Grand Report Title

Nicolae-Alexandru Ivan
Faculty of Automatic Control and Computers
University POLITEHNICA of Bucharest
Splaiul Independenței 313, Bucharest, Romania, 060042
nicolae.ivan@stud.acs.upb.ro

January 15, 2016

Abstract

Insert abstract here.

1 Introduction

And there was light.

2 Virtualization

"Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others." (cite Amit Singh. An introduction to virtualization. http://www.kernelthread.com/ publications/virtualization/, 2004)

It implies having a machine on which the virtualization takes place, called a host machine, and a firmware or software, called a hypervisor or virtual machine manager, that creates the virtualized or guest machine.

The guest executes its code, but instead of running directly on the underlying hardware, it has to pass through the hypervisor, which may decide to run the instructions on the hardware or modify them beforehand.

2.1 Hypervisor types

Depending on how they function, there are two types of hypervisors, as classified in ...:

- **Type-1** also called **native** is a hypervisor that runs directly on top of hardware, while the guest machines are run as processes

- **Type-2** is a hypervisor which runs inside an operating system

2.2 Types of virtualization

In order to perform efficient virtualization, most instructions must be executed directly by the hardware. Interpreting the instructions would incur a performance drop.

However, not all instructions can be directly executed. There are two types of such instructions:

- **Control-sensitive instructions** which perform changes on hardware resources through various operations, e.g. I/O
- **Behaviour-sensitive instructions** which read the privilege level and might reveal to the guest that it is not running directly on hardware

(cite OK_Virt..)

Depending on how the framework handles sensitive instructions, there are two main types of virtualization:

1. Full virtualization
2. Paravirtualization

(cite vmware..)

2.2.1 Full Virtualization

This technique is composed of a combination of direct execution and binary translation. As seen in Figure 1, user code is executed directly, while guest OS requests are translated into instructions that modify the virtual hardware.

In this case, the guest OS does not know it is being virtualized. Thus, no modifications to the operating system are necessary. The hypervisor may cache the translated instructions for future use.

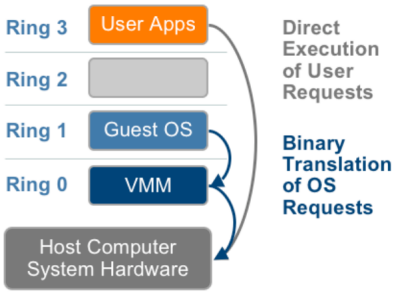

Figure 1: Full Virtualization

2.2.2 Paravirtulization

Paravirtualization means the hypervisor provides interfaces for operations that are not virtualizable. Because of this, guest OS instructions are now replaced by hypercalls which interact with the virtualization layer.

This approach incurs less performance penalty, but it requires modifications to the operating system which is to be virtualized. However, this also brings the advantage of being able to replace multiple consecutive sensitive instructions with a single hypercall, reducing the penalty induced by exceptions.

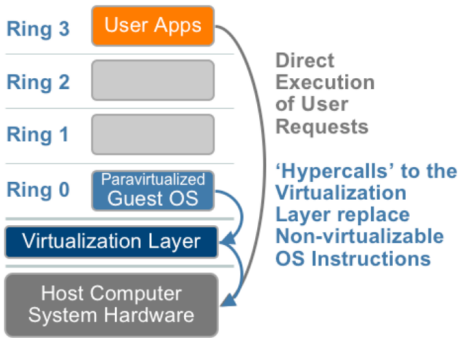

Figure 2: Paravirtualization

2.3 Hardware Assisted Virtualization

In 2006, Intel release its VT-x and AMD presented AMD-V, both introducing the notion of hardware assisted virtualization. The feature consists of a new root level of privilege in which the virtual machine manager runs. Sensitive calls trap to this level without translation or paravirtualization.

Memory is also virtualized in order to accommodate guest systems. This is done by an extra level of indirection in memory management. The memory management unit must map guest physical memory to host machine physical memory.

2.4 Virtualization on ARM

ARM virtualization works similarly to Intel and AMD, in the sense that it also introduces a new privilege level, namely the **hyp** mode. In order to virtual memory usage while in this mode, an additional memory management unit was introduced. An additional extension, called Large Physical Address Extension, was also added, extending the addressable physical memory space to 40 bits. (cite http://genode.org/documentation/articles/arm_-virtualization)

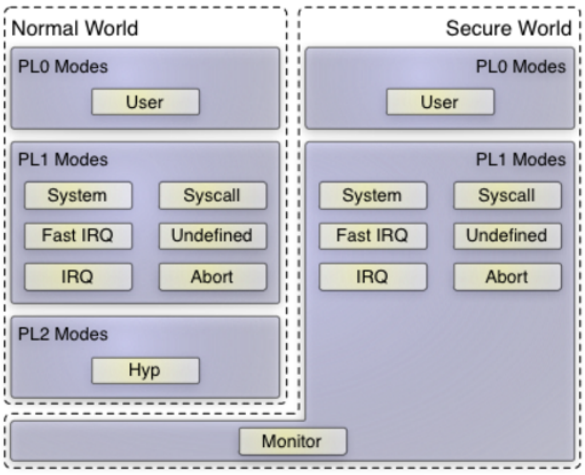

Figure 3: ARM privilege levels

2.5 Advantages of Virtualization for Embedded Systems

Advances in performance of embedded systems have made them a viable platform for virtualization. The main reason why virtualization on embedded systems is necessary is to have multiple operating systems, each responsible for some subset of functions performed by the device.

Another important aspect of virtualization is the security benefits it brings. The isolation of one guest from another means that a vulnerability in one subsystem cannot lead to an exploit in another. This limits the possibilities a potential attacker has when trying to compromise a system.

The above mentioned isolation also results in the possibility of using software distributed under different license agreements in different guests, while having efficient communication between these facilitated by the hypervisor.

2.6 Limitations of Virtualization for Embedded Systems

While performance of embedded systems has increased considerably, complexity of software systems has increased as well. The fact that each guest runs a separate operating system renders the virtual machines quite expensive from a resource point of view.

Another issue with performance is brought on by the scheduling mechanism. Due to the encapsulation of virtualized systems, the hypervisor can only associate a priority for each guest system. The result is that a low priority task running inside a high priority machine will be scheduled before a high priority task inside a lower priority guest.

Although virtualization does bring benefits in terms of security, it has some drawbacks. The increased amount of code on which a guest operating system relies on in order to function equates to increased amount of potential vulnerabilities.

3 FreeBSD and bhyve

`BSD` stands for "Berkeley Software Distribution". It is derived from AT&T's Research UNIX operating system. In 1990, the BSD code was released without the proprietary AT&T code. Since the proprietary code contained a good portion of the kernel, it was not until 1992 that a complete operating system based on the BSD code was released - 386BSD. In 1993, the FreeBSD operating system split from the project.

3.1 FreeBSD vs Linux

While FreeBSD originated from the UNIX systems, Linux was built to be a similar alternative for these. Their similarity stems from the fact that both are POSIX-compliant. Therefore, they share many mechanisms, tools and applications.

The most significat difference between the two consists in the licensing. Linux is licensed under GNU General Public License(GPL). This allows freedom to modify and redistribute the source code as long as it is also licensed under the GPL. FreeBSD is licensed under the BSD license, which is more permissive. It does not require derived work to maintain the license, but only to include a copy of the BSD license and original copyright.

Another difference consists of the available software. One aspect is availability of both packaged and sourced software. While most Linux distributions have little to no support for building software from source, FreeBSD provides an extensive collection of software source code which the user can customize and build. Conversely, while BSD maintainers are more conservative when modifying software packages, Linux distribution maintainers make modifications in order to improve component interconnection and management. (https://www.digitalocean.com/community/tutorials/a-comparative-introduction-to-freebsd-for-linux-users)

3.2 bhyve

`bhyve` is an abbreviation for BSD hypervisor. It is a type-2 hypervisor, similar to Linux KVM. As of FreeBSD 10.0, it is part of the base system. It support various guest systems, ranging from other BSD systems to Linux distributions.

Being a legacy-free hypervisor, it is reliant on the virtualization features of CPUs. It requires both CPU and memory virtualization technologies.

bhyve has several components:

- **vmm.ko** - kernel module, manages VT-x stare, context switching, guest physical memory and other aspects
- **libvmmapi** - userland API
- **bhyveload** - userspace bootloader, creates vm and does initial setup, loads guest operating system
- **bhyve** - userspace run loop, emulates stdin/stdout, tap devices, block devices
- **bhyvectl** - utility that can modify virtual machine state; can also delete VMs

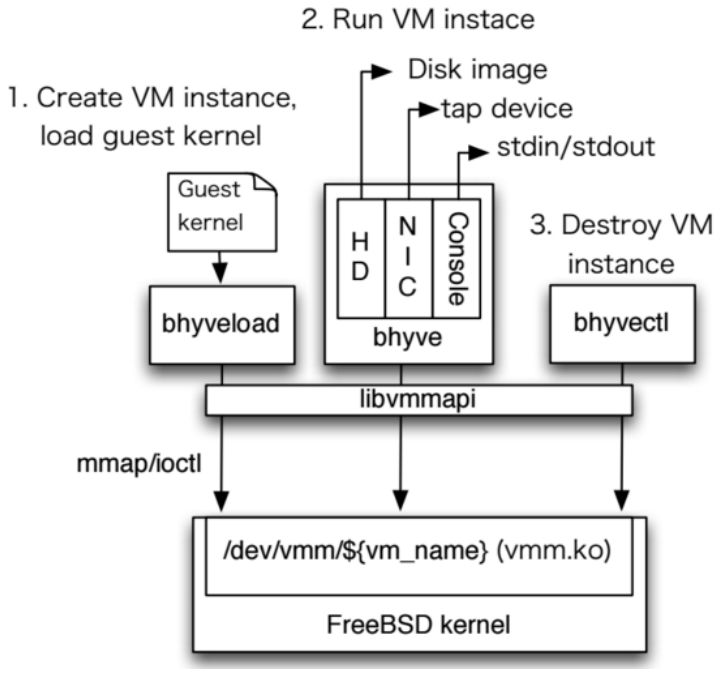

Figure 4: bhyve overview

4 Experimental Setup

In order to define recursion, one must first define recursion.

5 Scenarios and Results

The end justifies the means.

6 Conclusion and Further Work

Novel and revolutionizing approach. Aiming for world domination, as shown in .

Acknowledgment

The authors would like to thank XYZ for their support and dedication.

References

[1] A Comparative Introduction to FreeBSD for Linux Users. https://www.digitalocean.com/community/tutorials/a-comparative-introduction-to-freebsd-for-linux-users.

[2] An in-depth look into the ARM virtualization extensions. http://genode.org/documentation/articles/arm_virtualization.

[3] Explaining BSD. https://www.freebsd.org/doc/en_US.ISO8859-1/articles/explaining-bsd/index.html.

[4] Understanding Full Virtualization, Paravirtualization, and Hardware Assist, 2007.

[5] T. Asada. Introduction to bhyve. https://docs.google.com/viewer?url=http%3A%2F%2Fbhyvecon.
to_bhyve.pdf.

[6] P. Grehan. Extending bhyve beyond FreeBD guests. http://people.freebsd.org/ grehan/talks/eurobsdcon_2013_bhyve.pdf.

[7] G. Heiser. Virtualization for embedded systems, 2007.

[8] M. T. Jones. Virtualization for embedded systems, 2011.

[9] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures, 1974.

[10] A. Singh. An introduction to virtualization. http://www.kernelthread.com/publications/virtualization/, 2004.

[11] P. Varanasi and G. Heiser. Hardware-Supported Virtualization on ARM.