Chapter 15
Embedded Virtualization Techniques for Automotive Infotainment Applications

Massimo Violante
Politecnico di Torino, Italy

Gianpaolo Macario
Mentor Graphics Embedded Software Division, Italy

Salvatore Campagna
Politecnico di Torino, Italy

ABSTRACT

Automotive infotainment applications are examples of embedded systems in which a heterogeneous software stack is used, which most likely comprises a real-time operating system, an automotive-grade Linux, and possibly Android. Thanks to the availability of modern systems-on-a-chip providing multicore computing platforms, architects have the possibility of integrating the entire software stack in a single chip. Embedded virtualization appears an interesting technology to achieve this goal, while providing the different operating systems the capability of exchanging data as well as optimizing resource usage. Although very well known in server-class systems, virtualization is rather new to the embedded domain; in order to leverage its benefits, it is therefore mandatory to understand its peculiarities and shortcomings. In this chapter, the authors illustrate the virtualization technologies with particular emphasis on hypervisors and Linux Containers. Moreover, they illustrate how those technologies can cooperate to fulfill the requirements on automotive infotainment applications. Finally, the authors report some experimental evidence of the performance overheads introduced when using embedded virtualization.

INTRODUCTION

High-performance systems-on-a-chip (SoCs) are paving the way for new classes of embedded systems that promise lower cost, higher integration, low power consumption and higher performance. Many applications exist where heterogeneous software components cooperate for reaching a common goal, which may benefit from the integration into SoCs, possibly based on multi-core

DOI: 10.4018/978-1-4666-6194-3.ch015

technology. As an example, we can consider the architecture of two embedded systems belonging to different application scenarios:

- **Industrial Machines:** Embedded systems used for controlling industrial machines are typically running two types of software: a graphical front-end responsible for implementing the human machine interface (HMI), and a real-time backend responsible for executing the control loop of the machine. The two software components have different requirements in terms of dependability (HMI is typically not responsible for safety-critical operations, while the real-time backend is likely to be safety critical), and performance (the real-time backend has stringent timing requirements, while the HMI does not). HMI and real-time backend can be implemented resorting to different operating systems (e.g., Windows/Linux for HMI, and a real-time operating system like VxWorks, QNX, etc., for the real-time backend), and may be mapped on different processor-based systems.
- **Infotainment Systems:** Embedded systems employed in infotainment systems combine safety-relevant software components (for example for managing communications with the vehicle network), automotive-grade operating systems compliant with reference architectures like GENIVI (GENIVI, 2014) to provide all the services required by automotive applications (e.g., localization services, navigation services, etc.), as well as commodity software components for implementing other services, like Android for HMI and Android market access. Safety-relevant software components are normally delegated to specialized microcontrollers, while the other components are executed by a general-purpose microprocessor.

Analyzing the above-mentioned application scenarios we can see the benefit of high performance SoCs. SoCs (possibly equipped with multicore processors) can integrate on single devices all the software components, thus reducing the size of the resulting embedded systems, their power consumption, and costs. The benefits of SoCs are fully exploitable only if a proper infrastructure is available for allowing the coexistence of heterogeneous software components with conflicting requirements. As an example, running on the same device a software component that is safety-relevant along with a non-safety-relevant one poses new challenges to designers. Designers must indeed guarantee that the non-safety-relevant software never interferes with the safety-relevant one, no matter what it is doing. Similar considerations apply to the coexistence of real-time and non-real-time software components.

To facilitate the coexistence of heterogeneous software components, more than 40 years ago researchers developed the concept of virtual machines and hardware virtualization. Through virtual machines, designers can create a self-contained abstraction of the underlying hardware. As a result, any software component running on the virtual machine executes as if running on a dedicated hardware. Any misbehavior of a software component is confined inside the virtual machine running it; the virtual machine monitor, which orchestrates the operations of all the virtual machines, blocks any interference between different software components. For several years the concept of virtual machine remained relegated to high-end computing systems, which were the only ones capable of providing adequate computing power to run multiple virtual machines; only recently virtualization started becoming interesting also for embedded systems thanks to the availability of high-performance SoCs, and a number of products specifically designed for embedded systems started to appear on the market.

Being a relatively new technology for embedded systems, hardware virtualization needs to