VOSySzator: A flexible embedded RISC-V system virtualizer targeting the cloud

Alvise Rigo[1]*, Daniel Raho[1], Samuele Paone[1] and Timos Ampelikiotis[1]

[1]Virtual Open Systems, France

Abstract

VOSySzator [1] is a flexible embedded system virtualizer based on rust-vmm building blocks which was originally conceived by Virtual Open Systems to target embbedded virtualization use cases. It was specifically designed to overcome challenges that embedded system integrators are asked to address when implementing fast and low-overhead virtualization solution. However, as part of the Vitamin-V 2023-2025 Horizon Europe project's action, it will be extended to support cloud use cases as well, specifically for the RISC-V architecture. The features set of VOSySzator as virtualization solution for embedded systems is of interest for the RISC-V architecture which has many similarities to existing embedded architectures like ARMv8.

Introduction

System virtualization in embedded systems is a practice that is commonly used in niche use cases and market segments or in very large markets as the automotive one. In essence, these virtualization solutions tend to be very specific, tailored to a limited set of hardware. But most of all, these solutions result to be expensive, given the high development costs of certified software. There are, however, some products belonging to other market segments that could also benefit from system virtualization like, for instance, network equipment, set-top boxes and kiosks. In such cases, due to the lack of flexible solutions that can be easily tailored to specific use cases, OEMs renounce to the extra benefits of virtualization (like snapshotting, isolation from the host kernel, file system overlays, etc.) in favor of more conventional system designs, believed to be cheaper and easier to maintain.

VOSySzator is meant to democratize virtualization in all markets, by proposing an accessible solution for system designers and integrators to introduce virtualization to all those scenarios that for the aforementioned reasons are keeping a more traditional design. By definition, an embedded system is typically a small device designed to draw as few watts as possible, although yielding performance adequate to the type of task the system is supposed to address. This introduces one of the key challenges that VOSySzator is designed to address, which is efficient and fast device virtualization. As an answer to this, VOSySzator will make heavy use of *device-passthrough*, a virtualization technique that allows to *expose* (the verbs *pass-through* or *attach* are also widely used) a device to the guest Operating Systems, allowing it to have full control of the device, with almost native performance [2].

*Corresponding author: a.rigo@virtualopensystems.com

Funded by the European Union. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the HaDEA. Neither the European Union nor the granting authority can be held responsible for them. Project number: 101093062

VOSySzator supporting RISC-V

In the context of Vitamin-V, VOSySzator will inherit support for the RISC-V architecture. Vitamin-V is an Horizon Europe project that aims to develop a complete RISC-V open-source software stack for cloud services as well as an advanced virtual execution environment. In the action of this project, rust-vmm will be ported to the RISC-V architecture and VOSySzator, which is based upon it, will support a new range of use cases and opportunities connected to the newly supported architecture as well as to the cloud domain.

Virtual Open Systems considers the fundamental features of VOSySzator as a good match for the cloud, especially for *serverless computing* where the cloud infrastructure must be able to provide fast and efficient virtual machines with minimal footprint.

Core Features

Device pass-through is the virtualization technique that guarantees the highest performance when it comes to device virtualization. As a matter of fact, the interaction between the guest CPU and the device happens directly, with no interference of the host kernel or the hypervisor. The main price to be paid as far as performance is concerned regards the delivery of the device IRQs into the guest, which is mediated by software (KVM) or hardware mechanisms and thus might induce a small overhead. VFIO [3] is the kernel architecture-independent driver which in Linux takes care of setting up all the necessary bits (mainly page tables and IRQ routing) to configure the device pass-through; it comes in two different variants,

PCI and Platform, respectively for PCI and platform (i.e., memory-mapped [MMIO]) devices. At the time of writing, rust-vmm provides already the necessary binding to the corresponding IOCTLs, but this is not enough for a system emulator.

The nature of an embedded system where MMIO devices are used poses a considerable challenge when it comes to device pass-through as each device might depend on one or more devices. To have a glimpse of this, it suffices to inspect a flattened device tree (`dtb`) which is the description of the hardware which is parsed by the kernel. Dependencies like clocks, pins, memory regions, etc. are all encoded in this file, defining a graph of dependencies interconnecting all the devices in the hardware. Except few, non-connected nodes of this graph, all the other nodes exhibit one or more dependencies that must be met also after the device has been attached to the guest virtual machine. VOSySzator proposes a novel solution that analyses such dependencies and tries to satisfy them, actuating two different functions: the first one tries to pass-through the dependencies along with the desired device. This is the most straightforward solution which might not work all the time as not all devices can be passed-through. There are some devices that either because of user's requirement or for technical reasons shall stay attached to the host (and thus bound to the host driver). For example, unbinding a clock controller from the host would harm the stability and functionalities of the whole system. In these cases, the second function is deployed, which creates a dedicated proxy inside the guest OS to *remote* the functionality of the unmet dependency from within the guest to the host via a vhost-based mechanism. In the example of the clock controller, a dedicated driver implements a set of para-virtualized `clk_ops` operations that forwards the call to the host (where the call will be actually validated and actuated).

A well-thought, user-friendly and flexible solution for device-passthrough brings benefits that are not exclusively about performance. In fact, the configuration and deployment of virtual machines that can offer the same set of devices as the host system is a decisive factor to succeed in running existing BSPs or system images in the guest system with a few to none modifications. The host system in this case can be the very same hardware that originally runs the image, but also different hardware with the same set of devices. To support this use case, VOSySzator will provide the means for the user or system integrator to define a custom physical layout of the guest memory, which is also a handy feature to overcome some limitations due to the lack of an IOMMU.

The concept of device pass-through is not only meant for embedded scenarios. Also in the cloud, specifically in HPC applications and services, VFIO can be used to expose accelerators into the guest, to grant maximum performance while ensuring the complete isolation of the HPC tasks [4, 5]. In these scenarios, VOSySzator, which aims at making the pass-through experience as integrated and smooth as possible, can help encouraging the deployment of RISC-V-based cloud HPC solutions in those cases where the availability of a few virtual machines equipped with powerful accelerators is more important than many general-purpose guests. To pursue this vision, Virtual Open Systems plans to add VOSySzator support to the well-known and widely used `libvirt` toolkit, making it a drop-in replacement for other system virtualization solutions.

Conclusions

In embedded systems, virtualization is a powerful concept that gives system designers and integrators the opportunity to explore new solutions to create new BSPs, firmwares and system images that are easier to develop, debug, test and maintain. At the same time, RISC-V is gaining more and more ground in the IoT, smart appliances, home automation, automotive, industrial and, in general, in cyber physical systems and is expected to gain popularity in cloud systems as well. In this perspective, VOSySzator aims at providing the key functionalities to design and implement system software for this new wave of devices, as well as to provide the tools to move existing software from a bare-metal to a virtualized execution, with all the added value of virtualization.

References

[1] Virtual Open Systems. *VOSySzator: Virtualization Framework for Embedded Systems.* URL: `http://www.virtualopensystems.com/en/products/vosyszator/`.

[2] Antonios Motakis, Alvise Rigo, and Daniel Raho. "Platform Device Assignment to KVM-on-ARM Virtual Machines via VFIO". In: *2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*. 2014, pp. 170–177. DOI: `10.1109/EUC.2014.32`.

[3] *Linux VFIO API.* URL: `https://docs.kernel.org/driver-api/vfio.html`.

[4] Michail-Alexandros Kourtis et al. "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015, pp. 74–78. DOI: `10.1109/NFV-SDN.2015.7387409`.

[5] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. Panda. "Performance Characterization of Hypervisor-and Container-Based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters". In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2016, pp. 1777–1784. DOI: `10.1109/IPDPSW.2016.178`.