19 Virtual machines

A *virtual machine* is an efficient, isolated duplicate of a real machine (computer), implemented by software.

The software running on a virtual machine is confined to the resources and abstractions provided by the virtual machine.

The virtual machine may or may not immitate any real hardware.

System virtual machine: provides a platform for the execution of a complete operating system. Examples: VMWare, Xen.

Process virtual machine: provides a platform for the execution of a single program (process). Example: Linux process, Java VM, .NET VM.

System virtual machines: Implemented by a software component called a *virtual machine monitor (VMM)* or *hypervisor*. A VMM can run on the bare hardware (native VM) or on top of an operating system (hosted VM).

Advantages:

- Can run multiple system images (operating system environment plus applications) on the same physical machine. Can run multiple instances of the same OS, different versions or configurations of the same OS or different OSes.
- Virtual machines are strongly isolated from each other. This is relevant for reliability, security and for quality of service.
- The machine implemented by the VMM can differ from any real hardware. This can be useful to experiment with hardware that does not (yet) exist.
- The VMM can record precisely the execution and current state of a VM. This is useful for debugging and for migrating an active VM to another machine; it can even be used to make an (unmodified) VM fault-tolerant or accountable. Thus, running an existing OS environment inside a VM can lend this OS new capabilitites.

Virtual machine implementation: Simplest implementation is to *simulate* the

VM's instruction set in software. This is very flexible (can simulate any machine regardless of the underlying hardware), but very expensive: 10-1000x overhead.

Efficient implementations use *emulation*. This requires that the underlying real hardware has mostly the same instruction set as the VM.

An emulated VM executes directly on the hardware, but when the VM uses certain instructions that require virtualization, the machine traps into the VMM. Instructions that must be virtualized include those that access hardware devices (device registers), change the machine state (user/kernel mode), or change the TLB or page tables.

Machines that are designed with virtualization in mind make this much easier and efficient. Intel's x86 architecture, for instance, makes it hard. It has instructions that fail silently when executed at user mode, instead of causing an exception. Such instructions must be replaced (via binary rewriting) to invoke a trap instead.

Paravirtualization: One way around this is to define a VM that doesn't correspond exactly to the x86 architecture. This is called paravirtualization (as opposed to full virtualization) and is (or can be) used in most modern VMMs. By disallowing instructions that are hard to virtualize, can simplify the VMM implementation and improve performance. However, this means that an existing OS must be modified to run on the paravirtualized virtual machine.

Another advantage of paravirtualization is that hosted operating systems can be aware of the virtualization. For instance, they can be made aware of the distinction between virtualized and real time, and they can yield a CPU (rather than halting it) when it has not work to do.

Recent x86 CPU have added explicit support for virtualization (AMD-V, Intel VT-x). This makes virtualizing the x86 architecture more efficient.

Modern VMM implementations can achieve performance very close to that of the physical machine, particularly with paravirtualization.

Uses of VMMs: VMM have been used on IBM mainframes since the 1970s.

Server consolidation: VMMs are increasingly used in data centers. VMMs allow operators to run different services (possibly even from different customers) on the same physical hardware, thus increasing hardware utilization without sacrificing isolation, and without imposing a particular operating system environment on customers.

Pickled VMs: A booted, initialized VM can be suspended and its state stored on disk. In case of a load spike, VMs can be started up from the pickled state very quickly, adding replicated servers.

VM Migration: VMs can be migrated among physical machines, which is useful for load balancing, and to ensure continued availability of services even as machines are being taken down for scheduled maintenance. Moreover, VM migration enables the quick start-up of additional server replicas in case of a load spike.

Fault tolerance: Can record the exact execution of a VM and stream the trace to a second VM (running on different hardware), which replays the trace. When the first VM fails, the second can take over. This is called primary-backup replication.

Also, VMMs are used on personal computers to support multiple OS environments (e.g. Linux and Windows), or to isolate untrusted software (e.g. Peer-to-peer apps). In corporate environment, VMMs are often used on desktops to permit the use of legacy custom software (e.g. based on Windows 95) alongside with modern office software based on XP/Vista/Win7.

For more detail, see for instance “Xen and the Art of Virtualization” (SOSP 2003).