Virtualizing Embedded Systems – Why Bother?

Gernot Heiser
NICTA and University of New South Wales
Sydney, Australia
gernot@nicta.com.au

ABSTRACT

Platform virtualization, which supports the co-existence of multiple operating-system environments on a single physical platform, is now commonplace in server computing, as it can provide similar isolation as separate physical servers, but with improved resource utilisation.

In the embedded space, virtualization is a new development, which is likely to become more widespread in the next few years. Unlike the server world, where virtualized systems typically run multiple copies of the same (or similar) operating systems, most uses of virtualization in the embedded space are heterogenous, combining different classes of operating systems: an RTOS for traditional embedded real-time programming, and a fully-featured ("rich") operating system to support complex applications such as user interfaces.

We provide a number of examples of present or likely use cases of virtualization in embedded systems, and explain the motivation and benefits, as well as some of the differences to server-style virtualization.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Design, Reliability, Security

Keywords

Virtual machines, hypervisors, virtualization, safety, security, processor consolidation

1. INTRODUCTION

Virtualization has proved to be a game-changer in the server space, fundamentally because it enables better resource usage through co-locating services, while maintaining strong quality-of-service (QoS) isolation. Resources here refers not only to the computing platform per se (CPU, memory etc.) but also plant infrastructure and electric energy required for powering computers and air conditioners.

There are secondary benefits of virtualization which are becoming increasingly important, such as ease of checkpointing whole systems, migrating services between geographically distributed centres, replay for debugging, the ability to run old versions of an operating system (OS) to support legacy software.

In addition, virtualization is used increasingly on desktops. Usually, the main motivation is to provide access to a second OS, usually for running applications not available on the user's primary OS. Other uses include low-level system development, and sandboxing of potentially dangerous code.

At a first glance, none of the above virtualization use cases seem to have any relevance to embedded systems. Yet there is increasing interest in virtualization in the embedded world [Hei08], including end-user deployments. In this paper we look at what drives the uptake of virtualization in embedded devices. We examine in detail three kinds of virtualization uses: consumer-electronics (CE) devices, where virtualization is used to co-locate real-time environments with desktop-like OSes that provide a high-level programming environment for applications, devices where virtualization is used to provide an isolated environment for safety- or security-critical components, and cars, where virtualization is used to integrate infotainment with automotive control and convenience functionality.

2. VIRTUALIZATION USE CASES

2.1 Consumer Electronics

Presently, the most developed use cases of embedded virtualization are in consumer electronics (CE) devices, especially the biggest of all (in terms of global revenue): the mobile phone. The first virtualized phone, the Motorola Evoke, went on sale in April 2009 [Hei09].

The driver for virtualization in this device (and most other CE use cases) is the *co-existence of multiple OS environments*. Running multiple OSes is also a main motivation for the adoption of virtualization in the server space. The big difference is that in servers, the OSes are either the same or of a similar nature (fullly-featured, feature-rich OSes such as Linux or Windows). In contrast, virtualization on CE devices typically combines vastly different OSes, such as a "rich" OS (eg. Android or Windows) and a simple real-time OS (RTOS).

The reasons behind such a heterogenous design are the

DAC 2011, June 5-10, 2011, San Diego, California, USA.

combination of real-time requirements on the one hand, and growing functionality (and hence complexity) on the other hand, which is typical for many CE devices.

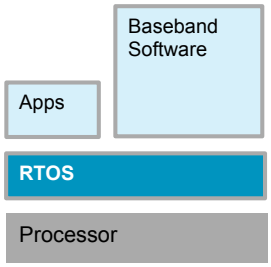

Figure 1: Old-style phone design.

Mobile phones are a good example of this. They have grown from simple voice-communication devices to personal computers in the pocket. This immense growth of functionality (and software complexity) necessitated a change in system design. Only a few years back, the standard design of a (feature-rich by the standards of the day) phone consisted of a general-purpose processor (typically complemented by a DSP and possibly other accelerators) under the control of an RTOS. The RTOS supported the real-time operation of the radio software (called the *baseband stack*), as well as an application programming environment for the user-visible functionality (dialler, address book, rudimentary internet applications), as shown in Figure 1.

The growth in user-visible applications lead to *smartphones*, characterised by the existence of PC-like applications, such as web browsers and full-featured email clients, and, most importantly, the ability to download and install a wide range of "apps". This turned the phone into an open platform, similar to a PC. And, in order to support the development of apps (by now there are hundreds of thousands), it required a feature-rich OS with a standardised and familiar programming interface (API).

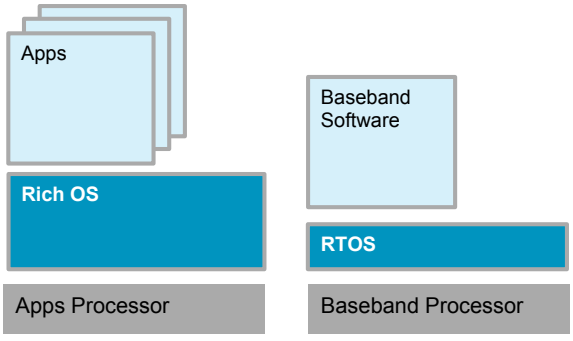

Figure 2: Smartphone design.

As shown in Figure 2, the typical design of a smartphone uses two separate processors, a *baseband processor* which runs an RTOS supporting the radio protocol software, and an *application processor*, which runs a rich OS (Linux/Android, Symbian, iOS or Windows) and provides the open environment for the numerous apps.

There is an intermediate design, enabled by virtualization, as shown in Figure 3. This design is suitable for lower-end smartphone-like devices, which run a rich OS but do not require the processing power (and energy use) of two separate processors. It uses a single processor (core), which a *hypervisor* presents to software as two separate (virtual) processors, one for the baseband, the other for the application stack.

The above-mentioned Motorola Evoke used this design: A single ARM926 processor, designed as a baseband processor and clocked at around 200MHz, runs the Linux and BREW OSes in separate virtual machines, provided by the OKL4 microkernel acting as a hypervisor.

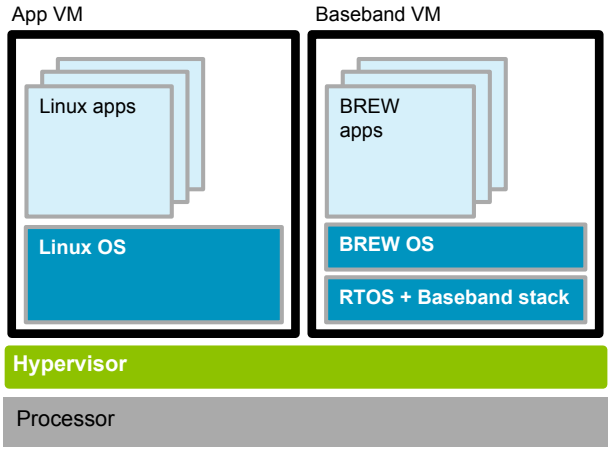

Figure 3: Virtualized phone design as in the Motorola Evoke.

The designers of the Evoke made good use of the fact that the two OS environments, although in different virtual machines, shared the same physical processor: They employed the fast communication mechanisms provided by the OKL4 microkernel to allow media players hosted on Linux to use the BREW rendering engines [Hei09]. Communication overheads would have been much higher if the OSes ran on separate physical processors, probably too high to share rendering engines.

Processors used in contemporary smartphones have the baseband and application cores integrated on a single chip, which eliminates most of the cost advantage of the design of Figure 3. However, this does not spell the end of virtualization for phones, to the contrary. The increased processing power available on phones, and the trend for phones (and tablets) to replace PCs, creates new drivers for the uptake of virtualization technology.

Phones and tablets are increasingly replacing laptops as devices of choice used for remotely accessing enterprise IT systems, as well as for personal information, connectivity and entertainment. However, people prefer to use a device of their choice, rather than a standard-issue (and locked-down) business phone or tablet, and they are certainly disinclined to carry two similar devices, one for personal and the other for business use.

Virtualization can be used to support a "bring your own device" (BYOD) scenario, where an employee can own (and mostly control) a device but still access the enterprise system securely [DoCo06,OKL10]. In the BYOD approach, two separate logical phones, private and business, exist on a single physical one, as depicted in Figure 4. Each runs in its own

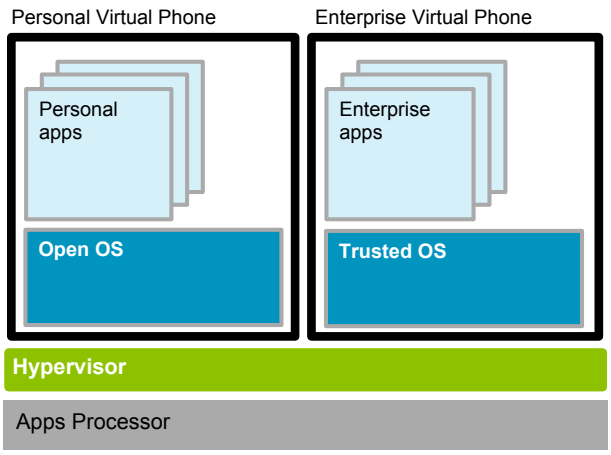

Figure 4: Personal and work phones integrated on a single physical device.

virtual machine on the physical phone hardware. The business phone is under control of enterprise IT staff, and configured for the needs of the enterprise, while the owner has full control over the personal phone. Virtualization-enforced isolation keeps enterprise data safe, even if the user installs a malicious app on the personal phone. And in turn, the user's private data is out of reach of the business phone.

2.2 Secure Communication Devices

The above BYOD phone/tablet use case is an example of the use of virtualization for security. A slightly different design is presently prototyped for more sensitive national-security use cases. The idea is to make secure communication devices less expensive by basing them on standard commercial designs. As indicated in Figure 5, this approach uses virtualization to provide a secure environment with a minimal *trusted computing base* (TCB). This environment houses critical components which implement a secure voice-over-IP communication channel. It uses crypto software to allow secure transmission of voice calls over the untrusted phone software [OKL11].

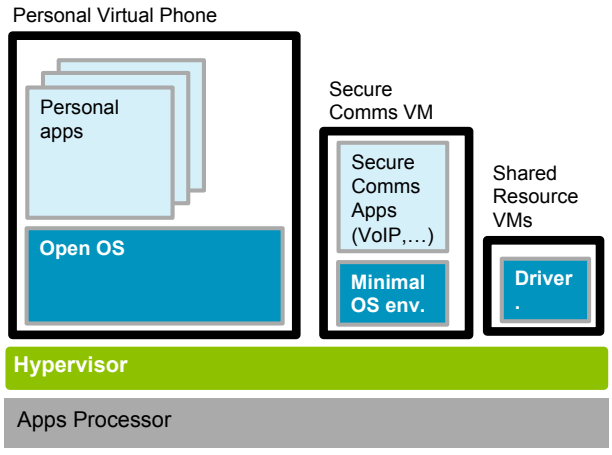

Figure 5: Secure communication on standard phone.

The main difference to the BYOD use case is that instead of supporting two complete systems, this design only isolates the minimal (security-critical) functionality, and relies on the standard environment to handle the (encrypted) data. The secure environment does not require much of an OS, mostly some support libraries; the code executes on bare (virtual) hardware. This keeps the attack surface small.

2.3 Medical devices

Many other virtualization use cases are a result of the increasing functionality (and, as a result, complexity) of many security- or safety-critical devices. Examples are medical devices, payment systems and cars.

Life-supporting medical devices used to be bulky, expensive and located in hospitals and clinics, where they were operated by specialists with years of training. Increasingly such devices are wearable or implanted, which means that they must be operated remotely (requiring sizeable and complex communications software) or by the patient (requiring easy and intuitive user interfaces). In either case, the critical, life-supporting function must co-exist with large and complex software stacks, often including complete multi-million-line OSes (like Linux or Windows). Such software is inevitably buggy and thus unreliable, and must therefore be prevented from unduly interfering with the life-supporting software.

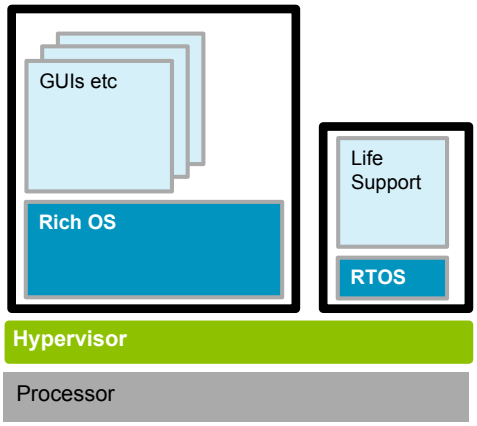

Figure 6: Patient-operated medical device with graphical user interface (GUI).

Isolation could be achieved by physical separation with a simple connection (eg. a serial line), but this approach is expensive, effectively doubling the system cost. A better alternative is to co-locate both parts onto one (single- or multi-core) processor, and achieve the isolation by putting the parts into separate virtual machines, as shown in Figure 6.

Conceptually, this design is similar to the one of Figure 5: it also tries to minimise the amount of trusted code. The same general structure, a small, critical subsystem co-existing with a large, untrusted software stack, appears in many modern designs. An example are payment systems, which are increasingly part of a larger (frequently mobile) system.

2.4 Cars

Automotive software presents another case for virtualization, for yet different reasons. The standard approach to automotive electronics used to be to provide a separate mi-

crocontroller for each function. With the proliferation of control and convenience functionality, this is quickly becoming really expensive, as each of these *electronic control units* (ECUs) needs power supplies and at least one communication interface, and must be packaged to withstand heat, water, grease, acid and vibrations. The balance-of-system cost can easily be an order of magnitude higher than the cost of the basic microprocessor and memory components.

This issue has been partially addressed by the move to protected multi-tasking operating systems, in particular AUTOSAR [AUTO], which supports implementing multiple functions on a single ECU. However, AUTOSAR is very much designed for traditional automotive control and convenience functions, and does not support the growing area of infotainment, which requires interaction with CE devices and standards. Moreover, the two domains are becoming increasingly integrated: for example, information on road conditions obtained through the infotainment system is used to adjust engine and steering behaviour (eg. an "electronic stability program" reacts to road conditions).

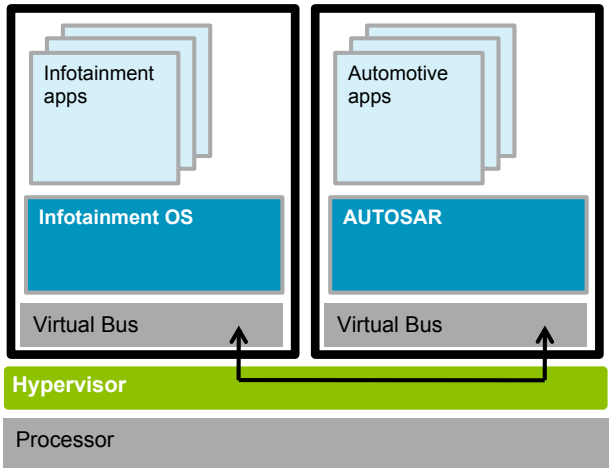

Figure 7: Integration of infotainment and control/convenience functions in cars.

Virtualization supports a safe integration of the infotainment and the classic automotive worlds, by providing an automotive OS (such as AUTOSAR) on the same ECU as one (or several) CE OSes (Linux or Windows), plus a gateway which converts the different communication standards, as shown in Figure 7 [HH08]. The benefits of this approach are significant, in terms of reduced hardware cost as well as overall system complexity. As a consequence, the first deployments of virtualized ECUs are not far off (expected in 2012). Similar issues, and possible solutions, exist in the world of aerospace, although deployment there is likely to take longer.

3. HARDWARE SUPPORT

The importance of virtualization in server and desktop computing has resulted in the introduction of architectural support [UNR+05]. Intel includes their architectural support in their Atom processors, which are aimed at use in high-end embedded systems. The mobile embedded space is these days dominated by ARM processors. The importance manufacturers put on virtualization can be seen in the fact that late last year ARM announced their own architectural support [ARM10]. Hardware implementations are expected for 2012. Vendors of other processor architectures are also working on virtualization extensions to their architectures.

These extensions have in common that they support *full virtualization*, enabling the execution of unmodified native OS binaries in a virtual machine. Without such extensions, an OS must be modified ("para-virtualized") in order to run in a virtual machine, an expensive (in terms of engineering cost) and error-prone procedure. Hardware extensions can also help to reduce the run-time overhead of virtualization, by significantly reducing the frequency of hypervisor invocations (although this is a lesser issue in embedded processors, where very low overheads are achievable even without hardware support [HL10]).

4. CONCLUSIONS

Virtualization in embedded devices is happening. It is already deployed in some CE domains, and will soon appear elsewhere, especially the automotive sector. Fundamentally its attraction is the ability to support the co-existence of vastly different subsystems at reduced overall system cost. Typically, virtual machines are used to support unprotected RTOSes together with high-level rich OSes, vertical-specific OS and communication standards together with CE standards, and security- or safety-critical subsystems together with large, complex application stacks. Processor manufacturers are working on providing hardware support in order to reduce the cost of virtualization.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

5. REFERENCES

[ARM10] ARM Architecture Group. *Virtualization Extensions Architecture Specification*, 2010. URL `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0406b_virtualization_extns/index.html`.

[AUTO] AUTOSAR. `http://www.autosar.org`.

[DoCo06] NTT DoCoMo and Intel Corp. Open and secure terminal initiative (OSTI) architecture specification. `http://www.nttdocomo.co.jp/english/corporate/technology/osti/`, Oct 2006.

[Hei08] G. Heiser. The role of virtualization in embedded systems. In *1st WS Isolation & Integration Emb. Syst.*, pages 11–16, Glasgow, UK, Apr 2008. ACM SIGOPS.

[Hei09] G. Heiser. The Motorola Evoke QA4: A case study in mobile virtualization. White paper, Open Kernel Labs, Jul 2009. `http://www.ok-labs.com/_assets/image_library/evoke.pdf`.

[HH08] A. Hergenhan and G. Heiser. Operating systems technology for converged ECUs. In *6th Emb. Security in Cars Conf. (escar)*, Hamburg, Germany, Nov 2008. ISITS.

[HL10] G. Heiser and B. Leslie. The OKL4 Microvisor: Convergence point of microkernels and hypervisors. In *1st APSys*, pages 19–24, New Delhi, India, Aug 2010.

[OKL10] Open Kernel Labs. Open Kernel Labs unveils SecureIT mobile for building secure smartphones. http://www.ok-labs.com/releases/release/open-kernel-labs-unveils-secureit-mobile-for-building-secure-smartphones, Oct 2010.

[OKL11] Open Kernel Labs. OK Labs and Fixmo to deliver mobile security for military, government, and enterprise. http://www.ok-labs.com/releases/release/open-kernel-labs-and-fixmo-collaborate-to-deliver-mobile-security-solution, Feb 2011.

[UNR+05] R. Uhlig, G. Neiger, D. Rodgers, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kägi, F. H. Leung, and L. Smith. Intel virtualization technology. *IEEE Comp.*, 38(5):48–56, May 2005.