Hypervisors

Introduction

- Credits:
  - P. Chaganti – Xen Virtualization – A practical handbook
  - D. Chisnall – The definitive guide to Xen Hypervisor
  - G. Kesden – Lect. 25 CS 15-440
  - G. Heiser – UNSW/NICTA/OKL

Agenda

- Introduction to virtualization
- Techniques to implement virtualization
- The role of virtualization in embedded systems
- A (quick) overview on the Xen Hypervisor

Introduction

- Virtualization is a technique of partitioning the resources of a single computing platform into multiple segregated, virtualized, execution environments.
- Each environment runs independently of the other, thus allowing multiple operating systems to run on the same hardware.

Introduction

- The concept of virtualization already present in every-day computing…
- Most modern operating systems contain a simplified system of virtualization;
- Each running process is able to act as if it is the only thing running. The CPUs and memory are virtualized.

Introduction

- Virtualization of the CPU: If a process tries to consume all of the CPU, the operating system will preempt it and allow other processes to execute;
- Virtualization of the memory: a running process has its own virtual address space that the operating system maps to physical memory to give the process the illusion that it is the only user of RAM.

Introduction

- Virtualization of the CPU: If an OS tries to consume all of the CPU, the hypervisor will preempt it and allow other processes to execute;
- Virtualization of the memory: a running OS has its own virtual address space that the hypervisor maps to physical memory to give the process the illusion that it is the only user of RAM.

Introduction

- Each execution environment is called a *guest* and the computing platform on which they execute is called the *host*.
- The software enabling these multiple execution environments is commonly referred to as **Hypervisor** or Virtual Machine Monitor (VMM).
- The Hypervisor runs on the host and acts as a *bridge* between the host and the guests;

Mixed OS Environment

- Multiple VMs can be implemented on a single hardware platform to provide individuals or user groups with their own OS environments

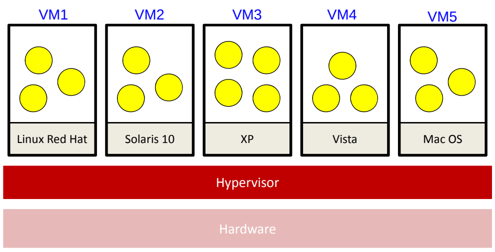


Figure: G. Kesden

Mixed OS Environment

- Virtualization implies a two-level hierarchical scheduling framework

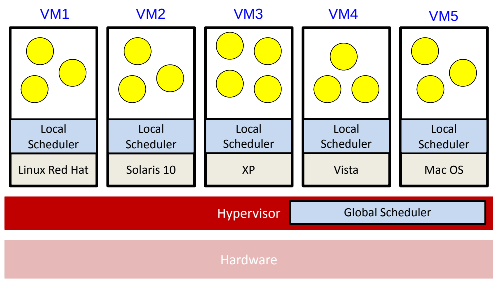


Figure: G. Kesden

Benefits of Virtualization

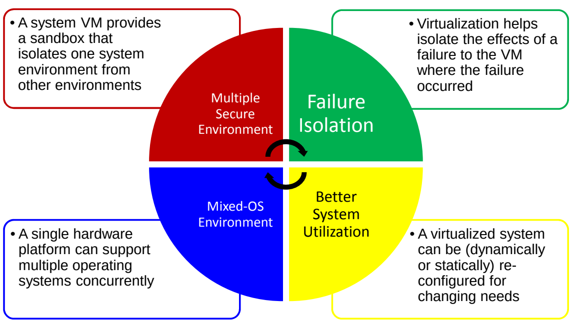


Figure: G. Kesden

Virtualization Properties

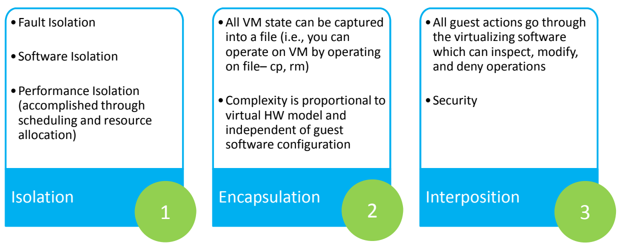


Figure: G. Kesden

Methodologies

Three main methodologies used for providing virtualization:

- System Emulation
- Paravirtualization
- Binary Translation
- OS Level Virtualization

Methodologies

System Emulation – All the hardware resources are emulated.

- The guest operating system can be run without any modification;
- It can use the hardware resources through the hardware emulation layer;
- The VMM executes the CPU instructions that need more privileges than are available in the user space.

Methodologies

System Emulation – All the hardware resources are emulated.

PRO

- Complete isolation
- Total portability (VMs are not related to any specific HW platform)
- No modifications to the OS are needed

CONS

- Slow! (Since everything is emulated)

Methodologies

Paravirtualization – No hardware emulation.

- The operating system that runs on a guest needs to be a modified version that is aware of the fact that it is running inside a hypervisor;
- Lower number of privileged CPU instructions that need to be executed;
- Typically paravirtualization of device drivers is also needed

Methodologies

Paravirtualization – No hardware emulation.

PRO

- More efficient than System Emulation
- Virtualized OSes can directly communicate with hardware resources

CONS

- Need to modify the OS!
- Isolation is more challenging

Methodologies

Hardware-assited Paravirtualization – higher efficiency thanks to special CPU instructions

- CPUs are fully aware of the presence of a virtualization stack
- CPUs provide an Instruction Set Architecture that simplifies the development of a VMM
- Automatic trap of sensitive instructions
- Automatic space isolation (i.e., memory areas) to improve efficiency

Methodologies

Binary Translation – intercept OS code

- Run-time translation of some OS instructions
- User-level code is directly executed on the real hardware
- No modifications to the OS are needed: the guest OS is not aware of virtualization
- Specific device drivers are required

Methodologies

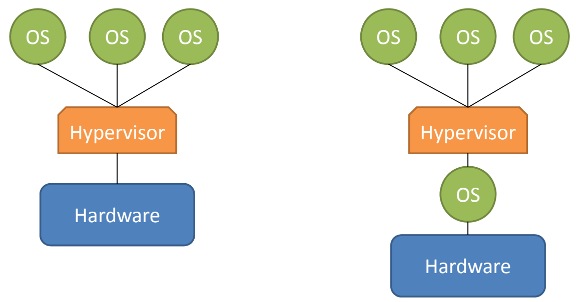
OS Level Virtualization – Each guest is isolated and runs in a secure environment.

- Only multiple instances of guests that run the same operating systems as the host;
- Close to sandboxes;
- Low run-time overhead.
- E.g., FreeBSD Jails, Solaris Zones

Types of Hypervisor

- Gerald J. Popek and Robert P. Goldberg – “Formal Requirements for Virtualizable Third Generation Architectures”, 1974
- Type 1: native (bare-metal) hypervisors
  - The Hypervisor runs directly on the host's hardware to control the hardware and to manage guest operating systems.
  - E.g., Xen, VMWare ESXi, Microsoft Hyper-V
- Type 2: hosted hypervisors
  - These hypervisors run on a conventional operating system just as other computer programs do.
  - E.g., VMWare Workstation, VirtualBox

Types of Hypervisor



Type-1 (*bare-metal*)   Type-2 (*hosted*)

Implementation

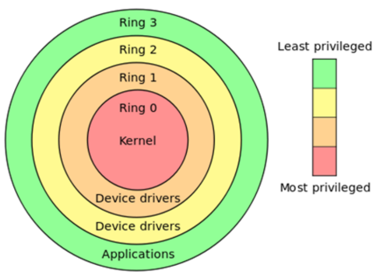
Preliminaries

- Sensitive instructions = those that attempt to change the configuration of resources in the system
  - Examples: update virtual to physical memory mappings, communication with devices, manipulation of global configuration registers, etc.
- Privileged instructions = those that are executed in privileged mode (protected, ring 0,…) and trap if executed in user mode

Implementation

Preliminaries

Example: Privileged rings in x86



Implementation

"Trap and Emulate"

- Raise of an exception (trap) when the guest executes a privileged instruction (e.g., accessing a physical resources);
- The exception handler is used to invoke the hypervisor code.

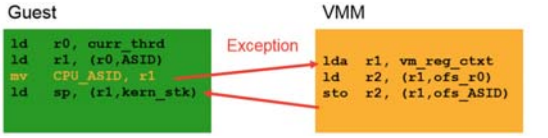


Figure: G. Heiser

Implementation

"Trap and Emulate"

Popek and Goldberg, 1974

"For any conventional third-generation computer, an *effective* VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions."

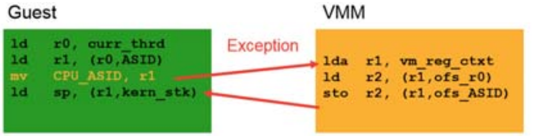


Figure: G. Heiser

Implementation

"Trap and Emulate"

Popek and Goldberg, 1974 – In other words…

It is sufficient that all the instructions that could affect the correct functioning of the VMM (sensitive instructions) always trap and pass control to the VMM.

Figure: G. Heiser

Implementation

"Trap and Emulate"

Most common architectures are not virtualizable according to definition of Popek and Goldberg

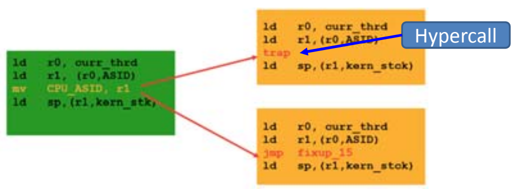
- x86 – lots of unvirtualizable features
  - E.g., PUSH of PSW (Processor State Word) is not privileged
- MIPS – mostly virtualizable, but…
  - Kernel registers k0,k1 (needed to save/restore state) are user-accessible
- ARM – mostly virtualizable but…
  - Some instructions are undefined in user-mode

Implementation

Impure Virtualization

Change the Guest OS code replacing sensitive instructions

- Paravirtualization – by trapping code (hypercalls)
- Binary translation - In-line code emulation (run-time)



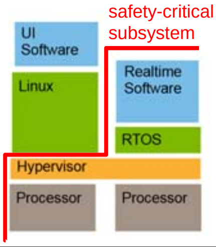
Embedded Systems

- Virtualization historically used for easier sharing of expensive mainframes.
- Gone out of fashion in 80's and resurrected in recent years for improved isolation in modern computing systems.
- Why virtualization for Embedded Systems?

Embedded Systems

Certification Issues

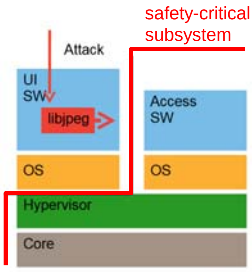
- Encapsulation of a safety-critical subsystem that can be certified independently of the other subsystems running on the same platform



Embedded Systems

Security

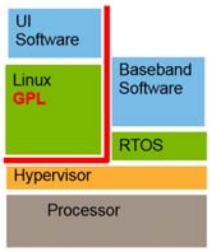
- Protection against exploits;
- E.g., software attacked by UI exploits
  - It is possible to compromise the core SW from an attack of the UI SW
  - Virtualization protects this kind of attacks ensuring a separation into different VMs



Embedded Systems

License Separation

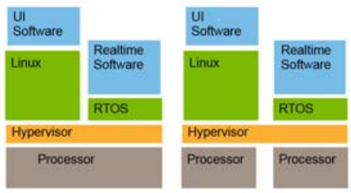
- System composed of Linux + proprietary SW (not open-source)
- VMs can be used to isolate Linux



Embedded Systems

Software-Architecture Abstraction

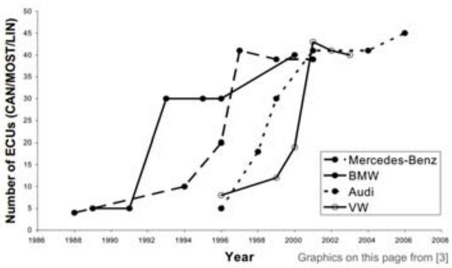
- Support for product series: same software running on different hardware;
- Decoupling from the real hardware.
- Benefits
  - Time-to-market;
  - Engineering cost.



Embedded Systems

Automotive Case-Study

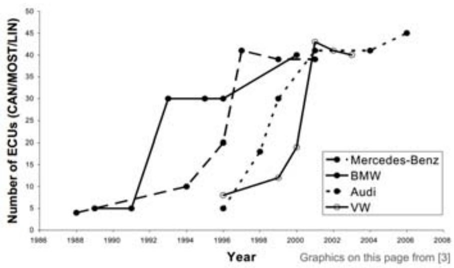
- Proliferation of ECUs: more than doubled in 10 years



Embedded Systems

Automotive Case-Study

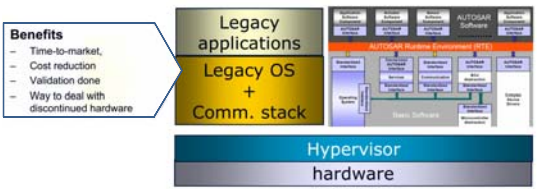
- Trend: Integration in fewer, more powerful, ECUs



Embedded Systems

Automotive Case-Study

- Thanks to virtualization it is possible to re-use a complete legacy ECU software



An Overview on The Xen Hypervisor

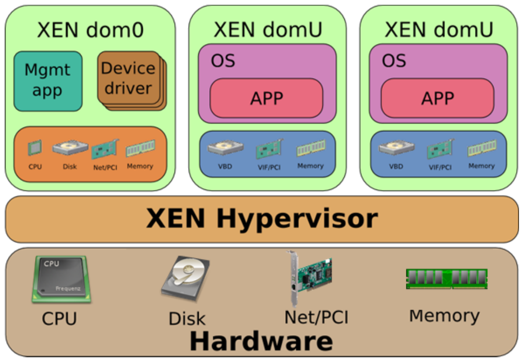
The Xen Hypervisor

- What is Xen?

  "Xen is an open-source paravirtualization technology that provides a platform for running multiple operating systems in parallel on one physical hardware resource"

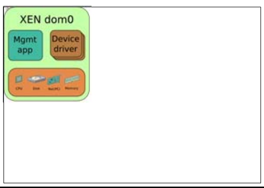
- Originally developed in 2003 at the University of Cambridge Computer Laboratory

The Xen Hypervisor



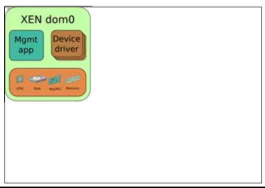
The Xen Hypervisor

- Xen refers to each virtual machine that runs on a system as a domain.
- When Xen boots up, it first starts the hypervisor, which is responsible for starting a domain named Domain0 (*dom0*) in which a specific host operating system runs.



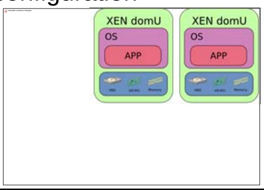
The Xen Hypervisor

- Domain0 is a privileged domain that can access the hardware resources and can manage all the other domains (e.g., create, destroy, save, restore, etc.)



The Xen Hypervisor

- An Unprivileged Domain (*domU*) guest is more restricted.
- Typically not allowed to perform hypercalls that directly access to the hardware.
- Not able to manage other domains or the hypervisor configuration



The Xen Hypervisor

- Xen is based on para-virtualization
- Requires modification of the guest OS
  - Insertion of hypercalls to replace privileged instructions;
  - Time virtualization
  - but…

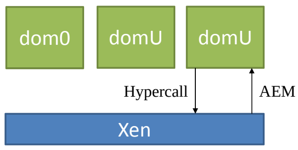
The Xen Hypervisor

Supports Hardware-assisted virtualization

- Newer processors have a set of instructions that makes virtualization easier
  - x86: Intel VT-x and AMD Pacifica (AMD-V)
  - The CPU provides traps for certain privileged instructions;
  - Enable Guest OSes to be run without paravirtualization modifications (e.g., old OSes like Windows XP)

The Xen Hypervisor

- Domain → Xen
  - Hypercall (synchronous)
- Xen → Domain
  - Asynchronous Event Mechanism (AEM) that replaces device interrupts



The Xen Hypervisor

- The Xen hypervisor is the basic abstraction layer of software that sits directly on the hardware below any operating systems.
- It is responsible for CPU scheduling (VCPU to CPU assignment) and memory partitioning of the various virtual machines running on the hardware device.

The Xen Hypervisor

- Xen currently supports 4 VCPU schedulers
  - Credit
  - Credit2 — Proportional Fair Share (e.g.,Weighted Round-Robin)
  - RTDS — Global EDF with Reservation Servers
  - ARINC 653 — Fixed Time Slices

The Xen Hypervisor

- Xen does not provide any device drivers.
- It has no direct knowledge of networking, external storage devices, video, or any other common I/O functions found on a computing system
- But provides a mechanism by which a guest operating system can be given direct access to a physical device…

How does the I/O work in Xen?

The Xen Hypervisor

I/O in Xen

- dom0 is a privileged domain that can access all the hardware in the system
- The OS running on dom0 has the device drivers and performs I/O operations on behalf of unprivileged guest domains (domU);
- Shared memory is used for the communication between a domU and dom0

The Xen Hypervisor

Thank you!

Alessandro Biondi
alessandro.biondi@sssup.it