Hypervisors in Embedded Systems

Applications and Architectures

Jack Greenbaum
Green Hills Software, Inc
Santa Barbara, California, USA
jackg@ghs.com

Cesare Garlati
prpl Foundation
Santa Clara, California, USA
cesare@prplFoundation.org

***Abstract* — As microprocessor architectures have evolved with direct hardware support for virtualization, hypervisor software has become not just practical in embedded systems, but present in many commercials applications. This paper discusses embedded systems use cases for hypervisors, including their use in workload consolidation and security applications.**

***Keywords* — hypervisor; virtualization; virtual machine; guest OS; embedded systems; security; IoT; Internet of Things.**

I. Introduction

Hypervisors are a type of operating system software that allows multiple traditional operating systems to run on the same microprocessor [1]. They were originally introduced in traditional IT data centers to solve workload balancing and system utilization challenges. Initial hypervisors required changes to the guest OS to compensate for a lack of hardware support for the isolation required between guest operating systems. As microprocessor architectures have evolved with direct hardware support for virtualization, hypervisors have become not just practical in embedded systems, but are present in deployed applications [2]. Hypervisors are here to stay in embedded systems. This paper discusses embedded systems use cases for hypervisors, including their use in workload consolidation and security applications.

Hardware support for virtualization in modern microprocessors has been the necessary enabler for virtualization to move from the data center to embedded systems. All of the major processor architectures have evolved with virtualization extensions. Notable examples include Intel VT-x, ARM Virtualization Extensions, and MIPS VZ extensions. This support includes a distinct hypervisor execution mode at a higher privilege level than the traditional supervisor mode, and IO MMUs to isolate peripheral devices used by different guest operating systems from each other. Without an IOMMU the unique IO requirements of embedded systems cannot be properly separated. The Intel version is called VT-d, and most ARM processors have a “System MMU”. In the data center the IOMMU is sometimes called Single Root Virtualization, or SRV.

The rest of this paper focuses on the use cases for hypervisors in embedded systems, and introduces the capabilities that hypervisors provide to implement these use cases.

II. Use Cases

A. *Consolidation*

The most common use of hypervisors is to consolidate multiple workloads onto a single platform in order to reduce size, weight, power, or cost. This is the same use case that has driven broad adoption of hypervisors in the IT server space. As servers have grown to have more capacity than any single application, virtualization lets one server combine multiple applications. But integrating multiple applications from different customers onto one operating system puts too many constraints on what functions can be combined. Virtualization instead runs multiple operating systems on the same hardware, allowing complete applications to run on the same hardware with very little interaction.

Consolidation use cases are becoming common in automotive systems. One example is combining the instrument cluster and in-vehicle infotainment (IVI) systems into a single electronic control unit (ECU). The instrument cluster is typically built on a real-time operating system (RTOS), while IVI is often built on Linux or another general purpose OS (GPOS). The real-time and safety requirements of the instrument cluster cannot be met by GPOS, and the media libraries required for IVI are expensive to port to an RTOS. Therefore, integrating these two functions into one OS is not feasible. But a hypervisor with real-time and safety guarantees can run both the RTOS and GPOS on the same processor within a single ECU. This saves not only cost (by having only one processor and circuit board), but also space in the vehicle that is increasingly full with more and more ECUs for modern safety features.

B. *Legacy Operating Systems*

As systems evolve over time, it often becomes necessary to make a shift to a new operating environment to enable new features. Preserving the existing features of the system would then require porting already tested and field proven software to the new platform. Virtualization, on the other hand, allows running the existing operating environment alongside of the new software on the same processor. One example is a software defined radio. Over time the product requirements may evolve to require a transition from a simple LCD user interface to a graphical user interface (GUI). The radio software may have a high cost to recertify. By using virtualization, the radio protocol software can be maintained while a second OS with a modern GUI library can be run alongside. By running the radio protocol software unchanged (or with minimal change), a GUI can be added while minimizing or eliminating recertification costs for the radio protocol software. This use case is most common in deeply embedded and very cost sensitive applications.

C. Multiple Security Levels (MILS)

A third example is a combination of the consolidation and legacy cases. The application in this case is to provide security isolation between two different workloads that have different security postures. Two examples include running Trusted Execution Environments and dual-persona smart phones.

Trusted Execution Environments (TEEs) provide security critical processing in an environment isolated from the rest of the system. Use cases include secure boot, cryptographic services, and security critical device feature management including the IOMMU. This is similar to consolidation in that cryptographic services have traditionally been offloaded to a separate, smaller, core. The advantage to running cryptographic services in a TEE on the main processor cores is typically higher performance than traditional approach.

A dual persona phone acts like two separate smart phones. The common application is one partition is an operationally secure partition, while the other partition can be updated or reconfigured by the user. Typically, the secure partition is controlled by a business IT department or a government entity that manages compartmentalized information. Such a partition may have access to restricted networks and therefore contains high value encryption keys and information. The software load on the secure partition is often locked down and verified at boot time. The second partition is often called a user partition, and has access to the public internet and can install apps from an app store and access other unsecured content. The underlying assumption is that the hypervisor provides a higher level of isolation than the individual operating systems being virtualized

III. Hypervisor Capabilities

All hypervisors provide isolated sharing of System On a Chip (SoC) resources, but differ in the scope and depth of support for sharing the different hardware elements.

A. Memory Sharing

The most basic hypervisor runs on a multi-core SoC and provides only for sharing of memory. Each CPU core runs a separate hardware load. The hypervisor configures the SoC's virtualized memory management - see section below - to restrict each CPU to a portion of the memory address space of the SoC, including both RAM and peripheral registers. This allows multiple Operating systems to run on a single SoC with disjoint peripherals and secure shared access to RAM.

B. CPU Sharing

A more capable hypervisor also allows sharing of individual CPU cores via time slicing. This allows different workloads to have access to all CPU resources during times of heavy demand, and to partition that access based on priority. For example, when consolidating RTOS and GPOS workloads, the RTOS is typically given priority on the CPUs, while the GPOS gets a guaranteed minimum amount of execution time. The GPOS has full access to the CPUs when the RTOS is idle. Note that a hypervisor that supports CPU sharing in this way typically must be written with real-time behavior in mind, and is often based on an RTOS.

C. Peripherals Sharing

Another set of hypervisor features revolves around sharing of peripherals, such as mass storage, communication links, and GPUs. Embedded systems are often cost sensitive, so the ability to share devices such as eMMC mass storage is required. There are several different techniques for implementing peripheral sharing, including mediated pass through, device emulation, and paravirtualization. Each approach has its strengths and weaknesses. A full discussion of these concepts is beyond the scope of this paper, but when considering the use of a hypervisor the sharing of devices is as important to consider as is the sharing of the CPU.

IV. Conclusion

Hypervisors have moved from the data center to embedded systems, enabled by hardware support in modern microprocessors. We have outlined the common use cases for virtualization, and considerations for device sharing.

References

[1] Security Guidance for Critical Areas of Embedded Computing, prpl Foundation, January 2016 - https://prpl.works/security-guidance/

[2] prplSecurity Framework Application Note, prpl Foundation, July 2017 - https://prpl.works/application-note-july-2016/