Use of XtratuM in an Automotive Application (*)

J. Sánchez , S. Peiró, A. Crespo, J. Simó, M. Masmano, P. Balbastre
Instituto de Automatica e Informatica Industrial
Universidad Politecnica de Valencia, Spain
{jsanchez,mmasmano,speiro}@ai2.upv.es, {jsimo,acrespo,patricia}disca.upv.es

Abstract

Virtualization is playing a key role in many of today software systems. At first used mainly to improve resource utilization, this technology is expanding and has reached the market of embedded systems. In this scope, XtratuM offers a virtualization solution capable of running mixed-purpose applications. This paper presents the key contributions to the OVERSEE project, whose architecture depends on the availability of this technology. This project has the goal of offering an appropriate infrastructure for the information technology systems of the upcoming smart vehicles.

1 Introduction

The OVERSEE project [2] aims to provide a dependable and secure infrastructure for the execution of mixed criticality applications on automotive systems. To meet this challenge, an architecture based on virtualization has been selected. XtratuM [1] offers an ideal foundation for enforcing a strong level of isolation, both temporal and spatial, thus ensuring that vehicle functionality and safety cannot be harmed by any OVERSEE application.

It is worth to mention that this project uses the Common Criteria Separation Kernel Protection Profile [3], which considers virtualization as a main technique to achieve a high level of robustness and data protection. Therefore, the benefits of virtualization in OVERSEE are reflected in two aspects, namely, to enforce rules for authorized information flows between virtual machines and to ensure a strong temporal isolation capable of running critical real-time applications.

XtratuM offers a virtualization layer very close to the native hardware. This hypervisor relies on the paravirtualization technique, which means that software running on top of it must be modified in order to interact with the hypervisor and not with the underlying hardware. Therefore, virtualization support cannot be limited to creating virtual machines but it has to be extended to provide support for any run-time environment wanted to be run over XtratuM.

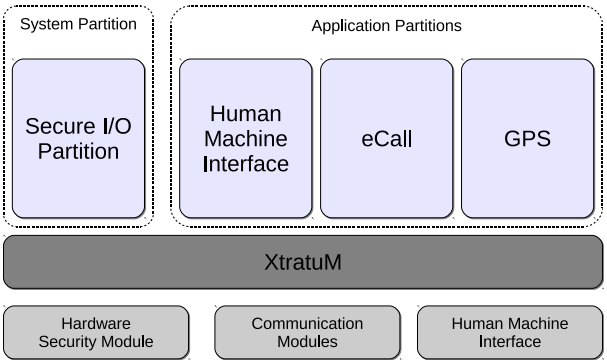

Figure 1: OVERSEE architecture overview.

Ultimately, this project has pushed forward the cloud of available facilities in order to face the design of a XtratuM-based system. The most interesting developments are presented in this paper, which are those related with run-time environment support (mainly Linux), a new technique introduced to allow dynamic scheduling over XtratuM and a novel device virtualization layer based on the use of an I/O server partition.

2 Virtualization Layer

The hypervisor abstracts the hardware and offers a software interface in order to create several virtual machines. Each of this virtual machines will

(*) This work has been partially granted by the project COBAMI: DPI2011-28507-C02-02

be referred to as a partition. Each partition is run spatial and temporarily isolated from each other, hence the term partition is very convenient to depict this very fact.

The selected processor for this project has been an Intel Atom Z530 single core 1.6 GHz. XtratuM for x86 architecture has been taken as a starting point, so the main efforts in the OVERSEE project have been focused on providing a good support for paravirtualization implementation. Linux is an increasingly popular solution for embedded systems, so it's been the primary target of the paravirtualization efforts. Thanks to the Xen group, the Linux kernel provides an interface, which simplifies enormously the task of porting the kernel to a hypervisor, known as *paravirt-ops*.

Implementing the full set of *paravirt-ops* is only half way to provide a running virtualized environment. The privilege levels of the segments need be modified, as well as several other patches inside the kernel as, for example, those regarding the Interrupt Descriptor Table (IDT) entry points.

3 Dynamic Scheduling

In order to ensure a strong temporal isolation, the scheduler relies on a cyclic plan statically defined. Besides providing a predictable behaviour, this scheduling technique has a counterpart which is the lack of flexibility. Hard real-time applications benefit from this fact, but non real-time constrained applications may be impaired when a considerable amount of CPU time is wasted.

Considering the use cases defined for the project, among which is video playback, a technique to allow dynamic scheduling has been successfully introduced, which was presented on [10]. With this new service, it is possible to define on the cyclic plan specific time windows where the dynamic scheduling of partitions is allowed. The goal has been to provide a way to improve CPU utilization by dynamically sheduling partitions.

The implemented dynamic scheduling facility relies on a scheduling technique known as Application Defined Scheduling. Basically, the hypervisor does not know anything about the scheduling policies, its only task is to perform the context switches. The decision about which partition is scheduled next is left to a trusted partition with special permissions, known as *Spare Host*.

In order to be scheduled in spare time, the partitions must send to the Spare Host a measure of the CPU utilization. For this project, the Linux kernel has been patched in order to implement a method of measuring CPU usage. Linux provides a measure of CPU which is based on an unreliable method; each time the kernel receives an interrupt it logs which was the interrupted context to establish the CPU utilization. However, this method is not exact and has been avoided in OVERSEE. Instead, the Linux kernel scheduler has been patched to have an exact measure of CPU utilization.

In order the hypervisor to decide which partition is to be scheduled next a list known as *spare plan* is used. This list contains pairs partition,duration that inform the hypervisor how it should schedule the partitions inside the defined slots of the cyclic plan. The *spare plan* is calculated by the *spare host* using whatever policy is considered convenient. Hence, the architecture of the dynamic scheduling facility can be divided in two layers. The lowermost and closest to the hypervisor layer that takes the *spare plan* and performs the context switches, and the uppermost layer taking political decisions about which partition is to be scheduled. In summary, the current available policy of the spare plan is designed for maximizing CPU usage, but other policies could be implemented as well.

4 Device virtualization

Finally, device virtualization has been the biggest part of the OVERSEE project concerning resource virtualization. Device virtualization is considered as a state of the art problem. Solutions can be found in the scope of Intel processors for mainframes [11], but no standard solutions for embedded systems are available. Therefore, the implemented device virtualization engine is a completely novel system which relies on the concept of I/O server.

Device management is a very extense subject itself and, specifically, in Intel systems. On this processor, there is an I/O address space which can be accessed by special instructions. Chunks of this space can be individually assigned to partitions so partitioning can be applied here. However, the I/O address space is only one of several ways of driv-

ing devices in Intel. The ISA bus, the standard VGA raster or the PCI address spaces are accessed via memory mapped regions that have to be controlled in a different way. Among other reasons, the PCI devices have access to DMA engines that bypass the MMU and can break spatial isolation. The developed device virtualization facility is used to overcome the problem of memory mapped devices.

The current architectures under study for device virtualization make some assumptions that do not hold to a virtualized system based on XtratuM. Specifically, they assume that there is an I/O server which has full access to the guest partition memory address space or that the system has an available IOMMU. Among these solutions can be found Virtio. Virtio [8] is the first effor to build a standard interface for device virtualization so it has been taken as a starting point.

The first implementation efforts for the device virtualization were published in [9]. This was the first attempt to implement an I/O server and provide a set of virtualized devices to Linux partitions. However, this architecture has been redesigned from scratch, in order to have a device virtualization engine capable of offering devices to any kind of partition, whether Linux based or not.

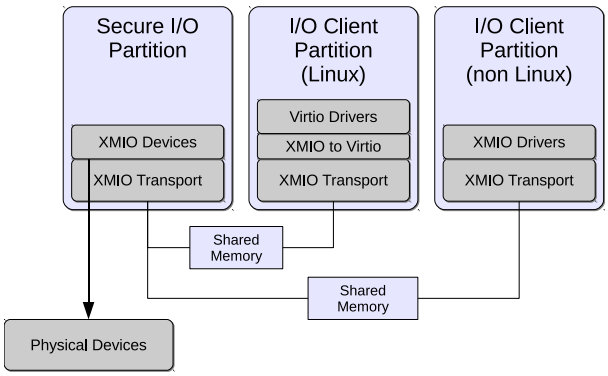

Figure 2: XtratuM device virtualization engine (XMIO) overview.

Due to the XtratuM design policies, implementing device drivers at the hypervisor level is not considered. The current version of XtratuM I/O virtualization engine (XMIO) is based on the use of shared memory. As stated, there is still no hardware support for Intel embedded processors, which binds the data transfers between I/O server and I/O clients to memory copies. This decreases significantly the device performance and increases the burden of the CPU. Efforts have been made to optimize the performance of the memory copies by using the SSE2 Intel extensions and minimize the impact of the data transfers.

5 Conclusions

This paper presents the main results regarding virtualization issues on the OVERSEE project. This project has been a success regarding the virtualization infrastructure provided, capable of being used on commercial systems. Current efforts are targeted to include additional policies for the dynamic scheduler and for the design and implementation of a new version of the device virtualization engine capable of running as a standalone application.

References

[1] XtratuM: Baseline Software Specification, http://www.xtratum.org.

[2] OVERSEE Project: Open Vehicular Secure Platform. FP7-ICT-2009-4. Project Id: 248333, 2010-12. http://oversee-project.org/

[3] Separation Kernel Protection Profile Information Assurance Directorate June 2007 http://www.commoncriteriaportal.org/files/ppfiles/pp_skpp_hr

[4] The XEN hypervisor. http://www.xen.org.

[5] The Lguest Hypervisor. http://lguest.ozlabs.org.

[6] The KVM Hypervisor, http://www.linux-kvm.org.

[7] Virtio PCI Card Specification v0.8.10 DRAFT. http://ozlabs.org/ŕusty/virtio-spec/virtio-spec.pdf.

[8] Virtio: Towards a De-Facto Standard For Virtual I/O Devices. http://ozlabs.org/ŕusty/virtio-spec/virtio-paper.pdf.

[9] Device Virtualization in a Partitioned System: the OVERSEE approach. S. Peiró, J. Sánchez, M. Masmano. Jornadas de Tiempo Real 2011. polaris.dit.upm.es/ str/jtr11/papers/019.pdf.

[10] Planificación dinamica de tiempo libre en sistemas particionados basados en XtratuM. J. Sánchez, A. Crespo, J. Simó, M. Masmano Jornadas de Automática 2011

[11] Intel Virtualization Technology for Directed I/O. Intel. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf