Are Virtual Machine Monitors Microkernels Done Right? Xen and the Art of Virtualization

I. Background

Virtual machines:

- Observation: instruction-set architectures (ISA) form some of the relatively few well-documented complex interfaces we have in world
- Machine interface also includes the meaning of interrupt numbers, programmed I/O, DMA, etc.
- Anything that implements this interface can execute the software for that platform
- A *virtual machine* is a software implementation of this interface (often using the same underlying ISA, but not always)

Examples:

- IBM initiated VM idea to support legacy binary code; i.e. support the interface to an old machine on a newer machine
- Apple does this to run old Mac programs on newer machines (with a different ISA)
- MAME is an emulator for old arcade games (5800+ games!!) that actually execute the game code straight from a ROM image.
- Modern VM research started with the Disco project at Stanford, which ran multiple virtual machines on a large shared-memory multiprocessor (since normal OS's couldn't scale well to lots of CPUs). This led to VMWare.
- VMWare: lots of uses including customer support (support many variations/versions on one PC using a VM for each), web hosting, app hosting (host many independent low-utilization servers on one machine -- "server consolidation")

VM Basics:

- The real master is no longer the OS but the "Virtual machine monitor" (VMM) or "hypervisor" (hypervisor > supervisor)
- OS no longer runs in the most privileged mode (which is reserved for the VMM)
- But OS thinks it is running in most privileged mode and still issues those instructions?
  - Ideally, such instructions should cause traps and the VMM then emulates the instruction to keep the OS happy
  - But in x86, some such instructions fail silently! VMWare solution: dynamically rewrite binary code to replace those instructions with traps; Xen solution: rewrite the OS slightly to avoid them
  - x86 has four privilege "rings" with ring 0 having full access. VMM = ring 0, OS = ring, app = ring 3 (x86 rings come from Multics, as do x86 segments)

Virtual Machines

How accurate is the emulation?

- Original paper on whether or not a machine is virtualizable: Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". *Communications of the ACM* 17 (7): 412 –421.
- Disco/VMWare/IBM: Complete: runs unmodified OSs and applications
- Dinali: introduced the idea of "paravirtualization" -- change interface some to improve VMM performance/simplicity
  - Must change OS and some apps (e.g. those that use segmentation)
  - But can support 1000s of VMs on one machine...
  - Great for web hosting
- Xen: change OS but not applications (support the full *application binary interface* (ABI)
  - Faster than full VM -- supports ~100 VMs per machine
- Moving to a paravirtual VM is essentially porting the software to a very similar machine

II. Disco: Emulate the MIPS interface

1) emulate R10000

- simulate all instructions: most are directly executed, privileged instructions must be emulated, since we won't run the OS in privileged mode (Disco runs privileged, OS runs supervisor mode, apps in user mode)
- an OS privileged instruction causes a trap which causes Disco to emulated the intended instruction
- map VCPUs onto real CPU: registers, hidden registers

2) MMU and physical memory

- virtual memory -> virtual physical memory -> machine memory
- VTLB is a Disco data structure, maps VM -> PM
- TLB holds the "net" mapping from VM -> MM, by combining VTLB mapping with Disco's page mapping, which is PM -> MM
- on TLB instruction on the VCPU, Disco gets trap, updates the VTLB, computes the real TLB entry by combined VTLB mapping with internal PM->MM page table (taking the permission bits from the VTLB instruction)
- Must flush the real TLB on VM switch
- Somewhat slower:
  - OS now has TLB misses (not direct mapped)
  - TLB flushes are frequent
  - TLB instructions are now emulated
- Disco maintains a second-level cache of TLB entries: this makes the VTLB seem larger

than a regular R10000 TLB. Disco can thus absorb many TLB faults without passing them through to the real OS

3) I/O (disk and network)

- emulated all programmed I/O instructions
- can also use special Disco-aware device drivers (simpler)
- main task: translate all I/O instructions from using PM addresses to MM addresses

Optimizations:

- larger TLB
- copy-on-write disk blocks
  - track which blocks already in memory
  - when possible, reuse these pages by marking all versions read-only and using copy-on-write if they are modified
  - => shared OS pages and shared executables can really be shared.
- zero-copy networking along fake "subnet" that connect VMs within an SMP. Sender and receiver can use the same buffer (copy on write)

III. Xen: emulate x86 (mostly)

Key idea: change the machine-OS interface to make VMs simpler and higher performance

- Called *paravirtualization* (due to Denali project)
- Pros: better performance on x86, some simplifications in VM implementation, OS might want to know that it is virtualized! (e.g. real time clocks)
- Cons: must modify the guest OS (but not its applications!)
- Aims for performance isolation (why is this hard?)
- Philosophy: divide up resources and let each OS manage its own; ensures that real costs are correctly accounted to each OS (essentially zero shared costs, e.g., no shared buffers, no shared network stack, etc.)

x86 harder to virtualize than Mips (as in Disco):

- MMU uses hardware page tables
- some privileged instructions fail silently rather than fault; VMWare fixed this using binary rewrite --- Xen by modifying the OS to avoid them

Step 1: reduce the privilege of the OS

- "hypervisor" runs with full privilege instead (ring 0), OS runs in ring 1, apps in ring 3
- Xen must intercept interrupts; converts them to events posted to shared region with OS
- need both real and virtual time (and wall clock)

Virtualizing virtual memory:

Virtual Machines

- x86 does not have software TLB
- good performance requires that all valid translations should be in HW page table
- TLB not “tagged”, which means address space switch must flush TLB
- 1) map Xen into top 64MB in all address spaces (and limit guest OS access)
- 2) guest OS manages the hardware page table(s), but entries must be validated by Zen on updates; guest OS has read-only access to its own page table
- Page frame states: PD=page directory, PT=page table, LDT=local descriptor table, GDT=global descriptor table, RW=writable page. The type system allows Xen to make sure that only validated pages are used for the HW page table.
- Each guest OS gets a dedicated set of pages, although size can grow/shrink over time
- Physical page numbers (those used by the guest OS) can differ from the actual hardware numbers; Xen has a table to map HW->Phys; and each guest OS has a Phy->HW map. This enables the illusion of physically contiguous pages.

Network:

- Model: each guest OS has a virtual network interface connected to a virtual firewall/ router (VFR). The VFR both limits the guest OS and also ensure correct incoming packet dispatch.
- Exchange pages on packet receipt (to avoid copying); no frame available => dropped packet
- Rules enforce no IP spoofing by guest OS
- Bandwidth is round robin (is this “isolated”?)

Disk:

- virtual block devices (VBDs): similar to SCSI disks
- management of partitions, etc. done via domain0
- could also use NFS or network-attached storage instead

Domain 0:

- nice idea: run the VMM management at user level
- easier to debug and hypercall checks catch potential errors (narrow interface!)

IV. Microkernels vs VMMs

Microkernel idea: create a small OS kernel and then run other aspects of the traditional OS, such as the file system, in other processes.

- Simple modular OS components
- Requires good IPC performances (to communicate among the now separate parts of the OS)
- Mach is one version, Windows started this way as well (due to Mach), but slowly moved pieces back into one monolithic OS (e.g. graphics)

Virtual Machines

- Small TLBs also hurt microkernels, since more processes need to be resident at once (but benchmarks were done with few processes so this didn't affect architecture much!)
- VMM view: divide up into essentially *non-communicating* pieces and switch among them -- no need for good IPC performance and *no dependencies* among the pieces
- Interprocess dependencies reduce reliability in practice: who is responsible for all of these modules? can you really make your own module effectively in practice?
- Xen: focus thus on *dividing* up resources, not managing them!
- Parallax is a file system that runs in another domain, more like a mounted file system