Virtualization

OS Lecture 21

UdS/TUKL WS 2015

Examples of Virtualization

» Qemu, Bochs, VMWare, Xen, Linux KVM, Microsoft Hyper V, Virtual PC, …

» Java virtual machine (VM), Python, JavaScript, .NET Runtime, …

» The UNIX process environment is a virtual machine.

» Linux syscall emulation on FreeBSD.

What is Virtualization?

- *hypervisor* & *virtual machine monitor* (VMM) provide a **translation** and **isolation** layer
- imitate (exclusive) platform *X* on top of (shared) platform *Y*
- *X* may or may not correspond to any actual hardware platform (e.g., Intel x86, Java VM)
- *X* and *Y* may or may not be the same (e.g., x86 on x86, x86 on PowerPC, JVM on any platform)
- Software for *X* may or may not be aware of the fact that *X* is not real, and may or may not know what *Y* is

Types of Virtualization

1. ***Process*** *virtual machine*: provide an (idealized) platform for the execution of a single program.
   - » typically provides high-level abstractions
   - » UNIX processes, Java VM, .NET VM, etc.
2. ***System*** *virtual machine*: provide a platform for the execution of a complete OS
   - » typically mimics existing hardware platform
   - » but can also provide higher-level interfaces

Benefits and Uses of System Virtualization

- **isolation**: by default, VMs share nothing
  → security, reliability, quality of service
- **configuration** and dependency management
- **server consolidation**: save energy and hardware costs
- **snapshots**: "freeze" a copy of a live VM, continue execution later
- **service elasticity**: quickly deploy many more pre-configured VMs in case of a load spike
- **reliability**: can *migrate* live VMs away from failing hosts without service interruption

Development and Research Uses

- hardware **prototyping**: test hardware that doesn't (yet) exist
- parallel **driver development**: have driver ready when hardware is ready
- kernel **debugging**: single-step kernel code and easily recover memory contents after crash
- **deterministic replay**: can precisely record and replay external inputs
- **sandbox**: can run & investigate code from untrusted sources (e.g., suspected malware)

Approach 1: Simulation

To virtualize *X* on top of *Y*:

» Write a program for *Y* that *simulates* an *X* machine.
→ Example: Qemu can simulate ARM on x86

» Essentially an **interpreter** for *X* machine code…

» …and a simulator for essential platform devices (disk controller, network, memory, BIOS)

» **Advantages**: flexible, versatile, always possible, unmodified guest OS

» **Disadvantage**: very slow (despite JIT compilation, etc.)

Approach 2: Emulation (aka Full Virtualization)

To virtualize *X* on top of *X*:

- Let guest OS execute *directly* on physical CPUs, but in *un*privileged mode. When guest OS tries to execute privileged instruction, it will *trap* into hypervisor.
- Trap is relayed to VMM, which can then check and **emulate** the effects of the privileged instruction, after which native execution resumes
- **Advantages**: fast, often within a few percent of native execution; unmodified guest OS
- **Disadvantages**: can only support native architecture (e.g., x86 on x86, but not ARM on x86), frequent traps are slow.

Challenge: Fail-Silent Instructions

What if some instructions behave differently in kernel and user mode, but don't cause traps in user mode?

- Fundamentally need traps to emulate correct behavior; otherwise fidelity of emulation not guaranteed.
- **binary rewriting**: edit kernel binary before or during execution to replace all fail-silent op codes (e.g., replace with illegal instructions to force trap)
- Fail-silent instructions make it more difficult to virtualize a platform both efficiently and transparently.

Approach 3: Para-Virtualization

To virtualize *a variant* of *X* on top of *X*:

- In contrast to simulation and full virtualization, para-virtualization is *not transparent* to the guest OS.
  - The OS needs to *cooperate* by making *hypercalls* instead of using privileged instructions.
- **Advantages**: most efficient form of virtualization (can *batch* hypercalls)
- **Disadvantage**: not transparent (e.g., Windows does not support the Xen para-virtualization ABI)

Challenges and Inefficiencies

- How should the **idle loop** be realized in a guest OS?
- **Lock-holder preemption** (LHP) problem: *what if spin lock in guest OS kernel is held by a virtual CPU (vCPU) that was preempted by hypervisor scheduler*?
- Cross-VM **interference**: contention for shared caches, shared memory bus, I/O bandwidth can cause substantial performance fluctuation.
- Why is virtualization used as a security mechanism? What if VMs **attack** the hypervisor?
- What if **hypervisors attack** the VM (e.g., to steal secrets)? (→ *Intel SGX extensions*)