at a single platform level or at a network level. It allows you to run multiple sessions for multiple people or tasks on the same server simultaneously (as cloud computing applications do), or to separate a single desktop into two virtual machines (such as when you use a laptop to run both Windows® and Linux®). This separation of tasks and extension of resources can be valuable for many different end uses, but different virtualization schemes will not all work equally within the constraints of embedded industrial systems, which include limited memory or limited power consumption caused by heat dissipation.

Embedded systems also have different priorities than enterprise or data center server infrastructure systems. Whereas enterprise systems may prioritize meeting metrics such as average throughput or transactions per second, embedded systems prioritize real-time operation. For many embedded systems, especially in industrial automation, meeting a deadline is part of the correct operation of the program; this determinism cannot be sacrificed to achieve other goals.

Real-time operating systems (RTOSs) and bare-metal-based periodic control loops are common software architectures that have been in place in industrial automation for years. Recent market trends are forcing industrial manufacturers to look at other solutions in order to incorporate more non-real-time functionality around the real-time systems and accomplish goals such as cloud connectivity for uploading machine data to perform predictive maintenance—a commonly discussed application around Industry 4.0 or the Industrial Internet of Things (IIoT). System designers need a solution to run these new tasks without compromising or interfering with their critical real-time tasks. Virtualization seems to be one path forward, but with the multiple types of virtualization solutions out there, deciding which one to use requires in-depth discussion and thought.

Types of virtualization and suitability for embedded systems

Virtualization can take two paths: full virtualization or static partitioning, also called core virtualization.

Full virtualization schemes simulate the hardware environment so that the software partitions creating virtual environments are not tied directly to the hardware partitions. For example, you could use one physical core as two virtual cores running two different OSs.

Static partitioning schemes isolate programs or tasks to certain portions of the existing hardware in order to simulate separate systems. The software partitions are bound to the hardware partitions—so you would have at most one OS per physical core, as shown in **Figure 1** below, and no support for over commitment of resources like central processing units (CPUs) or random access memory (RAM).

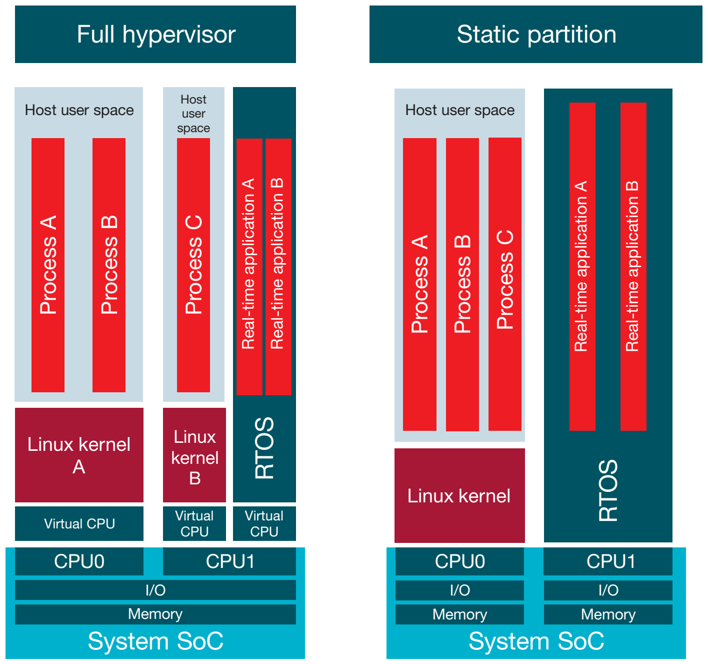

Figure 1: *Comparison of the two main types of virtualization.*

Both schemes enable the separation of isolated tasks (such as sharing a system among multiple users) or the separation of critical tasks from less-critical tasks (such as separating a secure domain from a general-purpose domain). Full virtualization further enables you to suspend entire OSs to persistent storage or even live migration from one physical processor to another over a network connection. Static partitioning trades off this flexibility for some guarantees of determinism.

Since the different virtualization schemes relate to hardware in different ways, each naturally has advantages for certain applications. Full virtualization is powerful; it can enable a single server to act like hundreds of servers. It does have drawbacks, however, since multiple software instances may attempt to use the same finite set of hardware resources simultaneously. Having the software manage the virtualization scheme can resolve conflicts, but there will be an impact on latency.

Another drawback can be the amount of computing resources (such as memory) needed to enable virtualized systems to run with useful responsiveness. Thus, full virtualization is more popular in cloud or enterprise systems that prioritize administrative and maintenance scalability. Resources such as memory are less constrained, and because the equipment is typically in a climate-controlled area, it's easier to deal with physical factors such as size and heat dissipation.

By contrast, the physical resources available on the platform limit the number of virtualized environments possible in a static partitioning scheme. Statically partitioned systems have the same benefits of separating mixed-criticality tasks, whether that criticality is related to security, safety or real-time operation. However, since the physical resources couple more directly to the virtualized environments, the timing of tasks is less affected. This makes partitioned systems more suited for embedded systems where computing resources are more limited.

Implementing virtualization

Both full virtualization and static partitioning require underlying software to create the virtualization scheme. This software is called a hypervisor. It runs in a privileged execution level and manages guest machines or guest OSs. The hypervisor also manages the sharing of actual physical resources by virtualized systems.

Much like the different levels of virtualization, there are also different types of hypervisors: type 1 and type 2. A type 1 hypervisor is a dedicated layer of software running on the hardware. It hosts OSs and manages resource and memory allocation for the virtual machines.

A type 2 hypervisor is also known as guest OS virtualization. Here, the lowest layer of software running is a host OS, providing drivers and services for the hypervisor hosting virtualized guest OSs, as shown in **Figure 2**. The guest OSs are unaware that they are not running directly on the system hardware.

A type 2 hypervisor acts as an abstraction layer. Guest OSs become processes of the host OS and depend on the host OS to access hardware resources. The advantages of type 2 hypervisors are that you don't need to make changes to the host or guest OSs, but the drawback is that the layers of abstraction can decrease overall system performance.

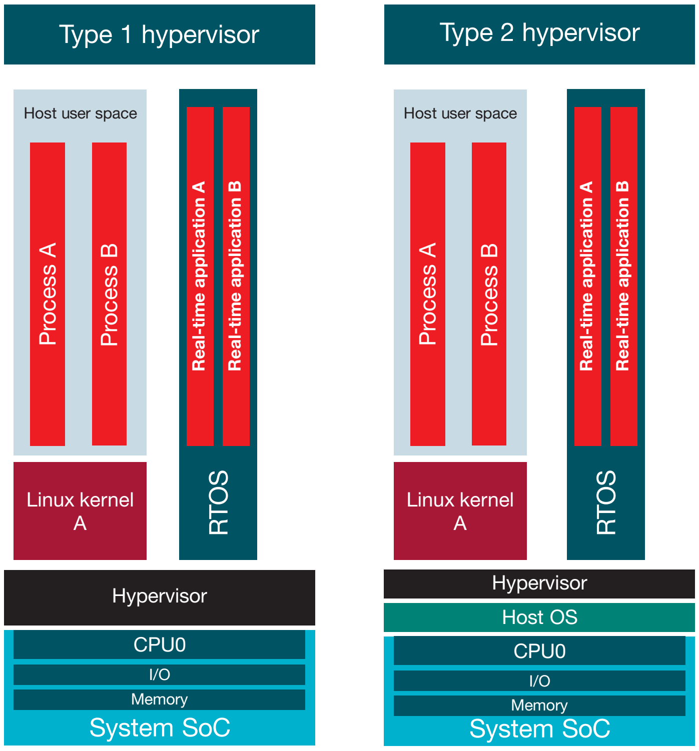

Figure 2: *Comparison of the two types of hypervisors, Type 1 and Type 2.*

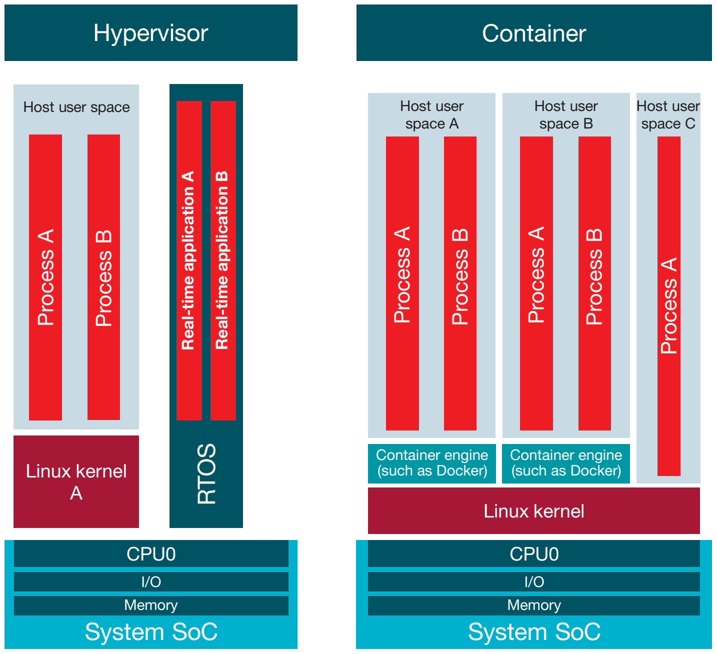

Figure 3: *Comparison of a container and a hypervisor running on an SoC.*

Another popular software solution for application separation aside from hypervisors are containers. Whereas hypervisors virtualize hardware to run multiple OSs, containers virtualize OSs to run multiple applications, as shown in **Figure 3**. Containers are less resource-heavy and can be more application-dense, but the disadvantage is that all applications need to use the same OS.

While the separation of applications is useful in embedded industrial equipment, real-time and non-real-time tasks will be less able to coexist because the system can't run both an RTOS and a high-level OS at the same time. Containers do not address the static partitioning and determinism topics, but you could use them in embedded systems for management of other high-level software stacks without real-time constraints in the Linux side of a partitioned system.

Virtualization solutions

Common virtualization solutions include KVM, Xen, Jailhouse and Docker. Many third-party OSs offer virtualization solutions, such as Wind River® VxWorks®'s virtualization profile, or Green Hills® INTEGRITY®'s multivisor option. These examples all fall within the classifications described above: full virtualization vs. static partitioning hypervisors, type 1 or type 2 hypervisors, or container systems.

KVM is a Linux-based and open-source full virtualization hypervisor. At a basic level, the Linux kernel turns into a hypervisor. It is debatably a type 1 or type 2 hypervisor, and Linux must be the host OS (although any OS can be a guest OS).

Xen is another open-source virtualization solution, but unlike KVM it supports both full virtualization and static partitioning and does not require a Linux host OS, as it is a true type 1 hypervisor. Since Xen supports para-virtualization, it can run on platforms without virtualization extensions, whereas KVM cannot.

Jailhouse is an open-source type 1 hypervisor specifically for partitioning. Unlike other bare-metal hypervisors like Xen, a normal Linux system loads and configures Jailhouse; its management interface is based on the Linux infrastructure. One of the benefits of this infrastructure is that it reuses all of the Linux hardware configuration and setup instead of rewriting it in the hypervisor. Guest OS types are still unrestricted.

Green Hills Integrity and Wind River VxWorks are two popular commercial RTOSs for industrial applications; both offer virtualization solutions. Wind River has a virtualization profile for VxWorks that is currently a type 1 hypervisor where the host OS must be VxWorks. The Green Hills Integrity multivisor is a type 2 hypervisor supporting full virtualization, based on INTEGRITY RTOS.

Docker is a commercial software solution for containers and can work with both Linux and Windows.

Jailhouse: a hypervisor for embedded industrial systems

While it is possible to run each of these virtualization solutions on the Texas Instruments (TI) Sitara™ AM572x and AM65x multicore Arm® Cortex®-A processors, TI currently only directly supports Jailhouse, an open-source hypervisor started by Jan Kiszka and Siemens, as part of the standard software offered in its Processor-software development kit (SDK)-Linux. We identified Jailhouse as the right open-source virtualization solution for industrial embedded systems for three reasons:

- It supports partitioning, which as I discussed earlier is better suited to embedded resources than full virtualization.
- It is more lightweight than other open-source hypervisors like Xen.
- It is optimized for simplicity rather than feature richness.

Once Jailhouse is activated, it runs bare-metal, meaning that it takes full control over the hardware and needs no external support. Unlike other bare-metal hypervisors, however, it is loaded and configured by a normal Linux system as I mentioned earlier. This simplifies its use and makes adoption quicker.

Jailhouse configures the CPUs and device virtualization features of the hardware platform such that none of these domains, called "cells," can interfere with each other in an unacceptable way. It performs no scheduling and only virtualizes resources in software that are essential for a platform and cannot be partitioned in hardware.

Figure 4 on the following page shows the different phases of operation for a system using Jailhouse virtualization. Linux boots first. After Jailhouse is enabled, resources are assigned to additional cells in the partitioning phase. The guest OS or

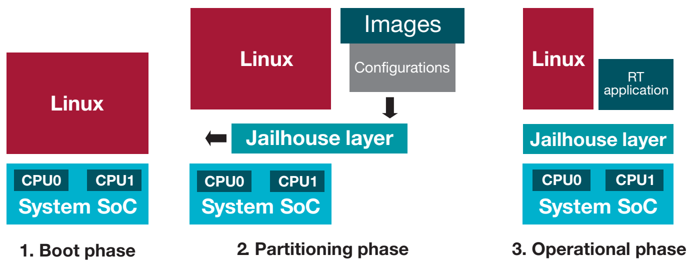

Figure 4: *Phases of operation for a system using Jailhouse virtualization.*

bare-metal application running inside the cell is called an "inmate." Because it is lightweight, constrained to hard-partitioned resources and does not require high computing power or large amounts of memory, Jailhouse-virtualized systems can achieve real-time performance similar to nonvirtualized systems. This makes Jailhouse appropriate for embedded applications, especially where two tasks or systems run side by side with mixed levels of real-time goals.

Returning to the earlier Industry 4.0 example of connecting to a server to upload data for predictive maintenance, you can see that many non-real-time tasks that come from Industry 4.0 or IIoT trends are related to networking. Jailhouse by nature has at least one Linux instance running, which is useful for embedded industrial systems because Linux has a large number of available networking stacks and protocol solutions. For example, it's possible to include security patches for networking stacks without touching the real-time application.

The need for virtualization in industrial embedded systems

Predictive maintenance isn't the only IIoT application pushing manufacturers to find virtualization solutions. Wired non-real-time networking or cloud connectivity both enable a large number of applications that apply to a diverse set of industrial automation applications.

A related application that applies to multiple industrial markets is remote monitoring or remote updating. In many industries—manufacturing, process automation, oil and gas—equipment is often installed in hard-to-reach or hazardous environments where sending a human to check or update the equipment is dangerous and expensive. A connected machine could instead have updates pushed to it remotely from a server.

Another example of something that cloud services in industrial automation enable is hardware-as-a-service (HaaS). In the HaaS business model, vendors provide machines to end customers as part of a contract that also includes maintenance and other services. Because industrial equipment is often very expensive, this model could benefit both industrial manufacturers (who have a predictable revenue stream) and their customers (who can have access to equipment with less upfront capital).

Another benefit to end customers may be more dependable equipment in cases where the contract may specify different terms if the equipment is not functioning properly. However, this model works best when the machine is connected so that the owner, who is not co-located with the machine, can monitor its status and provide services such as maintenance before the machine becomes inoperable.

An even more future-looking application for connected machines is modeling an entire process in the cloud, such as an assembly line or a chemical reaction in pharmaceutical manufacturing, for in-depth optimization analysis, as shown in **Figure 5** on the following page. This type of modeling would require connectivity deployed across almost the entire plant. Connectivity can be built-in to the larger equipment while it would be more cost-effective to have gateways aggregate information from smaller pieces of equipment. Connectivity in industrial equipment will only be able to become this widespread when it becomes cost-effective. While it sounds somewhat far off, leading industrial companies have been preparing to move in this direction by developing cloud platforms such as GE's Predix™ or Siemens' Mindsphere.

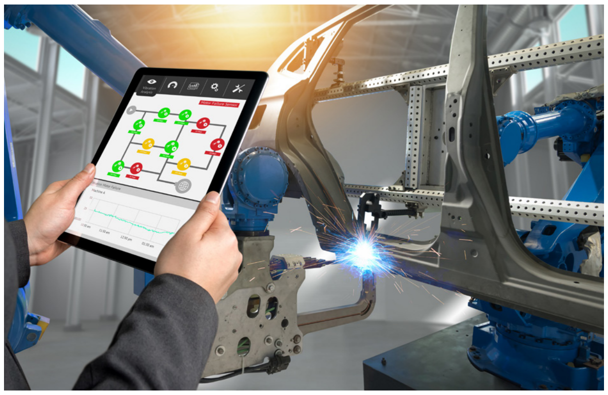
Figure 5: *Model of an automation process viewed on a tablet.*

All of these examples require cost-effective connectivity without compromising the machine's primary mission. Although web or server connectivity applications could benefit from lower latency, real-time operation is in the "nice-to-have" category and cannot be prioritized over the time-sensitive automation tasks that an RTOS manages in most industrial automation equipment: programmable logic controllers (PLCs), computer numerical control (CNC) controllers, motor drives, etc. You must take this compromise into account when considering both parts of cost-effectiveness: the hardware bill of materials (BOM) and the software development effort. While the simplest solution to prevent the two applications from interfering with one another is to separate them into two separate System on Chips (SoCs), this will add cost.

The most cost-effective way to manage software development is to leverage proven solutions that exist on RTOSs for time-critical applications and on Linux for networking applications. The compromise between these two is to run both applications on a multicore processor, with a partitioning hypervisor allowing a different OS to run on each core, thus enabling time-critical applications to meet their deadlines.

Although cloud connectivity and networking are often discussed, virtualization has uses beyond the coexistence of a critical real-time application and less-time-critical networking. Think about taking a standard PLC, machine controller or protection relay and adding a more sophisticated, larger human machine interface (HMI) to enable easier interaction with operators, as shown in **Figure 6**. It would be simpler to enable this larger display panel with standardized graphics frameworks tuned for Linux than it would be to support it with an RTOS, especially when considering that the real customizers of the graphical user interface (GUI) are generally the equipment manufacturer's end customers.

Figure 6: *CNC machine with large HMI*

However, putting the whole system on Linux would not be a good choice for the critical functions of the equipment: the PLC logic, motion control or protection algorithms. One solution might be to separate the HMI and the control into two separate processors on the board, but this has drawbacks in terms of size, cost and overall power consumption. A better solution is to run both tasks on a single CPU while maintaining separation between the tasks and the OSs optimized for those tasks with a simple, lightweight hypervisor such as Jailhouse. Download TI's "**Virtualization Jailhouse Hypervisor on AM572x Reference Design**" to learn more.

The need for virtualization will be increasing as applications that include cloud connectivity or demand easier interactivity between human operators and machines push manufacturers to include more non-real-time functions in real-time industrial equipment. Industrial manufacturers seeking a software strategy for virtualization will need to consider these questions when weighing their options: can they implement the solution quickly without investing a lot of time, R&D or software licensing fees; can they implement the solution on cost-effective hardware without requiring extensive memory; and most critically, can the solution prioritize real-time operation, which is the true goal of the equipment.