Introduction to Virtualization

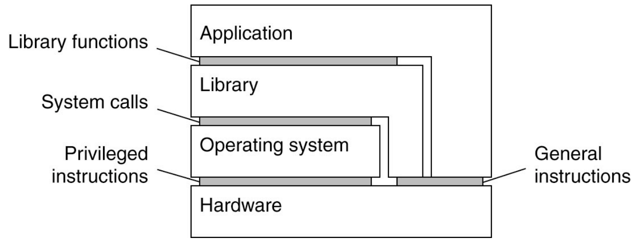
Prashant Shenoy

Virtualization

(a) (b)

- Virtualization: extend or replace an existing interface to mimic the behavior of another system.
  – Introduced in 1970s: run legacy software on newer mainframe hardware
- Handle platform diversity by running apps in VMs
  – Portability and flexibility

Types of Interfaces



- Different types of interfaces
  - Assembly instructions
  - System calls
  - APIs
- Depending on what is replaced /mimiced, we obtain different forms of virtualization

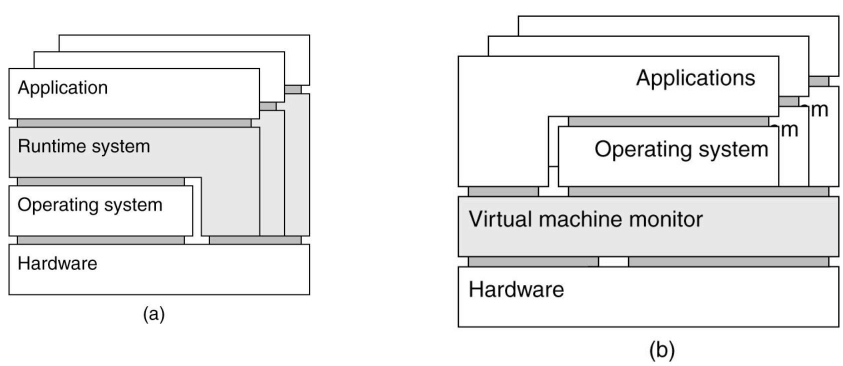
Types of Virtualization

- Emulation
  - VM emulates/simulates complete hardware
  - Unmodified guest OS for a different PC can be run
    - Bochs, VirtualPC for Mac, QEMU
- Full/native Virtualization
  - VM simulates "enough" hardware to allow an unmodified guest OS to be run in isolation
    - Same hardware CPU
  - IBM VM family, VMWare Workstation, Parallels,…

Types of virtualization

- Para-virtualization
  - VM does not simulate hardware
  - Use special API that a modified guest OS must use
  - Hypercalls trapped by the Hypervisor and serviced
  - Xen, VMWare ESX Server
- OS-level virtualization
  - OS allows multiple secure virtual servers to be run
  - Guest OS is the same as the host OS, but appears isolated
    - apps see an isolated OS
  - Solaris Containers, BSD Jails, Linux Vserver
- Application level virtualization
  - Application is gives its own copy of components that are not shared
    - (E.g., own registry files, global objects) - VE prevents conflicts
  - JVM

Examples



- Application-level virtualization: “process virtual machine”
- VMM /hypervisor

The Architecture of Virtual Machines

J Smith and R. Nair

IEEE Computer, May 2005

Slides courtesy of Bhuvan Urgaonkar

Goal of Paper

- Provide a taxonomy of virtual machines
  - Different goals
  - Different implementations

Early Computers

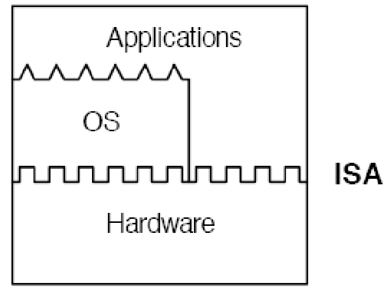
- Hardware designed
  - Software written for hardware
- Each system crafted with own instruction set
  - Software had to made specifically for each instruction set
- Eventually instruction sets became more standardized
  - However, software still requires a certain instruction set architecture and operating system that meets strict standards.

Virtual Machines

- Eliminate real machine constraint
  - Increases portability and flexibility
- Virtual machine adds software to a physical machine to give it the appearance of a different platform or multiple platforms.
- Benefits
  - Cross platform compatibility
  - Increase Security
  - Enhance Performance
  - Simplify software migration

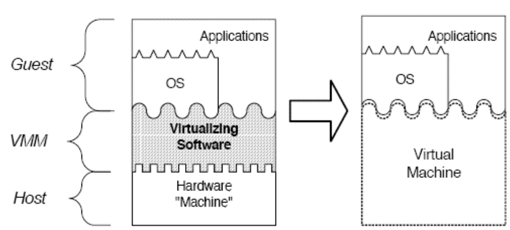
Initial Hardware Model

- All applications access hardware resources (i.e. memory, i/o) through system calls to operating system (privalaged instructions)
- Advantages
  - Design is decoupled (i.e. OS people can develop OS separate of Hardware people developing hardware)
  - Hardware and software can be upgraded without notifying the Application programs
- Disadvantage
  - Application compiled on one ISA will not run on another ISA..
    - Applications compiled for Mac use different operating system calls then application designed for windows.
  - ISA's must support old software
    - Can often be inhibiting in terms of performance
  - Since software is developed separately from hardware.. Software is not necessarily optimized for hardware.



Virtual Machine Basics

- Virtual software placed between underlying machine and conventional software
  - Conventional software sees different ISA from the one supported by the hardware
- Virtualization process involves:
  - Mapping of virtual resources (registers and memory) to real hardware resources
  - Using real machine instructions to carry out the actions specified by the virtual machine instructions



System/Process Virtual Machines

- Can view virtual machine as:
  - System virtual machine (i.e. think cygwin)
    - Full execution environment that can support multiple processes
    - Support I/O devices
    - Support GUI
  - Process virtual machine
    - Virtual machines can be instantiated for a single program (i.e. think Java)
    - Virtual machine terminates when process terminates.

Standard Interfaces

- When implementing virtual machines there are two standard interfaces
  - Deal with Process and System Level virtual machines
    - ISA -> has both user and system instructions
      - User instructions available to both the application programs and to the operating system
    - Application Binary Interface (ABI)
      - Composed of two components
        - » First all user instructions
        - » System call interface -> allows to work with OS privalaged instructions

Process Level Virtual Machines

- Provide user with application level virtual ABI environment
  - Examples
    - Multiprogramming
      - Provide end users with illusion of having a complete machine to itself
        - » Each process given own address space and access to file structure
    - Emulation and Binary Translators
      - Use interpretation to allow a program to be emulated on an ISA that is different then the ISA it was compiled on. (translate instruction when called into foreign ISA)
        - » Can also use translation to put foreign code in to the current machines ISA.
    - High Level VMS
      - When process VM at the same time you design the high level language.
        - » First done in Pascal.. Take high level code and translates it into intermediary language. Intermediary language is then translated to the specific ISA.

System Level Virtual Machines

- Provide complete environment in which many processes, possibly belonging to multiple users can exist.
  - Virtual machine is the interface to the ISA
- Divide a single set of hardware among multiple guest Operating Systems.
  - Reason -> different people want different operating systems.
  - Provides security
  - Can configure hardware by monitoring performance
    - Statistics allow it to configure hardware

Virtualization

- The computational function carried out by a computer system is specified in terms of:
  - architected state (registers, memory)
  - instructions
    - cause changes in the architected state.
- Today often more implementation state then architecture state
- How do you virtualize a foreign ISA
  - E.x. A foreign architecture maybe have 32 registers but your architecture only has 8 registers.
  - This means that a virtual machine may not map to an ISA efficiently.

Operating System Support for Virtual Machines

- Samuel T. King, George W. Dunlap and Peter M. Chen
- Proceedings of the 2003 USENIX Technical Conference

- Slides: courtesy of Bhuvan Urgaonkar

Outline

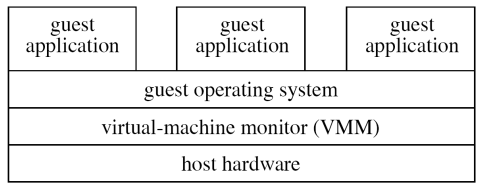
- Introduction
- Review of Virtual Machines
- UMLinux – an evaluated Type II VMMs
- Host OS Support for Type II VMMs
- Performance Results
- Conclusions

Introduction

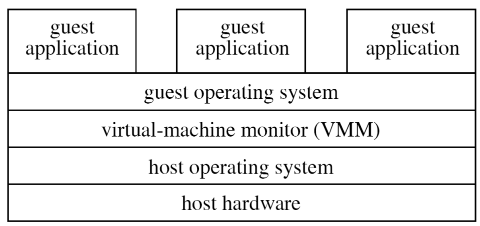
- About Virtual Machine Monitor (VMM)
  - A layer of software emulating hardware of a complete computer system.
  - Provide an abstraction – virtual machine (VM).
  - Could provide a VM identical to underlying hardware platform running VMM or totally different hardware platform.
- Uses of VMMs
  - To create illusion of multiple machines on a single physical machines.
  - To provide software environment for OS debugging.
  - To provide means of isolation that untrusted applications run separately.

Introduction

- Two types of VMMs
  - Type I



  - Type II



Virtual Machines

- The classification of VMMs can be based on whether the VM created by a VMM emulates the same underlying hardware.
  - VMs emulating the underlying hardware (homogeneous)
    - Some performance problems due to enumeration overheads, additional complexity in term of frequent task switches and memory mapping.
  - VMs emulating different hardware (heterogeneous)
    - Various degree of compatibility:
      - Denali supports only some instructions.
      - Microkernel provides high-level services that are not provided by hardware.
      - Java VM is completely hardware independent.

Virtual Machines

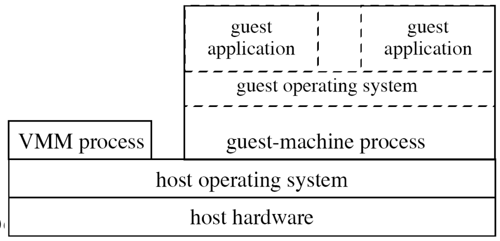
- Another classification based on Type I/II VMMs
- This paper focuses on homogeneous Type II VMMs:
  - Pros:
    - Run as a process that system developers/administrators can have an easier control on it.
    - As a debugging platform
  - Cons:
    - Undesirable performance due to lack of sufficiently powerful interfaces provided by underlying operating systems.
    - That's work to be presented in this paper.

UMLinux

- What is UMLinux?
  - UMLinux is a Type II VMM , a case Type II VMM studied in this paper
  - It runs upon Linux and the guest operating systems and guest applications run as a single process.
  - **Note**: The interfaces provided by UMLinux is similar but not identical to underlying hardware, so modifications on both guest OS and VMM are needed.
  - It makes use of functionality supplied by underlying OS, e.g.
    - process as CPU,
    - Host memory mapping and protection as virtual MMU
    - Memory files as file systems etc.
    - files and devices as virtual devices,
    - TUN/TAP devices as virtual network,
    - host signal as virtual interrupts,

UMLinux

- UMLinux system structure
  - A VMM process and a guest-machine process

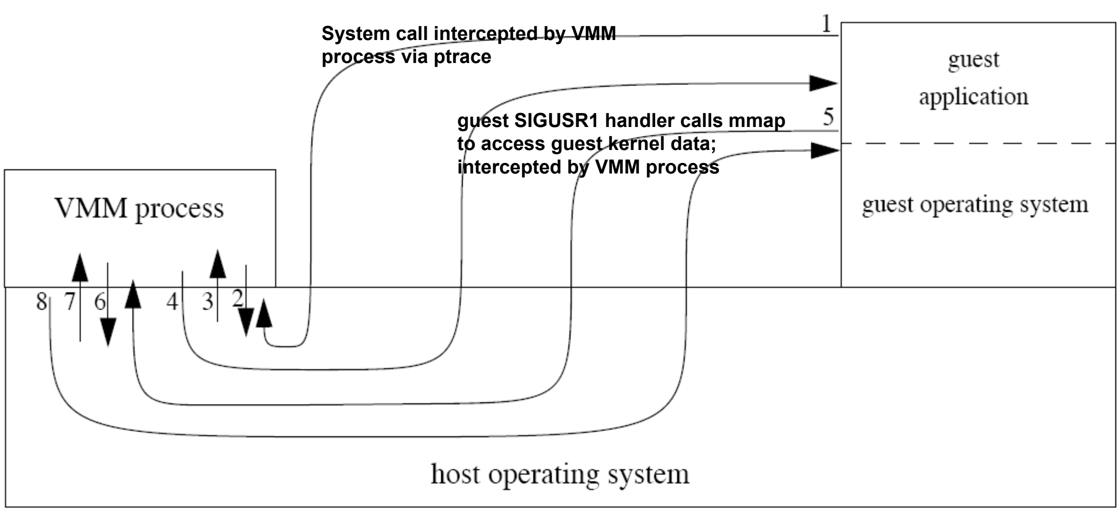


  - VMM pro
    - Redirects operating signal and system calls
    - Restricts the set of system calls allowed by guest OS
    - VMM uses "**ptrace**" to mediate access between guest machine process and host OS.

* **ptrace** is a system call to observe and control another process, and examine and change its core image and registers. It is primarily used to implement breakpoint debugging and system call tracing.

UMLinux

- UMLinux operations
  - Example:



Host OS support for Type II VMMs

- Three bottlenecks in running a Type II VMM
  - Inordinate number of context switches between processes.
  - A large number of memory protection operations.
  - A large number of memory mapping operations.

  - This paper proposed possible modifications to VMM and in general, the modifications involves only a few number of lines of code.