Device Virtualization in a Partitioned System: the OVERSEE Approach

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

Abstract
This paper presents a solution for device virtualization on a hypervisor driven partitioned system. The approach is in the scope of the OVERSEE project, which is the Open Secure Vehicular Platform. This software platform is intended to act as a single access point to vehicle networks. Such system will support different types of partitions, from real time constrained to non-trusted user partitions running general purpose operating systems. On this scenario, the partitions compete for access to hardware resources, specially for the system peripherals. Mostly oriented to non real-time partitions, this paper focuses on the design of a mechanism for device virtualization based on the Linux standard Virtio. As on real time approaches, the architecture is based on the use of an I/O server, a partition which has direct access to devices and handles the requests from other partitions to access peripherals.

1 Introduction
Partitioned software architectures can represent the future of secure systems. They have evolved to fulfil security and avionics requirements where predictability is extremely important. The separation kernel proposed in [1] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference. It is precisely XtratuM the selected kernel to serve as the base execution environment for the OVERSEE partitioned system.

The Open Vehicular Secure Platform [2] (or OVERSEE, for short) is intended to serve as a single access point to vehicle networks. It will provide a protected, standardized in-vehicle runtime environment and on-board access and communication point. Applications for this platform include, between others, positioning systems, stolen vehicle tracking, traffic information, web browsing, etc. Ultimately, OVERSEE aims to create an open software platform for which everyone can develop applications and download them. Of course, this platform must ensure that the applications cannot harm each other and, especially, internal in-vehicle software applications. The heterogeneous nature of the applications being executed under OVERSEE makes XtratuM system partitioning the best way to isolate and protect applications from each other.

Figure 1: Oversee architecture overview.
Finally, for this system, the chosen target hardware is the Intel Atom architecture, which is the Intel architecture for embedded systems.

XtratuM ensures temporal and spatial isolation of the applications that run over it. This applications may vary from bare-C applications to entire operating systems as, for example, Linux. Each of this applications is refered to as a partition or guest. On this paper, we will pay special attention to Linux guests. The use of Linux as a guest to run general purpose applications is very interesting; thanks to the development efforts made by the open source community, Linux offers a vast variety of device drivers, as well as many other software mechanisms (i.e. a TCP/IP stack), which reduce implementation burden of software projects.

With the expansion of hypervisor technology, Linux has begun to host extensions for system virtualization. Linux kernel supports at least 8 distinct virtualization systems, being Xen [4], KVM [6] or Lguest [5] among them, without taking into account the relatively new XtratuM hypervisor. With such large variety of virtualization systems, a new standard has appeared to fulfill the needs of device virtualization, which until now had to be implemented by each hypervisor. This standard is known as Virtio [8]: “a series of efficient well-maintained Linux drivers which can be adapted for various hypervisor implementations”. As will be explained on section , the special features of XtratuM needs Virtio to be modified in a different way than other virtualization solutions.

The remaining of this document is structured as follows. Section 2 introduces the XtratuM hypervisor and the Linux virtualization over it. Section 3 presents the approach of the Virtio specification to provide device virtualization. Section 4 describes the requirements imposed by Virtio and how to fulfill them under the XtratuM hypervisor. Section 4.1 presents describes the implementation of the Virtio support on the Linux guests running on the XtratuM hypervisor.

2 XtratuM and Linux Virtualization

XtratuM is a bare-metal hypervisor with extended capabilities for highly critical real-time systems. The design of XtratuM has tried to apply the philosophy of the ARINC-653 standard [3] (despite not being ARINC-653 compliant). This standard is used to achieve strong isolation between running guests, so it has rigid policies about resource management.

On the time domain, XtratuM allocates CPU to partitions following a plan which is defined at configuration time, i.e., it uses a cyclic scheduler. Dynamic schedulers are avoided when systems under control are critical real-time. This ensures a predictable behaviour as well as scheduler robustness against system temporary overloads. On the other hand, this scheme narrows maximum bandwidth of non-real-time guests, which may need more relaxed scheduling policies. As explained on section 4, implementation of virtual devices have to take into account this fact in order to optimize throughput.

Besides CPU management, XtratuM isolates spatially the partitions by defining a set of accessible physical memory areas for each one. Also, physical memory areas can be defined as shared, thus being accessible by several guests. This is precisely the chosen mechanism for driver virtualization.

XtratuM uses para-virtualization, so hardware access is wrapped by means of hypercalls. Therefore, guests have to be modified so that they do not directly access hardware but they call the hypercalls. Along with the virtualization technologies, Linux kernel has evolved to offer built-in para-virtualization mechanisms. This has greatly simplified the task of porting Linux to XtratuM architecture while enhancing forward compatibility.

2.1 Virtual devices

Intel x86 architecture offers a separate address space for accessing peripherals. This address space is composed by ports and can be mapped so that they appear in the physical memory address space. Partitions can be given permissions to use some of these ports in order to have direct access to some peripheral.

When talking about system input and output (I/O), there are two paradigms for device driver access. The first approach is to leave device management to the hypervisor. However, this is not a good approach as, for each device, a driver would have to be implemented at hypervisor level and, also, XtratuM complexity would grow too much.
Thus, only a few simple drivers (console, UART) have been implemented inside XtratuM. The second approach is to leave device driver implementation to partitions themselves. This approach is more flexible as we can use device drivers already implemented, like those included in the Linux kernel. For this method to work, guests are given permissions to access some I/O ports on the configuration step.

By leaving devices to partition control, new problems arise, related to device management. If a device on the system is dedicated so that it is mapped to at most one partition, there is no problem. Nevertheless, when there is the need for sharing devices between several guests, special management has to be applied. Operating system device drivers control mutual exclusion of the threads accessing the same device, so that input/output transactions are atomic, thus ensuring a correct operation. However, there are no mutual exclusion mechanisms on XtratuM, as this would break partition isolation (partitions may control the way other partitions are executed).

A proper solution to the device virtualization problem is to create a separate secure I/O partition with exclusive access to devices. This partition virtualizes these devices for each of the user partitions. With these model, the virtual devices on the user partition side will send requests to the I/O partition for accessing real devices. Underneath the virtual devices, a software layer will provide the necessary mechanisms for transporting the requests, as the partitions are spatially isolated. Figures 1 and 3 depict the model with different detail level.

3 Virtio

The Virtio standard for Virtual device I/O [7] provides an stable and efficient mechanism for device virtualization. Originally it was designed for providing virtual device I/O to Linux guests running hosted under virtual environments, where the hypervisor (host) also happens to be a Linux system.

The Virtio design is logically composed of the three parts that are required in order to provide a device virtualization solution; that is: the device model, the driver model and the mechanism used to glue devices and drivers together.

1. The Virtio device model 3.1 provided by the I/O partition, is in charge of offering a suitable device abstraction closely resembling a typical hardware device.

2. The Virtio device drivers 3.2 used by the Linux guest partitions are in charge of managing and accessing the Virtio devices offered by the I/O partition.

3. The Virtio transport mechanism 4.2, is in charge of providing an efficient mechanism for connecting both the Virtio devices and drivers.

3.1 Virtio Devices

The Virtio device model is designed with PCI devices virtualization in mind, that is Virtio devices are very similar in several aspects to PCI devices. This similarity can be further explained by examining the typical operations performed on devices:

1. Device configuration: is performed on a "configuration memory space" associated to a Virtio device that contains: IRQ, Status, Device features and Data descriptors information, as it would be found on a PCI device.

2. Device activity notification: is performed using the extended IRQ (Interrupts ReQuests) mechanism provided by the hypervisor, used by the Virtio device to requests attention to the guest.

3. Device operations: common device operations as device data transfers are performed on buffers allocated by the guest and provided to the device (I/O Partition) which resembles programming of DMA data transfers.

3.2 Virtio Drivers

The following Virtio drivers are supported on XtratuM which are available on the guest partitions:

3.2.1 The virtio_net driver

This driver implements a virtual network device that provides TCP/IP communication to the guest partitions. virtio_net provides a virtual Ethernet network interface card that provides guest partitions point to
point communication with the host partition. This can be used to perform NAT/filtering to provide the guests access to Internet.

The chosen architecture for the virtual network can be seen on figure 2. Each of the guests has its own eth interface, which is virtually connected to vnetX, a virtual network interface on the host side. The I/O partition will act as a router, taking packets from the virtual networks and possibly routing them to other guests or the outside world through the real network.

### 3.2.2 The virtio_block driver

This driver implements a virtual block device that provides storage to the guest partitions.

virtio_block allows the guest partitions to have a virtual storage device where a standard Linux distribution can be installed and used as the root file-system.

### 3.2.3 The virtio_console driver

This driver implements a virtual console device to access the console of the guest partitions.

virtio_console is probably not targeted for the end user, but to developers to perform configuration, debugging and development tasks through the system console.

### 3.2.4 The virtio_rng driver

This driver implements a virtual RNG (Random Number Generator) for the guest partitions.

virtio_rng provides a fast RNG to speed-up the security operations like: key-generation, authentication and encryption operations where a fast RNG is required.

### 4 Virtio on XtratuM

Virtio introduces some assumptions which hold for hosted hypervisors; i.e. hypervisors where the guests are run hosted as user space processes on top of a general purpose operating system (Linux). As for example the Lguest and KVM hypervisors.

Instead XtratuM is a native (or bare machine) hypervisor thus some of the Virtio assumptions do not hold for the implementation of the Virtio under XtratuM.

---

**Figure 2: Virtual network architecture.**

Here we focus on the assumptions that affect the Virtio support on the XtratuM hypervisor. Further Virtio requirements are discussed on the Virtio paper [8].

The assumptions that are relevant for the support of Virtio under XtratuM are:

1. **Hardware device support:**

Virtio is meant to virtualize I/O devices for hosted guests under the Linux OS. The host supports (drivers) and has direct access to the hardware devices.

This assumption: **Hardware device support** is not met by the XtratuM hypervisor as it does not provide drivers for all the hardware supported by the Linux kernel.

This assumption is solved by providing the I/O partition as a Linux host partition which has both access to the physical hardware and drivers support.

2. **Shared memory:**

The host has access to all the memory of the guest.

This assumption: **shared memory** presents issues with this design, as XtratuM partitions have strong spatial isolation, there is no way to access the guest memory, unless explicitly allowed in the XM_CF configuration file.

To overcome this problem the guest defines a memory area for the Virtio device virtualization in the XM_CF file which is shared exclusively with the host partition.
This provides a level of security as it ensures that no other guests have access to the data exchanged between the guest and the host using a Virtio device.

3. **Host scheduling:**

The host operating system controls scheduling of the guest ("hosted" user process).

This assumption: **host scheduling** presents issues with this design, as XtratuM partitions have strong temporal isolation provided by the XtratuM fixed cyclic scheduling policy.

To overcome this problem the scheduling of the partitions must be carefully chosen to achieve a compromise between:

(a) Host performance: serving the I/O requests made by guest.
(b) Guest performance: which must get their requests served.

### 4.1 Virtio implementation

In order to support Virtio for Linux partitions on the XtratuM hypervisor, the Linux partitions need to be modified to make them aware of Virtio devices, this is achieved by adding a low level virtio back-end.

The Virtio back-end is in charge of the tasks of device setup and discovery, and publishing and activity notification of Virtio buffers. This operations rely on the specific mechanisms provided by the hypervisor, which makes the back-end hypervisor dependent and required on both the host and guest partitions:

**Virtio back-end (host):**

Provides Virtio **virtqueues** support to the host partition, is also in charge of providing devices to the guest partitions. This is done by maintaining the device configuration space (Virtio descriptor page).

After providing a valid device description, once the guests starts using the device the host is in charge of attending and serving all the Virtio requests made by the guest partitions.

**Virtio back-end (guest):**

Provides Virtio **virtqueues** support to the guest partition, is also in charge of the discovery/removal of the Virtio devices.

The device discovery and configuration is done as usual on a bus (e.g. the PCI bus), by reading and writing the device configuration space (Virtio descriptor page) which has been previously by the host partition.

Additionally the management of notifications of Virtio device activity are performed by means of XtratuM IPVI (Inter Partition Virtual Interrupts).

### 4.2 Virtio Transport

The Virtio Transport Ring is the mechanism used for exchanging shared buffers between the guest and the host. As stated before, Virtio initial assumptions do not match XtratuM features, so the transport mechanism has to be modified in order to share buffers.

The Virtio drivers generate lists of scatter/gather buffers (or **scatterlists**). Such lists are the mechanism used by Linux to deal with the virtual/physical memory maps. Due to this memory model, even if a buffer looks contiguous in the virtual memory map, it may be scattered through several pages of physical memory. Thus, the Linux kernel offers a mechanism to get all these scattered buffers from a pointer in virtual memory. Those buffers are then recovered from the virtio transport ring and sent to the I/O partition.

#### 4.2.1 Shared buffers

In the case of XtratuM, partitions are granted spatial isolation. Therefore, the buffers passed to the Virtio ring belong to the guest partition memory map, and are not accessible by the I/O partition. Thus, the Virtio transport ring has been modified in order to copy the scatterlists to a memory area shared between the guest and the I/O partition. The resulting model has been depicted on figure 3.

The shared memory area has to host buffers of unpredictable but bounded sizes. This memory area is managed by a dynamic memory allocator. To avoid the fragmentation problem, the allocator only gives blocks of pre-defined sizes, much like a slab allocator. Therefore, besides avoiding
fragmentation problem, its simplicity allows giving buffers in constant time.

5 Conclusions

In this paper we have presented the work done to provide device virtualization to Linux guests running on the XtratuM [1] hypervisor in the scope of the OVERSEE project [2].

The presented approach is based on an I/O partition that exclusively owns the hardware devices and performs device virtualization. The device virtualization itself is solved by using the virtual I/O device standard knowns as Virtio [8] which provides “a series of efficient well-maintained Linux drivers which can be adapted for various hypervisor implementations”.

The assumptions that Virtio performs about how the underlying hypervisor works are reviewed in order to design and achieve an efficient Virtio implementation on the XtratuM hypervisor.

6 Future work

Another way to solve the problem of making accessible the memory buffers, is to provide some page flipping mechanism in the hypervisor, which allows exclusive sharing of physical memory pages. The page flipping solution is further discussed in the Virtio paper [8].

An interesting option for providing Virtio device virtualization is the Virtio-PCI. The basic Virtio implementation uses a shared memory for the Virtio device, instead the Virtio-PCI allows the devices to be plugged to the PCI bus, provided that the virtual environment supports PCI virtualization. This is further discussed in the Virtio Specification [7].

Finally, current work only provides support for the text console, and block and network devices. Additional research is going on to implement virtual multimedia devices, capable of giving audio and video support to guest partitions. This devices have some additional complexity, as they are more interactive and user oriented and, thus, additional mechanisms have to be taken into account. For example, the user may focus his attention on a specific partition, which will take control over the video graphics adapter. The implementation of multimedia devices will expand the possibilities of XtratuM beyond critical real-time systems, to more general purpose applications.

References


[9] Linux Kernel Virtio source code.  