Secure Operating System Design and Implementation
Virtual Machines

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

January 27, 2012

Part I

Virtual Machines Overview

Overview

Definition

A Virtual Machine (VM) provides an abstract machine, that is, a software-implemented computer.

Definition

A Virtual Machine Monitor (VMM) is the software which implements a virtual machine.

- We are interested in the case in which the abstract machine coincides closely with a physical machine
- In cases where they diverge significantly, the resulting virtual machine is called an interpreter.
- We are interested here in VMMs which can support OSs

Non-interpreter VMs

Two basic levels at which non-interpreted VMs can work

Full Virtualization VM. The abstract machine architecture includes the privileged and unprivileged instructions of the underlying architecture.

Paravirtualization VM. The abstract machine architecture includes the unprivileged instructions of the underlying machine and privileged instructions must be used through a separate interface.

Unchanged	**full virtualization**	**paravirtualization**
OS	✓	
processes	✓	✓

Full vs. Para virtualization

- Full virtualization is best for existing OS, as no porting is required.
- This saves work (even if source is available)
- May be necessary (if source is not available)
- But some architectures (386) are not efficiently virtualizable
- Paravirtualization requires porting
- but may be faster
- Even when the underlying architecture supports full virtualization
- We'll next discuss architectural requirements for full virtualization

New Operating System Implementation Techniques

Hardware Implement directly on hardware
- Best performance
- Device drivers needed to support all hardware on which OS runs
- Need two computers, one for target and one for development

Full virtualization fully replicate underlying computer
- Limited number of devices, big savings in development dollars
- One computer
- better development

Paravirtualization replace privilege instruction with hypervisor calls, use pseudo devices
- Advantages of full virtualization
- Higher performance
- Security, though, depends on layering

Part II

A simple VM

A simple VM

- Consider the simplest VM
- Without any performance considerations
- Build it as an interpreter
- It executes each instruction one at a time
- It completely simulates the processor architecture
- No problem, privileged and unprivileged instructions the same
- We can run as many OSs as we want

Performance issues

- Unfortunately, the simple VM is very slow
- The danger comes from privileged instructions
- We'd like to execute unprivileged instructions at full speed
- But privileged instructions have to be handled carefully
- The problem is that privileged instructions are mixed with unprivileged instructions
- To sort them out, we can use either hardware or software techniques

Techniques to increase performance (Binary Rewrite)

- **Binary rewrite**: rewrite the instruction stream into an equivalent instruction stream without privileged instructions
- Its expensive to do the rewrite, but once its done it can be reused
- Most flexible (does not require anything of environment)
- x86 instruction stream difficult to analyze.
- E.g., difficult to tell where instructions start
- Therefore must be done at last minute (right before a branch to the instruction)
- Original technique for VMware

Increasing performance (paravirtualization)

- Alternatively, manually separate
  - privileged instructions from
  - unprivileged instructions

  And replace privileged functions with calls to trusted procedures
- Provides the ability to tune at the OS level
- This is the technique used in Xen

Increasing performance (virtualizable architecture)

- An architecture is virtualizable if it can trap the privileged instructions
- And then implement the privilege instruction in a VMM
- This enables the OS to be run in user mode
- The first architecture to support a VM was CP-40 (1967)
- And was commercialized by CP-67 (1972)
- Intel architecture was not originally not virtualizable
- Architectural support was added by to PC
- (Intel VT-X in 2005 and AMD-V in 2006)

Part III

Popek and Goldberg requirements for full virtualization

Architectures and virtualization

- Full virtualization requires the privileged instructions work. Popek and Goldberg listed a set of sufficient requirements:
- **Equivalence** A program running under the VMM should behave essentially identical to direct execution on an equivalent machine.
- **Resource control** The VMM must be in complete control of the virtualized resources.
- **Efficiency** A statistically dominant fraction of machine instructions must be executed without VMM intervention.

Popek and Goldberg sufficient requirements

- **Privileged instructions** trap if the processor is in user mode and do not trap if it is in supervisor mode.
- **Control sensitive instructions** attempt to modify the resource configuration in the system.
- **Behavior sensitive instructions** have behavior or results which depend on the configuration of resources (the content of the relocation register or the processor's mode).

Popek and Goldberg's primary result is:

Theorem

For any conventional third-generation computer, a VMM may be constructed if each sensitive instructions for that computer is a privileged instructions.

Part IV

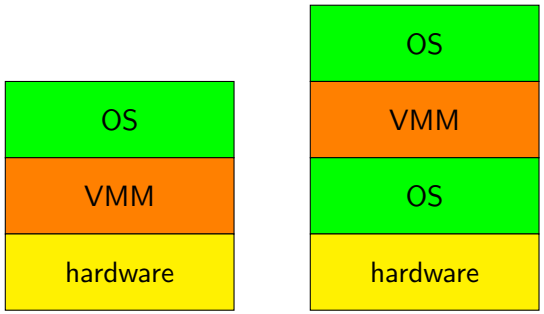
Bare metal vs. hosted VM

Paravirtualization and alternatives

there are two ways of structuring a virtual machine

Bare Metal (or Type 1) virtualization can run the VMM directly on the underlying hardware (e.g., Xen)

Hosted (or Type 2) virtualization runs the VMM on top of an existing OS (e.g., VMware workstation)



Comparison of bare metal vs. hosted VMs

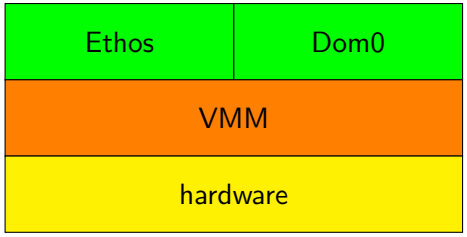
- Bare metal VMMs do not rely on the security of an underlying OS (since there is no underlying OS)
- OSs can be 500x more code than a bare metal VMM
- Bare metal VMMs can support multiple OSs at one time
- Hosted VMMs are easier to field, just like an application

Hosted VMMs

- Hosted VMMs run in user space, and thus care must be taken with privileged instructions
- To deal with privileged instructions
  - just-in-time compilation (**binary rewriting**)
  - virtualizable architecture
- VMware uses both techniques

Bare metal VMM

- Xen is a bare metal hypervisor
- The size of the hypervisor is minimized
- Which means that it jettisons as much functionality as it can
- It does this by locating device support an one of the OS's, called Dom0, which run on top of the hypervisor
- Dom0 is a privileged OS as it has access to I/O devices



Hypervisor/Dom0 split

- Hypervisor provides ensures memory isolation between OSs
- Xen is paravirtualized, OS makes explicit Hypervisor Calls, rather than implicitly trapping privileged instructions into the Hypervisor.
- Provides a pseudo-interrupt facility called Xen Events
- Provides communication between DomUs (such as Ethos) and Dom0
  - XenBus: for setting up name value pairs
  - RingBuffers: for building communication between OSs
- Dom0 implements
  - An ethernet driver for DomU
  - A block device (disk driver) for DomU
- a relatively low privileged DomU is like implementing part of OS in process

Part V

Implementing an OS on a VMM

Effect of VMMs on OS and Applications

- It may appear that VMMs, and in particular, fully virtualizing VMMs would have little effect on OS
- But that's not the case
- The largest impact is that multiple OSs can be run concurrently on a computer
- Which means that a new OS which supports only one application is feasible since using the new OS does not preclude using older OSs
- It also means that application (e.g., browsers) do not need to support multiple OSs
- This simplifies applications
- And removes the need for lowest denominator

Effect on OS design and Implementation

- OSs can support higher level semantics
- OSs needed not be universal (e.g., realtime, parallel processing)
- OSs need only support devices of the VM
  - A VM exports basically one device of each type
  - This eliminates the need to implement a large number of devices drivers (device drivers typically constitute 2/3rds or more of the OS code).
  - Device drivers constitute much more of the cost of an OS
- OSs on a VMM are largely independent of the underlying hardware
- A new OS can use facilities from another OS on the same host

Security Impact

- Bare metal VMMs are more secure than hosted VMMs, since they don't depend on an underlying OS
- Devices on Bare metal VMM are managed by another OS,
- Disk and Network devices can be sent encrypted data
- Hence, Dom0 can only effect availability of Disk and Network
- This solution does not work with I/O to humans, and so these remain vulnerable to Dom0, unless the device hardware encrypts
- Longer term. the trend is towards specialized OSs will be spun off to manage individual devices, for example a Video OS which will run an Nvidia display.

Ethos

- Ethos is built on top of Xen, a bare metal VMM
- It uses (for now) Dom0's filesystem to save implementation time
- It is specialized (does not support realtime, parallelism)
- It uses Xen's pseudo devices (for networking and console)
- It will implement only authenticated and encrypted networking
- It will implement strong authorization

Conclusions

- Building an OS on top of a VMM reduces the number of device drivers which need to be built, by far the largest cost of a traditional OS implementation
- It enables multiple OSs to be run simultaneous
- It provides a better debugging environment
- All development can be done on a single computer