Timing Aware Hardware Virtualization on the L4Re Microkernel System

Adam Lackorzynski[†,‡], Alexander Warg[†]

Kernkonzept GmbH[†]
Dresden, Germany

Technische Universität Dresden[‡]
Operating-Systems Group
Dresden, Germany

Email: adam.lackorzynski@kernkonzept.com, alexander.warg@kernkonzept.com
adam.lackorzynski@tu-dresden.de

***Abstract*—Hardware virtualization support has found its way into real-time and embedded systems. It is paramount for an efficient concurrent execution of multiple systems on a single platform, including commodity operating-systems and their applications. Isolation is a key feature for these systems, both in the spatial and temporal domain, as it allows for secure combinations of real-time and non real-time applications. For such requirements, microkernels are a perfect fit as they provide the foundation for building secure as well as real-time aware systems. Lately, microkernels learned to support hardware-provided virtualization features, morphing them into microhypervisors. In our demo, we show our open-source and commercially supported L4Re system running Linux and FreeRTOS side by side on a multi-core ARM platform. While for Linux we use the hardware features for virtualization, i.e., ARM's virtualized extension, we revert to paravirtualization for running the FreeRTOS guest. Paravirtualization adapts the guest kernel to run as a *native* application on the microkernel. For simple guests that do not use advanced hardware features such as virtual memory and multiple privilege levels, virtualization is simplified and the state of a virtual machine is significantly reduced, improving interrupt delivery and context switching latency. Both guests as well as the native application drive LEDs to exemplify steering actual devices as well as to show their liveliness. Taking down the Linux guest will not disturb the others.**

I. Introduction

Virtualization technology enables many interesting application scenarios, which require combining commodity off-the-shelf applications and real-time tasks in a secure, dependable and, most importantly, timing preserving manner. For example, cyber-physical systems such as autonomous cars, UAVs for wood-fire detection and SCADA systems, which besides many applications control our power grid, combine latency sensitive tasks such as model predictive control tasks for engines and road situations, flight stabilization and grid stability with maintenance tasks or other, less timing critical tasks that benefit greatly from the extended execution environments of commodity operating-systems (OSs).

Consider, for example, an autonomous driving scenario. As long as a safe exit to the emergency lane can be maintained at all points in time and in all situations and as long as this exit route can be executed entirely in the real-time subsystem, resource intensive tasks such as vision, scenery analysis and maneuver planning can remain in rich commodity environments, which speed up development time and reduce costs but sacrifice stringent timing guarantees. The aforementioned assumptions allow the real-time system to transition into a fail-safe mode for those situations where the commodity operating system ceases to respond in a timely manner.

However, for real-time tasks to operate reliably next to commodity OSs and their applications, faults in the latter must be confined and timing guarantees of the former preserved.

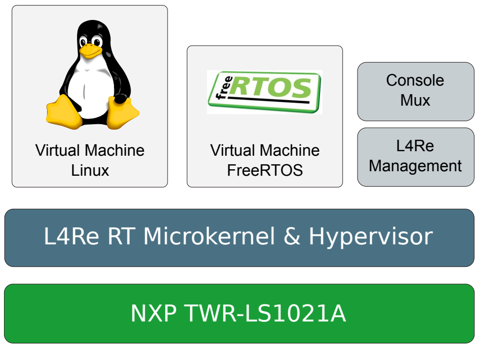

Fig. 1. Demo Setup, virtualized Linux and FreeRTOS running on an ARM platform.[1]

In this demo, we show how our open-source and commercially supported L4Re microkernel system [1], [2] exploits ARMv7's hardware virtualization capabilities to consolidate Linux and FreeRTOS on a multi-processor platform. The commodity OS Linux and the real-time kernel FreeRTOS are run independently of each other in two virtual machines (VMs), which prevents any malfunctioning of one to affect the other. While for Linux we use ARM's hardware virtualization capabilities, the FreeRTOS guest is paravirtualized.

II. The L4Re Microkernel System

L4Re is a capability-based third-generation microkernel-based system [1]. It evolved starting from the DROPS real-time system [3], later including secure system construction [4] and a major interface redesign towards a capability-driven security model [2] while keeping its real-time roots. With L4Re one can now securely isolate real-time and non real-time applications in a single system.

[1]Tux logo copyright by Larry Ewing, Simon Budig, Anja Gerwinski; FreeRTOS logo from http://www.freertos.org/

The system consists of the L4Re kernel and the L4Re user-level infrastructure that provides the necessary framework to build a wide range of applications, including services. Through its capability design and thus the inherent local naming scheme, interposing interfaces has become an essential part of the system, which allows easily exchanging and enhancing of the functionality of the system. For example, by interposing part of the scheduling interface, new core placement policies can be added and efficiently executed.

III. Hardware vs. Software Virtualization

Classical real-time operating systems, such as FreeRTOS, run on systems that do not provide hardware features for isolation, such as virtual memory and multiple privilege levels. Thus, the virtualization requirements for such guests are much simpler, allowing them to be virtualized as a native user-level process of the microkernel. In comparison to a hardware-virtualized virtual machine, the state of such a task, which is to be maintained and stored by the kernel during a context switch, is much smaller. Therefore, unless additional hardware is added to simultaneously maintain multiple virtual machines, paravirtualization reduces interrupt latency by requiring the kernel to capture only the user-level state of paravirtualized guests.

To further reduce this latency while allowing for fine grained control and scheduling of VM interrupts, we implemented the scheduling context scheme proposed in [5]. In this scheme, multiple scheduling contexts (an abstraction of time) can be attached to threads and VMs be activated upon arrival of an interrupt. Upon such an arrival, the host kernel scheduler compares the interrupt's and current scheduling context's priority to determine whether it can inject this interrupt and schedule the VM immediately or whether a higher prioritized task is present.

For commodity operating systems (such as Linux), virtualization is more involved. These systems use multiple privilege levels and virtual memory, which evades a naive virtualization using a user-level task. Although, source availability provided, it is possible to para-virtualize these systems [6], [7], hardware features for virtualization offer a major benefit as they provide additional privilege levels and virtual memory capabilities [8], [9]. However, these additional hardware features come at the cost of a larger state to be context switched, which translates to higher latencies and response times.

IV. Beyond Virtualizing Guest OSs

While virtualization is a crucial feature to isolate subsystems, virtualization by itself is not of much use. Instead, tasks, which implement a functionality while running inside the VMs, must also be able to interact with the outside world through sensors, actuators and other devices. Of course, guarantees about the timeliness of such device accesses must be preserved by the virtualization layer, in particular if part of the system becomes compromised.

To exemplify this requirement in our demo, the virtualized guest OSs (Linux and FreeRTOS) each steer an LED as an example of a more complex device and to report their health status. In addition, a native task of the host system performs the same operation to resemble scenarios where real-time functionality is implemented as native L4Re applications. Figure 1 shows this setup.

In the demo, the Linux guest can be commanded interactively and none of the activities within the Linux, including crashing the whole VM, shall affect the execution of the FreeRTOS guest or of the native application. The LEDs of the application and FreeRTOS task display the situation accordingly by showing no difference in their blinking behavior while the LED driven by Linux will eventually stop blinking. It remains in the state, which corresponds to the last setting made by Linux.

This elementary demo setup shall illustrate what is possible on modern microkernel-based systems today and inspire more sophisticated usage scenarios. We invite everyone interested to try out the open-source L4Re system, available at [1].

References

[1] "L4Re microkernel system," https://l4re.org/.

[2] A. Lackorzynski and A. Warg, "Taming Subsystems: Capabilities as Universal Resource Access Control in L4," in *IIES '09: Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems*. Nuremberg, Germany: ACM, 2009, pp. 25–30.

[3] H. Härtig, R. Baumgartl, M. Borriss, C.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter, "DROPS: OS support for distributed multimedia applications," in *Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, Sep. 1998.

[4] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter, "The Nizza Secure-System Architecture," in *In IEEE CollaborateCom 2005*. IEEE Press, 2005.

[5] A. Lackorzynski, M. Völp, and A. Warg, "Flat but trustworthy: Security aspects in flattened hierarchical scheduling," *SIGBED Rev.*, vol. 11, no. 2, pp. 8–12, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2668138.2668139

[6] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter, "The performance of μ-kernel-based systems," in *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, Saint-Malo, France, Oct. 1997, pp. 66–77.

[7] "L^{4}Linux," https://l4linux.org/.

[8] ARM Limited, *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*, ARM DDI 0406C.c ed., 2014.

[9] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D*, 325462-057US, December 2015 ed., 12 2015.