

## 17 Virtual machines

A *virtual machine* is an efficient, isolated duplicate of a real machine, implemented by software. In practice, the virtual machine may not correspond to 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: Java VM, .NET VM. We don't cover these here.

**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 operating system environments 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. Thus, running an existing OS environment inside a VM can lend this OS new capabilities.

**Virtual machine implementation:** Simplest implementation is to *simulate* the VM's instruction set. (Example: MIPS machine simulator in Nachos). 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 hardware has mostly the same instruction set as the VM.

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. (Intel's x86 architecture makes it hard!). One way around this is to define a VM that doesn't correspond exactly to the x86 architecture. This is called *paravirtualization*. By throwing out instructions that are hard to virtualize, can simplify the VMM implementation and improve performance. However, this means that an OS must be modified to run on the paravirtualized virtual machine.

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.

VMMs are increasingly used in data centers. VMMs allow operators to run services of different customers on the same hardware, thus increasing hardware utilization without imposing a particular operating system environment on customers and without sacrificing isolation.

Moreover, VMs can be migrated among physical machines, which is useful for load balancing. Moreover, VM migration ensures continued availability of customer services even as machines are being taken down for scheduled maintenance.

Also, VMMs are used on personal computers to support multiple OS environments (e.g. Linux and Windows). 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 or Vista.