

Virtualization Makes Better use of Open-Source Oses and apps

The increased use of software virtualization in embedded systems is allowing additional use of open-source operating systems and applications.



Author: Robert Day, LinuxWorks, Inc.

The increased use of software virtualization in embedded systems is enabling additional use of open-source operating systems (OSes) and applications. The notion of providing a virtualized interface to hardware and using software separation to contain different applications and OSes is presenting many new use cases for embedded software developers. One such new use more elegantly combines open-source software with proprietary or commercial applications.

Before examining this interesting use case, it's important to understand how software virtualization works in embedded systems. The term "virtualization" is overused and needs to be broken down to clearly see which virtualization technologies are most relevant to embedded systems developers.

Hardware vs. software virtualization

Hardware and software virtualization are more complementary than competitive. Hardware virtualization is now being implemented by many of the processor vendors and provides a more efficient mechanism for partition/OS switching and hardware resource allocation for a software virtualized environment.

Software virtualization provides an application programming interface for OSes

that enables embedded systems designers to take advantage of these virtualized hardware features without having to make any changes to the OS. This has great advantages for the embedded systems market as legacy applications on both legacy and open-source OSes can take advantage of new processor features without new ports or modifications, allowing for an unprecedented migration path for embedded software.

Generally, software virtualization technologies are interchangeably referred to as

hypervisors or virtual machine monitors (VMM) and some different technologies are underneath the hood that enable guest OSes to run on top of them. In general, there are two types of hypervisors—Type 1 (Native) and Type 2 (Hosted).

Two types of hypervisors 1. Type 2: Hosted VMM

Software emulation is a common approach to enabling multiple guest OSes to run on top of a real-time operating

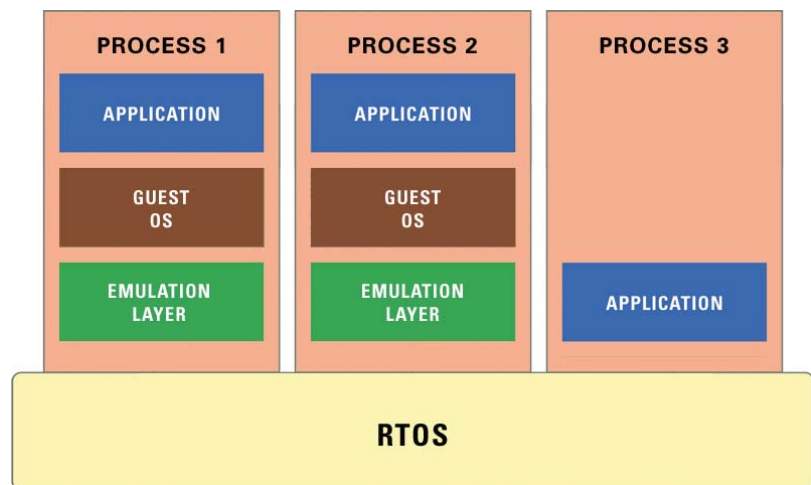


Figure 1: Example of a hosted VMM (Virtual Machine Monitor) solution

system (RTOS). Emulation packages, such as open-source QEMU (available under GNU Lesser General Public License) can run as an application within a process of an RTOS and then emulate the hardware environment that the guest OS is expecting. This type of virtualization is often referred to as a hosted VMM (Type 2).

A hosted VMM has some advantages in that generally the guest OS and applications running on it don't need to be modified. The downside is that this approach can have an impact on the performance of embedded systems because the guest environments are relying on an emulation layer running on top of the underlying OS. Hosted VMMs also have memory-footprint issues because a process-based RTOS plus an emulation layer plus a guest OS plus applications can consume lots of code and data space.

2. Type 1: Native hypervisor

A possibly more elegant and scalable approach for embedded systems is to use a small underlying separation kernel and native hypervisor (Type 1) to provide the hardware interface. This combination provides a small, efficient embedded-software-virtualization layer for running guest OSes. Although this approach requires a closer tie to the target processor, it can more easily take advantage of the new processor features such as hardware virtualization. This technology can also take advantage of an extra virtualization optimization, called paravirtualization, which isn't typically available in the hosted scenario.

Paravirtualization is a term used for a guest OS that's been modified to run on top of a hypervisor. In this case, the virtualization environment that the embedded applications run on has been optimized for performance, both for the processor environment that it's running on as well as the hypervisor. This approach—when combined with hardware virtualization extensions—offers a near-native execution performance for the embedded applications.

With open-source OSes such as Linux®, this approach is particularly appropriate, as the source to the kernel and board support packages are available, and the performance gains of paravirtualization can make Linux more widely applicable as an embedded OS.

Separation kernels

Another approach that can be used with this technology is full virtualization. This where the same hypervisor can offer a virtualization environment to the guest OS that's similar to running on the native hardware. This requires no changes to the guest

OSes, but because the hypervisor is adding more virtualization, there's a small performance hit over the paravirtualized approach.

An interesting approach

Where things really start to get interesting is when these two approaches are combined. In today's embedded systems, there are often components that are real-time and others (such as user or file-access systems) that don't require real time but often need a GUI. These components are often compartmentalized using different hardware and OSes (Linux or Windows® for the GUI, and an RTOS for the real-time portion, for example) to give the best building blocks while keeping the real-time determinism intact.

With a separation kernel and hypervisor, these worlds can be combined on a single hardware platform. The separation kernel communicates both with the underlying hardware platform to partition the appropriate resources and with the hypervisor to ensure that the guest OSes have what they require. The separation kernel and hypervisor determine how to best make use of hardware optimizations and extensions available. This includes hardware virtualization and multicore devices, offering separate secure partitions for each of the guest OSes to run in.

This software separation means that the real-time performance of the RTOS partition isn't compromised by the other, often open-source, guest OSes, and any fault conditions that occur in any of the partitions are confined to that partition. In a multicore system, the separation kernel can also allocate processors to partitions, effectively mimicking the traditional hardware separation using a hardware/software combination and not compromising system performance by sharing processors across different OSes.

The separation kernel can also determine inter-partition communication, using policies to determine which partitions can talk to one another and can also administer the sharing of physical peripheral devices such as displays, network connections and I/O functions between the different partitions, OSes and applications.

Using today's modern multicore and virtualized hardware gives multiple applications and OSes (either paravirtualized or fully virtualized) their own secure partitions to operate in. The relatively small size and efficient operation of this solution means that very little compromise is required for embedded systems developers to adopt this technology today.

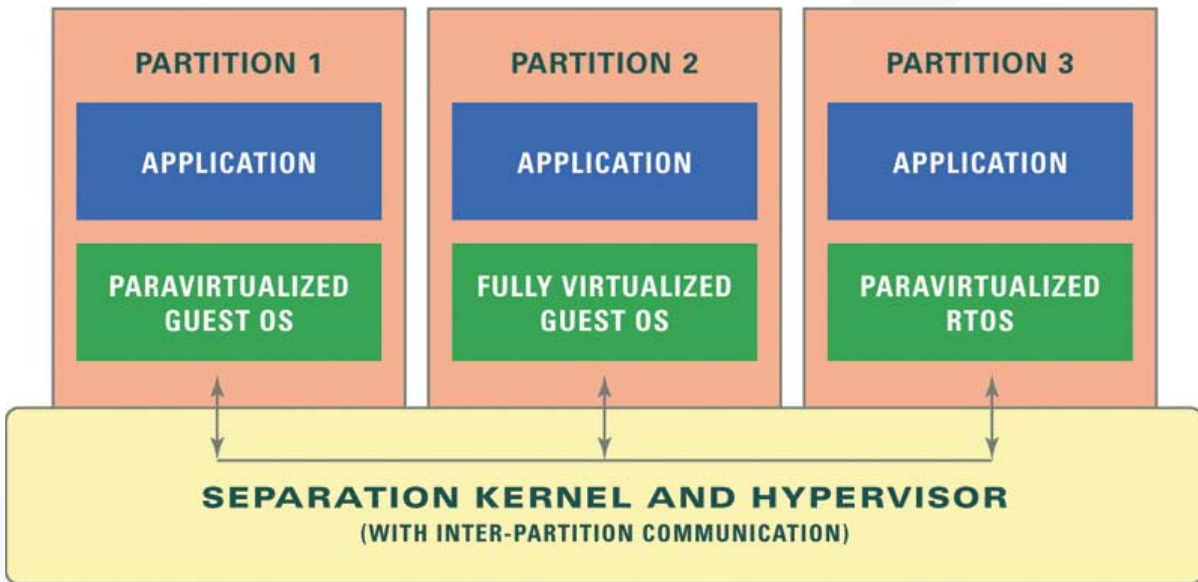


Figure 2: The combination of separation kernel and hypervisor allows multiple OSs to be run securely on the same physical hardware

A partitioning advantage

This solution also introduces open-source applications and OSes into embedded systems where their use has been traditionally prohibited either due to performance or licensing issues. By allowing the separation kernel and hypervisor to partition different OSes, embedded systems developers now have the opportunity to bring in open-source OSes for the user-interface portion of their system without compromising the hard real-time aspects of their system. The separation kernel can also keep the proprietary applications of the embedded system separate from the open-source parts. And, because no linking of GPL libraries is necessary (because they're loaded into their own partitions), the possible "copyleft" issues of using open-source products are avoided.

This software virtualization environment gives embedded software developers the option of bringing non-real-time applications into a hard real-time system and open-source licensed solutions next to proprietary ones on the same hardware. Such a solution was previously available only by using physical hardware separation.

Introducing open-source OSes and applications into embedded systems has often been difficult due to performance or licensing issues, and hence the benefits of reusing existing open-source

and commercial-software intellectual property has been missed. With the introduction of the embedded hypervisor, open-source software is now becoming available to a wider range of embedded systems, spreading the benefits of reuse and cost-effective software development across the spectrum of embedded devices.



Robert Day is Vice President, Marketing at LynuxWorks, Inc. and has more than 20 years of experience in the embedded industry. Based in San José, California, Robert is a graduate of the University of Brighton, England, where he earned a Bachelor of Science degree in computer science.



1.800.255.5969



LynuxWorks, Inc.
855 Embedded Way
San José, CA 95138-1018
408.979.3900
408.979.3920 fax
www.lynuxworks.com

LynuxWorks Europe
50 Broadway
London SW1H 0RG
United Kingdom
+44 208 906 9506
+44 208 906 2338 fax

©2011 LynuxWorks, Inc. LynuxWorks and the LynuxWorks logo are trademarks, and LynxOS and BlueCat are registered trademarks of LynuxWorks, Inc. Linux is a registered trademark of Linus Torvalds. All other trademarks are the trademarks and registered trademarks of their respective owners.

All rights reserved. Printed in the USA.