Ada Real-Time Services and Virtualization*

Juan Zamorano, Ángel Esquinas, Juan A. de la Puente
Universidad Politécnica de Madrid, Spain

jzamora, aesquina@datsi.fi.upm.es, jpunte@dit.upm.es

Abstract

Virtualization techniques have received increased attention in the field of embedded real-time systems. Such techniques provide a set of virtual machines that run on a single hardware platform, thus allowing several application programs to be executed as though they were running on separate machines, with isolated memory spaces and a fraction of the real processor time available to each of them.

This paper deals with some problems that arise when implementing real-time systems written in Ada on a virtual machine. The effects of virtualization on the performance of the Ada real-time services are analysed, and requirements for the virtualization layer are derived. Virtual-machine time services are also defined in order to properly support Ada real-time applications. The implementation of the ORK+ kernel on the XtratuM supervisor is used as an example.

1. Introduction

Virtualization techniques have raised significant interest in the embedded systems area. Virtualization enables a single hardware platform to be divided into a number of virtual machines, each of them providing a set of virtual resources that are mapped into the available physical resources. In this way, each virtual machine provides a partition for executing programs using a fraction of the physical processor time, memory capacity, and other devices. Since each partition is based on a virtual machine with a set of virtual devices, although with only a fraction of the capacity of the physical machine, it can host any kind of software organization, including different kinds of operating systems or run-time kernels (figure 1).

Virtualization can provide temporal and spatial separation between partitions. Temporal separation means

that each partition is guaranteed to have a fraction of the physical processor time, and no other partition can steal processor time from it. Spatial separation means that each partition is allocated a fraction of the global physical memory space in such a way that no other partition can access any part of it. In this way, applications running in different partitions are isolated from each other, and errors occurring in one of them cannot propagate to the others.

Virtualization has significant advantages for building complex embedded systems with high-integrity requirements, especially when there are subsystems with different levels of integrity. Isolation provides fault containment, and also simplifies the validation and verification process for high-integrity applications coexisting with lower integrity ones. It also enables more efficient fault detection and management techniques, and a better adaptation to the different system views that are commonplace in modern development methods. However, virtualization also creates new challenges, especially when real-time behaviour is considered. In addition to introducing some degree of execution-time overhead, multiplexing processor time among different partitions may undermine the temporal predictability of the applications. Both the implementation of the virtualization software layer and the application itself must be done in such a way that care is taken in order to keep temporal predictability, and to ensure that real-time applications running on virtual machines can be analysed for temporal correctness.

Figure 1. Virtualization and partitions.

*This work has been partially funded by MICINN, project TIN2008-06766-C03-01 RT-MODEL.

In the rest of the paper we analyse the effects of virtualization on real-time programs using the services defined in the Ada real-time annex [4, annex D]. Proposed Ada 2012 modifications are also discussed. Section 2 contains an overview of virtualization technology, with focus on bare machine supervisors. Ada real-time services are summarized in section 3, and the implications of implementing them on a virtual machine are analysed. Section 4 discusses the support needed from the virtualization layer in order to properly implement the Annex D real-time services. Finally, section 5 summarizes the porting of the ORK+ kernel to the XtratuM hypervisor as a case study. Conclusions and hints for future work are presented in section 6.

2. Overview of virtualization technology

2.1. Hypervisors

There are different approaches to virtualization, but not all of them are equally suitable for real-time systems, mostly for efficiency reasons. It is generally accepted that the best approach for embedded systems is based on the use of a hypervisor or virtual machine monitor [13]. A hypervisor is a layer of software that provides an execution environment in which several programs, including operating systems, can run in the same way as if they were executed on the real hardware. Type 1 or *bare-metal* hypervisors run directly on the physical hardware, usually in supervisor mode, whereas type 2 or *hosted* hypervisors run on top of an operating system. In order to get a predictable, efficient real-time behaviour, a bare-metal hypervisor is generally thought to be a better choice.

Hypervisors can work in two ways. When *full virtualization* is provided, the virtual machine interface is identical to the physical processor, and the code running in the partitions does not have to be modified. This requires hardware support which is not available on most embedded processors. *Paravirtualization* [6], on the other hand, is a technique in which the virtual machine is similar, but not identical, to the physical machine. This means that most machine instructions are executed on the real processor, but privileged instructions are replaced by system calls to the hypervisor. This technique requires changes in the guest operating system, but not in the application code.

2.2. Scheduling

In order to allow for real-time partitions to exhibit a predictable behaviour, as well as to ensure temporal isolation, processor time has to be multiplexed among the various partitions in a predictable way. A two-level

hierarchical scheduling scheme is often used, where a *global scheduler* allocates processor time to partitions, and a *local scheduler* is used within each partition to choose a process or thread to run when the partition is active.

Different kinds of global and local scheduling policies can be used [12]. In the rest of the paper a static global scheduling policy is assumed, as specified in the ARINC 653 standard [3] and implemented in the current version of the XtratuM hypervisor [7]. The Ada real-time scheduling policies will be used at the local level for partitions running real-time Ada applications.

2.3. Interrupt management

One of the key elements of virtualization is interrupt management. Interrupts are handled by the virtualization layer, and virtual interrupts are dispatched to partitions in a similar way as conventional operating systems dispatch hardware events to processes (figure 2).

Figure 2. Immediate event notification.

It should be noticed that in this context the notification of events to the target partition may be delayed if the partition is inactive. In this case, the delivery of the virtual interrupt is delayed until the partition is active (figure 3).

Figure 3. Delayed event notification.

It goes without saying that the dispatching of virtual interrupts has a direct impact on the performance of the real-time services provided by Ada partitions.

2.4. Virtualization interface

Hypervisors provide a virtualization interface that the guest operating systems or real-time kernels running in

partitions can use to replace the execution of privileged machine instructions. This interface usually takes the form of a set of system calls or *hypercalls*. Hypercalls give access to the hypervisor basic services: support for context switch, real-time clocks and timers, interrupt support, inter-partition communication, etc. In this way, the guest operating system can run in user mode, whereas the hypervisor is the only part of software running in supervisor mode.

In order to run Ada programs on a partition, the Ada run-time system has to be para-virtualized, i.e. it has to be modified so that the virtual hardware resources provided by hypercalls are used instead of the physical hardware.

3. Ada real-time services

3.1. Review of Ada real-time services

The Ada 2005 real-time services are specified in Appendix D of the ARM [4]. These services can be grouped into the following categories:

- Scheduling: priorities, dispatching policies, ceiling locking policy.
- Real time: real-time clock, delay until and timing events.
- Execution time: execution-time clocks and timers, group budgets.

The standard also defines the Ravenscar profile as a set of restrictions on tasking, including some restrictions on the above services.

The current Ada 2012 proposal [2] includes some additions and modifications, which can be summarized as follows:

- Support for multiple processors (AI05-0171-1); synchronous barriers (AI05-0174-1); group budgets for multiprocessors (AI05-0169-1); Ravenscar profile for multiprocessors (AI05-171-1).

The analysis in this paper is restricted to monoproductors, and therefore these real-time mechanisms will not be discussed.

- Improvements on real-time language features:
 - Scheduling: Yield for non-pre-emptive dispatching (AI-0166-1).
 - Execution time: monitoring the time spent in interrupt handlers (AI05-0170-1)
 - Fix for Ceiling_Locking with EDF (AI-055-1).

- Extended suspension objects, usable with EDF (AI05-0168-1)

There are also some other minor fixes which will not be discussed here.

3.2. Impact of virtualization on Ada real-time services

Running Ada real-time programs on top of a virtualization layer raises some problems, which are derived from the differences between the virtual machine and the underlying physical machine. The various issues related to virtualization are discussed in the next paragraphs.

Scheduling. As explained in section 2 above, a two-level scheduling scheme is assumed. In order to be able to ensure the real-time behaviour of the applications, a global scheduling method with a predictable, bounded temporal behaviour must be used. Static cyclic scheduling, as specified by ARINC 653, provides such kind of behaviour and is thus a possible choice. More flexible approaches are also possible (see e.g. [12]).

The local scheduler is used to determine which task is dispatched to run in a partition. For a partition running an Ada program, any of the task dispatching policies defined in the real-time annex can be used. However, the implementation of context switches at the lowest level of the real-time kernel has to be modified as privileged instructions have to be replaced by hypervisor calls. This will generally result in longer context switch times.

A final remark is that the local scheduler can only dispatch a task for execution when the partition is active, which may significantly delay the response time of real-time tasks, as discussed in section 3.3 below.

Real-time clock, delays and timing events. The implementation of these services relies heavily on the underlying hardware [14], and therefore has to be modified when running on a virtual machine. In this case the physical hardware timers are handled by the hypervisor, and partition code has only access to them through hypervisor calls providing clock readings and virtual timers. Both virtual machine clocks and virtual interval timers refer to physical real-time, but virtual timer interrupts may be delayed if the timer expires when the partition is not active (see figure 3). As a result, tasks may suffer significant activation jitter, which has to be taken into account for temporal analysis.

Execution-time clocks and timers. CPU time is kept on physical processors by updating a per-task execution-time counter on every context switch according to the real-time clock value [14]. However, this implementation cannot be used when running on a virtual machine, as the time intervals during which the partition is not active should not be counted. Therefore, the execution-time clocks of all the tasks in a partition have to be stopped when the partition becomes inactive, and restarted when the partition becomes active again. This requires some support from the hypervisor. The best solution is to implement partition-time clocks in the hypervisor, on top of which the local real-time kernel can base the per-task execution-time clocks.

Monitoring the time spent in interrupt handlers.

The way in which the execution time of interrupt handlers is kept is implementation-defined. A simple and legal implementation is to charge the time consumed by the interrupt handlers to the task that is running when the interrupt is generated. However, the Ada 2012 proposal allows an implementation to separately account for interrupt handling time. This case is considerably more complex, and for the moment this option is not recommended.

It should be noted that the execution time clock of any task can be charged with the execution time of any virtual interrupt delivered by the hypervisor to the partition, or even with the time spent in handling interrupts not related to the same partition. This may result in high inaccuracies in execution time measurement for any task, especially when there are non-real-time partitions with interrupts occurring at unpredictable times and unknown handling times.

Ravenscar profile. The Ravenscar profile is not affected by virtualization, except that the real-time behaviour of tasks may change due to the effects of running on a virtual machine.

3.3. Response-time analysis

Response time analysis for systems with hierarchical schedulers has been discussed by Almeida and Pereira [1], Davis and Burns [8], Pulido *et al.* [12], and Balbastre *et al.* [5], among others. Depending on the exact global and local scheduling methods that are used in a given system, a choice of techniques can be applied with various levels of accuracy. In any case, context switch and interrupt handling overheads, as well as other effects of virtualization on the temporal behaviour of the system, must be accounted for.

4. Required virtualization support

In this section we summarize the main features that the hypervisor implementing the virtualization layer must provide in order to support the execution of Ada real-time programs on one or more partitions, according to the discussion in section 3 above.

First of all, since we are assuming a paravirtualization approach, the hypervisor must provide hypercalls to replace all privileged instructions for the real processor architecture. This includes access to privileged registers, input/output instructions, interrupt support, and memory management, as well as any other processor-specific resources.

Real-time clocks and timers are basic resources for implementing the Ada real-time clock, delays and timing events. The hypervisor must provide a monotonic real-time clock base, and some timer mechanism based on it. Such basic mechanisms must be accessible by means of hypercalls, so that the Ada run-time system can use them to implement the Ada higher-level mechanisms.

Implementing execution-time clocks and timers requires a time base that only advances when a partition is active. The most efficient way to get it is that the hypervisor implements per-partition execution time clocks that measure the time spent in running each partition. Partition-time timers based on such clocks should also be implemented, and access to all of these mechanisms should be provided through appropriate hypercalls, as before.

If partition-time clocks are not provided at the hypercall level, the Ada run-time system must build an equivalent service based on lower-level services provided by the hypervisor. For example, the hypervisor might deliver a virtual interrupt to a partition whenever it is activated. If the global scheduler is a static cyclic executive, knowledge of the minor and major cycle durations could then be used to compute the duration of the interval during which the partition is inactive. The Ada run-time system could then adjust a locally maintained partition-time clock by subtracting the duration of the inactive interval from the elapsed time.

Execution-time spent in interrupt handlers raises additional problems. The Ada run-time system can account for the time spent in virtual interrupt handlers that run in the partition, but not for the time used by lower-level interrupt handlers within the hypervisor. In order not to charge it to the active partition, the hypervisor should stop the partition-time clock during the execution of the interrupt handler. If there is no partition clock at the hypervisor level, the only possibility to prevent interrupt handling time from being erroneously attributed to the running task would be to notify the occurrence and

duration of interrupt handlers in such a way that the Ada run-time system could adjust its local partition clock.

Monitoring the time spent in interrupt handlers is even harder. At the lower level, partitions can be pre-empted by interrupts generated by timers or input/output services requested by other partitions. Moreover, some interrupts may be delivered to several partitions, e.g. those generated by hardware timers supporting timing events or delays. The only way time spent in interrupt handlers can be properly accounted for is to implement interrupt clocks within the hypervisor. To our knowledge this is not done by any current hypervisor implementation.

5. Case study

XtratuM [11] is an open-source bare-metal hypervisor for real-time embedded systems. It implements most of the above requirements, with the only exception of interrupt-time clocks. The global scheduler is a cyclic executive based on the ARINC 653 specification, supporting a variety of local operating systems at the partition level. We have ported the ORK+ kernel [14] to XtratuM using paravirtualization techniques on a LEON2 platform [10], an implementation of the SPARC V8 architecture. The ORK+/XtratuM kernel acts a guest partition operating system, on top of which a Ravenscar Ada application can run (figure 4).

Figure 4. ORK+/XtratuM architecture.

The work is described in detail in another paper [9]. Paravirtualizing the kernel included adding a new Ada package for the hypercall interface, and modifying four more packages in order to replace privileged operations by XtratuM hypercalls. Overall, 1398 out of 7316 lines of code had to be modified, including the interface package and some low-level routines written in assembly language.

Evaluation experiments showed a low impact of virtualization on system performance. Although the total overhead for activating periodic tasks was found to be about 5 times the value for the original ORK+ run-

ning on a bare LEON2, the overall performance losses were found to be negligible for tasks with periods above 10 ms.

6. Conclusions and future work

The implementation of Ada real-time systems on virtual platforms has been analysed in the paper. The kind of virtualization kernel that has been taken as a reference is a bare-metal hypervisor with paravirtualization. Such kind of platform requires the Ada run-time system and the underlying real-time kernel to be modified so that it can run in user mode and access to physical devices is replaced by hypercalls giving access to virtual devices. The issues involved in such modifications have been analysed, and some requirements for the virtualization kernel have been derived. The approach has been applied to porting the ORK+ kernel to the XtratuM hypervisor. The porting has required only a moderate amount of effort and has given reasonable results in performance tests.

Planned future work includes doing a pilot implementation of interrupt time clocks on ORK+/XtratuM, and study more in depth the implications of virtualization on schedulability analysis. Another promising line is extending real-time virtualization concepts to multi-core processor platforms.

Acknowledgments

The authors wish to acknowledge the fruitful collaboration with the XtratuM team at the Technical University of Valencia.

References

- [1] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*, pages 95–103, New York, NY, USA, 2004. ACM Press.
- [2] ISO/IEC 8652:1995:AMD2(2007): *Information Technology — Programming Languages — Ada — Amendment 2, draft 11*, 2011. Available on <http://www.ada-auth.org/amendment2.html>.
- [3] ARINC. *Avionics Application Software Standard Interface — ARINC Specification 653-1*, October 2003.
- [4] ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007): *Information Technology — Programming Languages — Ada*.
- [5] P. Balbastre, I. Ripoll, and A. Crespo. Exact response time analysis of hierarchical fixed-priority scheduling. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications — RTCSA 2009*, pages 315–320, 2009.

- [6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177, October 2003.
- [7] A. Crespo, I. Ripoll, and M. Masmano. Partitioned embedded architecture based on hypervisor: The XtratuM approach. In *Dependable Computing Conference (EDCC), 2010 European*, pages 67–72, april 2010.
- [8] R. Davis and A. Burns. Hierarchical fixed priority preemptive scheduling. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium — RTSS 2005*, 2005.
- [9] A. Esquinas, J. Zamorano, J. A. de la Puente, M. Masmano, I. Ripoll, and A. Crespo. ORK+/XtratuM: An open partitioning platform for Ada. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies — Ada-Europe 2011*, number 6652 in LNCS, pages 160–173. Springer-Verlag, 2011.
- [10] Gaisler Research. *LEON2 Processor User’s Manual*, 2005.
- [11] M. Masmano, I. Ripoll, A. Crespo, and J.-J. Metge. XtratuM: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop*, Dresden. Germany., 2009.
- [12] J. A. Pulido, S. Urueña, J. Zamorano, T. Vardanega, and J. A. de la Puente. Hierarchical scheduling with Ada 2005. In L. M. Pinho and M. González-Harbour, editors, *Reliable Software Technologies — Ada-Europe 2006*, volume 4006 of LNCS. Springer Berlin/Heidelberg, 2006.
- [13] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, may 2005.
- [14] S. Urueña, J. A. Pulido, J. Redondo, and J. Zamorano. Implementing the new Ada 2005 real-time features on a bare board kernel. *Ada Letters*, XXVII(2):61–66, August 2007. Proceedings of the 13th International Real-Time Ada Workshop (IRTAW 2007).