Virtualization for embedded systems

The how and why of small-device hypervisors

M. Tim Jones (mtj@mtjones.com), Platform Architect, Intel

Date: 19 Apr 2011
Level: Intermediate

Summary: Today's technical news is filled with stories of server and desktop virtualization, but there's another virtualization technology that's growing rapidly: embedded virtualization. The embedded domain has several useful applications for virtualization, including mobile handsets, security kernels, and concurrent embedded operating systems. This article explores the area of embedded virtualization and explains why it's coming to an embedded system near you.

Tags for this article: mobile_and_embedded_systems, resource_virtualization

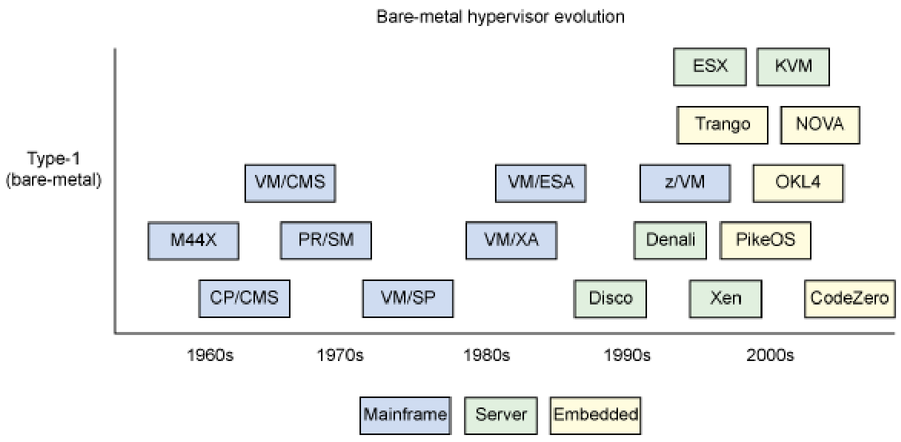
Table of contents

- What is embedded virtualization?
- Attributes of embedded virtualization
- Examples of embedded hypervisors
- Applications of embedded hypervisors
- The future of embedded virtualization
- Resources
- About the author
- Comments

Not only are the markets and opportunities that virtualization creates exploding, but the variations of virtualization are expanding, as well. Although virtualization began in mainframes, it found a key place in the server. Server utilization was found to be so small for a large number of workloads that virtualization permitted multiple server instances to be hosted on a single server for less cost, management, and real estate. Virtualization then entered the consumer space in the form of type-2 (or *hosted*) hypervisors, which permitted the concurrent operation of multiple operating systems on a single desktop. Virtualized desktops were the next innovation, permitting a server to host multiple clients over a network using minimal client endpoints (thin clients). But today, virtualization is entering a new, high-volume space: embedded devices.

This evolution isn't really surprising, as the benefits virtualization realizes continue to grow. Virtualization began and evolved on larger IBM mainframes, but the trend followed the evolution of computing into servers, desktops, and now embedded devices. In the 1990s, virtualization grew outside of the mainframe and, with native microprocessor support for virtualization, experienced a renaissance (see Figure 1).

Figure 1. Brief timeline of type-1 hypervisors



This article explores some of these applications to demonstrate how virtualization is extending its reach and finding new applications. It also explores where open source is leading the way.

What is embedded virtualization?

Embedded virtualization refers to a type-1 hypervisor deployed within an embedded system. This is a somewhat contradictory statement, however. One definition of an *embedded system* is a computer system designed to perform a small number of dedicated functions. But adding a hypervisor to an embedded system adds flexibility and higher-level capabilities, morphing the embedded device into a new class of system.

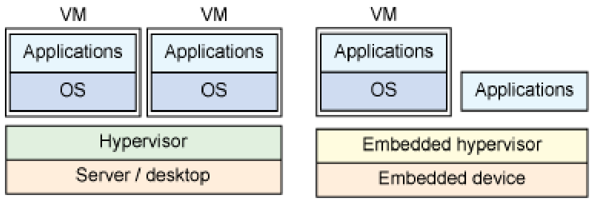
A *hypervisor* is a special type of operating system that (in the case of type-1 hypervisors) runs directly on the bare hardware. The hypervisor creates an abstraction of the underlying hardware platform so that it can be used by one or more virtual machines (VMs) without the VMs knowing that they share the platform. A VM in this context is simply a container for an operating system and its applications. An interesting advantage of this environment is that a VM is isolated from other VMs running on the hypervisor, which permits multiple operating systems or support for similar operating systems with differing configurations. This isolation also provides certain advantages, which I explore shortly.

Key abstractions for microkernels

In 1995, Jochen Liedtke defined three key abstractions that must be provided in a microkernel: *address spaces* (for isolation), *threads* (for concurrency), and *interprocess communication* (for communication between threads in separate address spaces).

In addition, embedded hypervisors are a hybrid of a virtualization platform with a microkernel (see Figure 2). This status allows them to support VMs (operating system plus applications) as well as individual applications.

Figure 2. Bare-metal operating systems and hypervisors



Although embedded devices have been commonly associated with severe resource constraints, today's devices range from powerful processors with server-level functionality (such as hardware support for virtualization) to power-optimized systems with less compute capacity and resources. This variety creates a more demanding environment for embedded hypervisors than their mainframe and server siblings.

⬆ Back to top

Attributes of embedded virtualization

Unlike traditional hypervisors, embedded hypervisors implement a different kind of abstraction with different constraints than other platforms. This section explores some of the constraints and capabilities provided in the embedded space.

Efficiency

All hypervisors strive for efficiency, but embedded hypervisors must deal with added constraints outside of traditional virtualization environments. Outside of processor sharing, memory tends to be one of the key limiters to performance in embedded environments. For this reason, embedded hypervisors must be small and extremely efficient in their use of memory.

Security

Being small has its advantages. The smaller the code size of an application, the easier it is to validate and prove that it is bug free. In fact, some embedded hypervisor vendors have formally verified their hypervisors and guaranteed them to be bug free. The smaller the hypervisor, the more secure and reliable the platform can be. This is because the hypervisor is typically the only portion of the system to run in a privileged mode, which serves as what is known as the *trusted computing base* (TCB) and leads to a more secure platform.

Communication

Embedded hypervisors are built for sharing a hardware platform with multiple guests and applications but also commonly extend communication methods to allow them to interact. This channel for communication is both efficient and secure, permitting privileged and non-privileged applications to coexist.

Isolation

Related to security is the ability to isolate guests and applications from one another. In addition to providing containment for security and reliability, it provides benefits in terms of license segregation. Using the embedded hypervisor's communication mechanism permits proprietary software and open source software to coexist in isolated environments. As embedded devices become more open, the desire to mix proprietary software with third-party and open source software is a key requirement.

Real-time capabilities

Finally, the embedded hypervisor must support scheduling with real-time capabilities. In the case of handsets, the hypervisor can share the platform with core communication capabilities and third-party applications. Scheduling with real-time characteristics allows the critical functions to coexist with applications that operate on a best-effort basis.

⬆ Back to top

Examples of embedded hypervisors

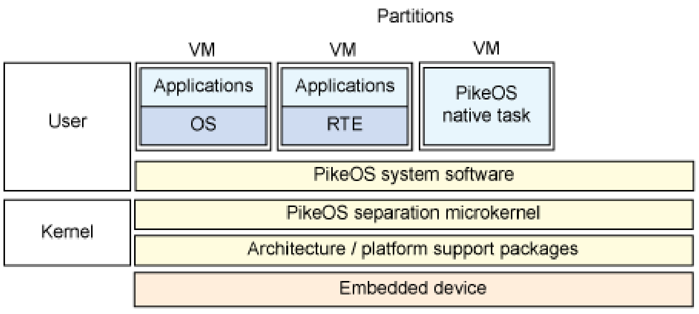
A number of hypervisors have appeared to address applications in the embedded domain—not only from the open source community but also proprietary versions. VMware introduced the Mobile Virtualization Platform to address an increasingly common use model: smart phones for business and personal use. By employing an embedded hypervisor on a smart phone, the single device can be for secure corporate use and for personal use by segregating the two use models in separate containers (VMs).

Let's explore some of the solutions (including open source) that address this and other uses.

PikeOS

PikeOS is an interesting architecture in that it primarily implements what is called a *separation kernel*. Like a hypervisor, a separation (or partitioning) kernel securely isolates environments for upper-level guests. PikeOS is used in the avionics industry for safety-critical avionics applications. Introducing an embedded hypervisor permits the use of older, legacy applications (within a VM) and newer applications on the same platform (see Figure 3).

Figure 3. PikeOS embedded virtualization



Within kernel space, PikeOS implements a set of architecture and platform support packages that exist for the particular hardware environment (x86, PowerPC, SuperH, and others) in addition to the separation microkernel, which provides the virtualization platform. PikeOS supports not only guest operating systems (with associated applications) but also simpler application programming interfaces and run time environments (RTEs) for a specific problem domain (such as the PikeOS native interface or real-time Java™).

The PikeOS system software layer allocates resources (in terms of both space and time) to the guests. The system relies on paravirtualization, so that guest operating systems are aware that they are virtualized.

While commonly used in the avionics industry, PikeOS may soon be in your vehicle through the Automotive Open System Architecture (AUTOSAR). You can learn more in Resources.

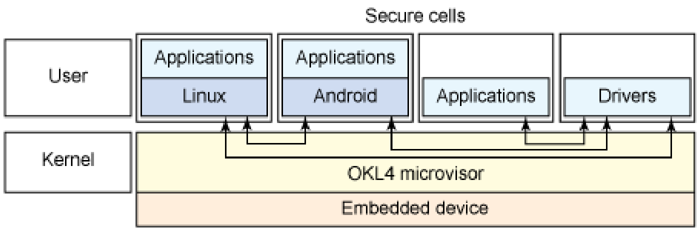
OKL4

In 2006, Open Kernel Labs (OK Labs) was founded for the development of microkernels and hypervisors for embedded systems. The Lab's work in each of these domains coined the term *microvisor,* which represents a microkernel with virtualization capabilities. OK Labs is by far the most successful in the space of embedded virtualization, deploying its open source OKL4 microvisor into more than a billion devices, such as the Evoke QA4 messaging phone, the first phone to support virtualization and operation of two concurrent operating systems (including Linux®).

The heritage of OKL4 comes from the L4 family of microkernels (developed by Jochen Liedtke). L4 was inspired by Mach (the Carnegie Mellon University microkernel developed as a drop-in replacement for the traditional UNIX® kernel). L4 was originally designed entirely in x86 Assembly in order to realize an optimal solution. It then was developed in `C++` and exists in a family of microkernels (from L4Ka::Hazelnut, designed for Intel® Architecture, 32-bit, and ARM-based architectures, to L4Ka::Pistachio, designed for platform independence and released under the Berkeley Software Distribution license).

The OKL4 microvisor implements partitions known as *secure cells* for partitioning VMs in the architecture. The OKL4 microvisor occupies the privileged kernel space, and all VMs, native applications, and drivers are pushed into separate isolated partitions with an efficient interprocess communication (IPC) mechanism to allow cells to communicate and cooperate (see Figure 4). In addition to traditional IPC between VMs, because hardware device drivers are pushed outside of the microvisor (as is typical with microkernels), the IPC is important: It is a common path input/output. Further, because individual applications and drivers can be integrated into the platform without an operating system, the component model for OKL4 is lightweight.

Figure 4. The OKL4 microvisor



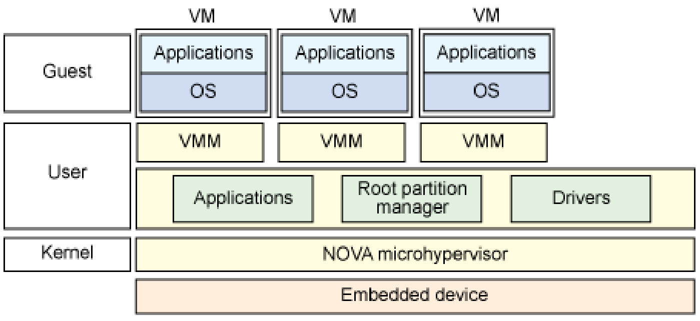
The microvisor implements the core microkernel with virtualization capabilities, which includes resource management as well as scheduling with real-time capabilities and low performance overhead. OKL4 implements paravirtualization, which means that operating systems must be instrumented to run on the microvisor. OK Labs provides support for a number of paravirtualized operating systems, including OK:Linux, OK:Android, and OK:Symbian.

NOVA

In 2010, the NOVA microhypervisor appeared. Like prior microkernel architectures, NOVA implements a thin microkernel with separate layers for non-privileged code. NOVA uses the virtualization capabilities of newer hardware platforms to increase the performance of a component-based system.

NOVA consists of a microhypervisor and user-level environment for core functions of the system. These core elements (shown in Figure 5) consist of a root partition manager (which manages resource allocations, outside of the microkernel proper), drivers for underlying hardware devices, and VM monitors (VMMs) for each guest to manage memory mapping (between the guest and host) as well as sensitive instruction emulation. NOVA implements full virtualization, so certain instructions (like the x86 CPUID) must be properly emulated for each guest based on its configuration. The VMM also implements device emulation for devices made available to each guest. Because NOVA implements full virtualization, unmodified guest operating systems are supported.

Figure 5. NOVA microhypervisor architecture



The microhypervisor itself implements the scheduler, memory management, message-passing communication interface, and other core features, such as protection domains (for spatial isolation) and scheduling contexts (for temporal isolation).

NOVA is considered a third-generation microkernel. Its predecessors include Mach and Chorus (first-generation microkernels) as well as L4 (a second-generation microkernel).

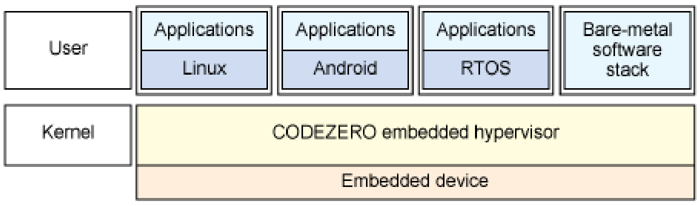
Codezero

The Codezero Embedded Hypervisor is a new microkernel that follows the L4 architecture but has been written from scratch to benefit from the latest research in microkernel design. It follows the fundamental principles of microkernels in that it implements address spaces, thread management, and IPC only in the privileged microkernel along with virtualization capabilities.

As shown in Figure 6, Codezero implements a typical abstraction layer over the hardware platform. The abstraction layer implements threading, IPC, address space management, address space mapping, security, power, and error recovery management. Codezero's scheduler includes kernel preemption for both guest threads and microkernel threads (in addition to time slices for preemption).

Virtualization in Codezero is implemented through containers. Each container is an isolated execution environment with its own set of resources (memory, threads, and so on). The partition also works in concert with Codezero's security and resource management policies, which define capabilities for each container.

Figure 6. Codezero embedded hypervisor architecture



Codezero benefits from recent advancements in microkernel designs. For efficiency, Codezero implements three forms of IPC (all based on the rendezvous model). Codezero implements short IPC (between user space threads), full IPC (256 bytes), and extended IPC (2048 bytes). IPC of larger buffers is performed through shared-page mappings. Codezero was also designed specifically for embedded systems and supports multicore processors as well as ARM-based designs.

Back to top

Applications of embedded hypervisors

The applications of embedded hypervisors is growing. One of the most common today is in mobile phones, where trusted and secure applications (baseband management) share the platform with third-party and untrusted applications. The isolation that the hypervisor provides is a key attribute to their success in this domain. The rapidly growing tablet market will also find uses for this technology as operating systems and applications evolve.

But applications are growing outside of handsets and tablets. Deeply embedded avionics and automotive applications are also finding use for the isolation and reliability aspects of hypervisors. Systems with a focus on security, survivability, or high configurability are finding applications for the technology.

Back to top

The future of embedded virtualization

As you've seen in the numerous embedded hypervisor examples, microkernels tend to be a common design pattern in their architecture and implementation. The approach tends to be thin and efficient, which provides a benefit to performance in addition to less code and improved security and reliability (on the basis of the TCB). Hypervisors continue to be an interesting target for research and continue to find new applications. Where virtualization and embedded systems go next should be fascinating to watch.

Resources

Learn

- To learn more about embedded virtualization, check out the Wikipedia page on embedded hypervisors. You can also find more information on hypervisors and separation kernels.

- Virtualization comes in many flavors. This article showcased the embedded variety, but you can learn about some of the other options in "Virtual Linux" (developerWorks, December 2006). You can also dig into more details of Kernel Virtual Machine and QEMU in Discover the Linux Kernel Virtual Machine (developerWorks, April 2007) and System emulation with QEMU (developerWorks, September 2007). Microvisors also have some history in real-time architectures. Read more in Anatomy of real-time Linux architectures (developerWorks, April 2008).

- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.

- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.

- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.

- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.

- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- PikeOS is an interesting example of an embedded hypervisor (and separation kernel). PikeOS is used widely in avionics systems and making its way into automotive systems, as well (via AUTOSAR).

- OK Labs developed the OKL4 Microvisor from the ground up as an embedded hypervisor. OKL4 has its heritage in the L4 microkernel and is widely used in the mobile phone industry. You can download the OKL4 microvisor and microkernel from OK Labs' wiki page.

- The NOVA microhypervisor was developed at the Technical University of Dresden and represents a third-generation microkernel. Learn more about NOVA at its research page or in this recent paper, "NOVA: A Microhypervisor-Based Secure Virtualization Architecture." Also at Dresden, you can find information on the L4 family of microkernels.

- Get source code for Codezero, another bare-metal embedded hypervisor (and operating system) that is targeted to embedded processors.

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

M. Tim Jones is an embedded firmware architect and the author of *Artificial Intelligence: A Systems Approach*, *GNU/Linux Application Programming* (now in its second edition), *AI Application Programming* (in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective*. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim works at Intel and resides in Longmont, Colorado.