EECS 262a
Advanced Topics in Computer Systems
Lecture 19

Xen/Microkernels
April 4th, 2016

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

- Xen and the Art of Virtualization
  P. Barham, B. Dragovic, K Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield. Appears in *Symposium on Operating System Principles* (SOSP), 2003
- Are Virtual Machine Monitors Microkernels Done Right?
  S. Hand, A. Warfield, K. Fraser, E. Kotsovinos, D. Magenheimer. Appears in *Proceedings of the 10th conference on Hot Topics in Operating Systems* (HotOS), 2005

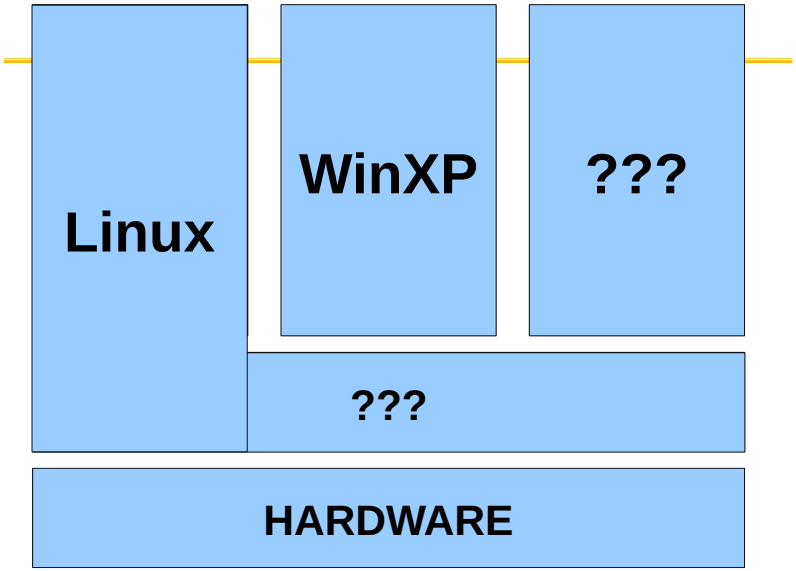
- Thoughts?

Why Virtualize?

- Consolidate machines
  - Huge energy, maintenance, and management savings
- Isolate performance, security, and configuration
  - Stronger than process-based
- Stay flexible
  - Rapid provisioning of new services
  - Easy failure/disaster recovery (when used with data replication)
- Cloud Computing
  - Huge economies of scale from multiple tenants in large datacenters
  - Savings on mgmt, networking, power, maintenance, purchase costs
- Corporate employees
  - Can choose own devices

Virtual Machines Background

- Observation: instruction-set architectures (ISA) form some of the relatively few well-documented complex interfaces we have in world
  - Machine interface includes 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)
  - 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.



Many VM Examples

- IBM initiated VM idea to support legacy binary code
  - Support an old machine's on a newer machine (e.g., CP-67 on System 360/67)
  - Later supported multiple OS's on one machine (System 370)
- Apple's Rosetta ran old PowerPC apps on newer x86 Macs
- MAME is an emulator for old arcade games (5800+ games!!)
  - Actually executes the game code straight from a ROM image
- Modern VM research started with Stanford's Disco project
  - Ran multiple VM's on large shared-memory multiprocessor (since normal OS's couldn't scale well to lots of CPUs)
- VMware (founded by Disco creators):
  - Customer support (many variations/versions on one PC using a VM for each)
  - Web and app hosting (host many independent low-utilization servers on one machine – "server consolidation")

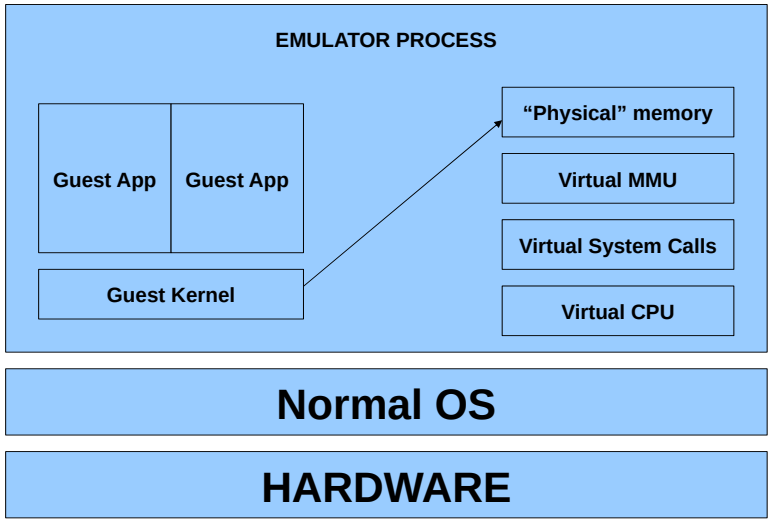
VM Basics

- The real master is no longer the OS but the "Virtual Machine Monitor" (VMM) or "Hypervisor" (hypervisor > supervisor)
- OS no longer runs in most privileged mode (reserved for VMM)
  - x86 has four privilege "rings" with ring 0 having full access
  - VMM = ring 0, OS = ring 1, app = ring 3
  - x86 rings come from Multics (also x86 segment model)
    - » (Newer x86 has fith "ring" for hypervisor, but not available at time of paper)
- 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 (old) x86, some such instructions fail silently!
  - Five solutions: SW emulation (Disco), dynamic binary code rewriting (VMware), slightly rewrite OS (Xen), hardware virtualization (IBM System/370, IBM LPAR, Intel VT-x, AMD-V, SPARC T-series)

Virtualization Approaches

- Disco/VMware/IBM: Complete virtualization – runs unmodified OSs and applications
  - Use software emulation to shadow system data structures,
  - Dynamic binary rewriting of OS code that modifies system structures, or
  - Hardware virtualization support
- Denali introduced the idea of "paravirtualization" – change interface some to improve VMM performance/simplicity
  - Must change OS and some apps (e.g., those using segmentation) – easy for Linux, hard for MS (requires their help!)
  - But can support 1,000s of VMs on one machine...
  - Great for web hosting
- Xen: change OS but not applications – support the full *Application Binary Interface* (ABI)
  - Faster than a full VM – supports ~100 VMs per machine
  - Moving to a paravirtual VM is essentially porting the software to a very similar machine

How to Build a VMM 1: SW Emulation (Disco)



Disco: Emulate the MIPS interface

1) Emulate R10000

2) MMU and physical memory

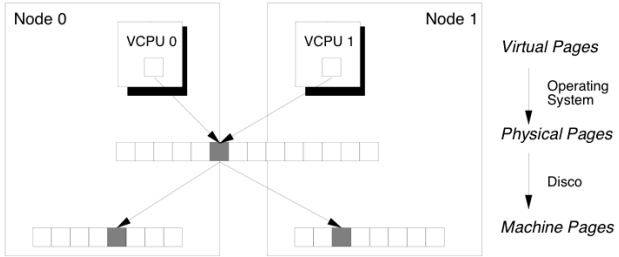
3) I/O (disk and network)

Edouard Bugnion; Scott Devine; Kinshuk Govil; Mendel Rosenblum (November 1997). "Disco: Running Commodity Operating Systems on Scalable Multiprocessors". *ACM Transactions on Computer Systems* **15** (4).

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 (I)



- Virtual memory ➔ virtual physical memory ➔ machine memory
- VTLB is a Disco data structure, maps VM ➔ VPM
- TLB held the "net" mapping from VM ➔MM, by combining VTLB mapping with Disco's page mapping, which is VPM ➔ MM

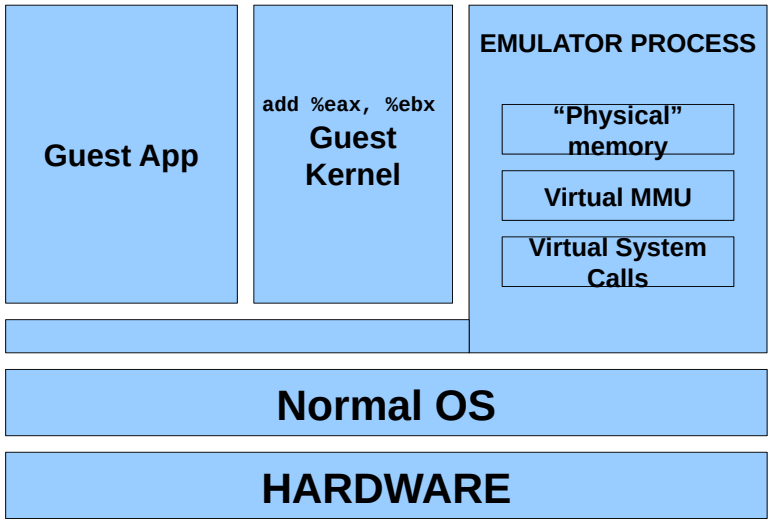
2) MMU and physical memory (II)

- On TLB modification 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

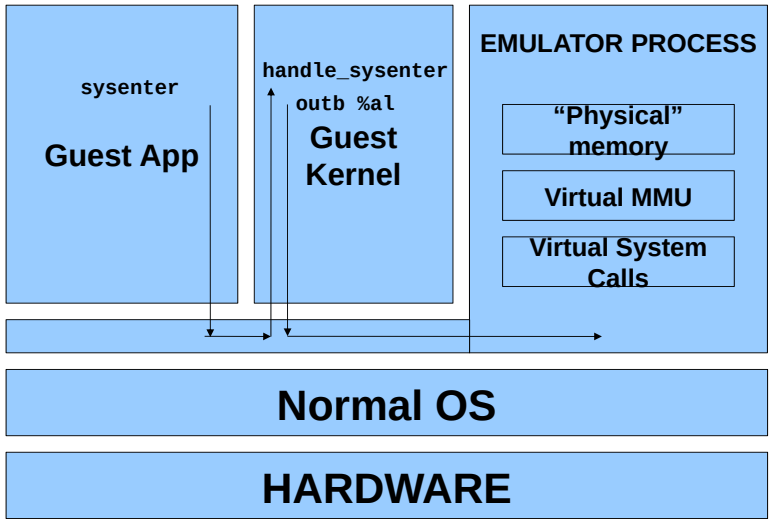
3) I/O (disk and network)

- 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)

How to Build a VMM 2: Trap and Emulate



How to Build a VMM 2: Trap and Emulate

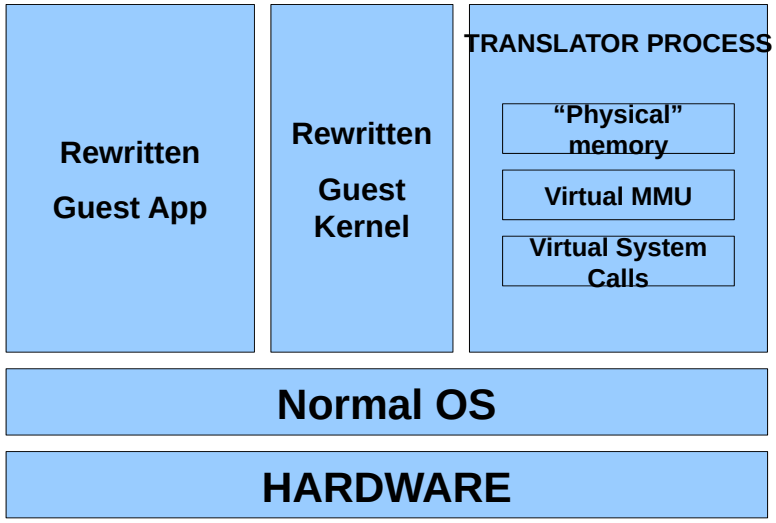


How to Build a VMM 2: Trap and Emulate

```
for(i = 0; i < 256; i++)
  mangle_pagetable_entry(&ptes[i]);
```

- 256 traps into the emulator
- Severe performance penalty

How to Build a VMM 3: Dynamic Binary Translation (VMware)



How to Build a VMM 3: Dynamic Binary Translation

```
for(i = 0; i < 256; i++)
  mangle_pagetable_entry(&ptes[i]);
```


How to Build a VMM 3: Dynamic Binary Translation

```
pte_t new_ptes[256];
for(i = 0; i < 256; i++)
  new_ptes[i] = mangled_entry(&ptes[i]);
register_new_ptes(new_ptes, 256);
```

- But when is this a safe alteration?

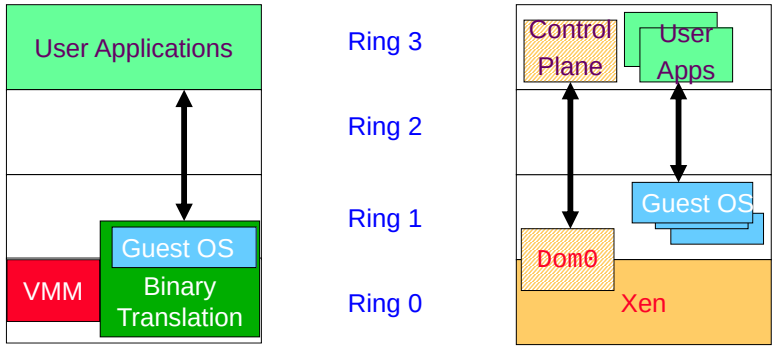
BREAK

How to Build a VMM 4: Paravirtualization (Xen)

Q. But when is this a safe alteration?

A. Let the humans worry about it

- Manually hack the OS: "paravirtualization"



Full Virtualization Paravirtualization

Xen: Founding Principles

- Key idea: Minimally alter guest OS to make VMs simpler and higher performance
  - Called paravirtualization (due to Denali project)
- Don't disguise multiplexing
- Execute faster than the competition
  - Note: VMWare does that too as "guest additions" are basically paravirtualization through specialized drivers (disk, I/O, video, …)

Xen: Emulate x86 (mostly)

- Xen paravirtualization:
  - Required less than 2% of the total lines of code to be modified
  - 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 Virtualization

- 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 and convert them to events posted to shared region with OS
  - Need both real and virtual time (and wall clock)

Virtualizing Virtual Memory

- Unlike MIPS, 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 (limit guest OS access) to avoid TLB flush
  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
  - Each guest OS has a Phy➔HW map
  - This enables the illusion of physically contiguous pages

Network

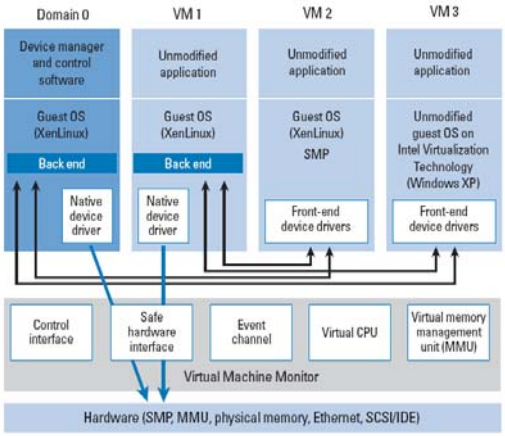
- Model:
  - Each guest OS has a virtual network interface connected to a virtual firewall/ router (VFR)
  - The VFR both limits the guest OS and also ensure correct incoming packet dispatch
- Exchange pages on packet receipt (to avoid copying)
  - No frame available ➔ dropped packet
- Rules enforce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

Disk

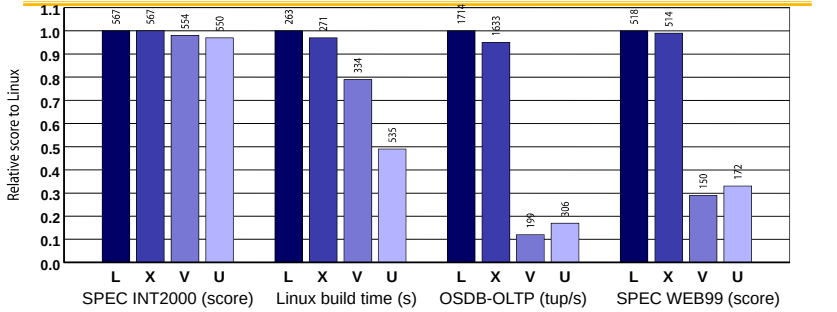
- Virtual block devices (VBDs): similar to SCSI disks
- Management of partitions, etc. done via domain 0
- Could also use NFS or network-attached storage instead

Domain 0 (dom0)

- Nice idea: run the VMM management at user level
  - Given special access to control interface for platform management
  - Has back-end device drivers
- Much easier to debug a user-level process than an OS
- Narrow hypercall API and checks can catch potential errors



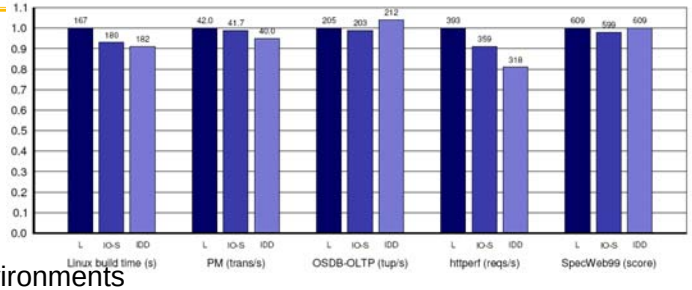
Benchmark Performance



Benchmark suite running on Linux (L), Xen (X), VMware Workstation (V), and UML (U)

- Benchmarks
  - Spec INT200: compute intensive workload
  - Linux build time: extensive file I/O, scheduling, memory management
  - OSBD-OLTP: transaction processing workload, extensive synchronous disk I/O
  - Spec WEB99: web-like workload (file and network traffic)
- Fair and reasonable comparisons?

I/O Performance



- Environments
  - L: Linux
  - IO-S: Xen using IO-Space access
  - IDD: Xen using isolated device driver
- Benchmarks
  - Linux build time: file I/O, scheduling, memory management
  - PM: file system benchmark
  - OSDB-OLTP: transaction processing workload, extensive synchronous disk I/O
  - httperf: static document retrieval
  - SpecWeb99: web-like workload (file and network traffic)

Xen Summary

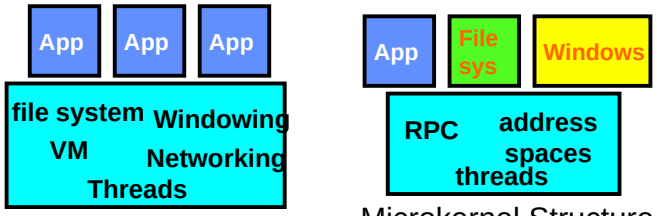
- Performance overhead of only 2-5%
- Available as open source but owned by Citrix since 2007
  - Modified version of Xen powers Amazon EC2
  - Widely used by web hosting companies
- Many security benefits
  - Multiplexes physical resources with performance isolation across OS instances
  - Hypervisor can isolate/contain OS security vulnerabilities
  - Hypervisor has smaller attack surface
    - » Simpler API than OS – narrow interfaces ➔ tractable security
    - » Less overall code than an OS
- BUT hypervisor vulnerabilities compromise everything…

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

Microkernel Operating Systems

- Example: split kernel into application-level servers.
  - File system looks remote, even though on same machine



Monolithic Structure Microkernel Structure

- Why split the OS into separate domains?
  - Simple modular OS components: process model enforces modularity, and allows incremental SW upgrades
  - Location transparent: service can be local or remote
    - » E.g., Each X Window client can be on a separate machine from X server and neither has to run on the machine with the frame buffer
  - Fault isolation?

Microkernels

- Requires good IPC performance
  - A lot of communication among the now separate parts of the OS
- Research and commercial examples:
  - Mach
  - Windows NT (due to Mach), but slowly moved pieces back into one monolithic OS (e.g. graphics)
- Issues:
  - Small TLBs also hurt microkernels
    - » More processes need to be resident at once
    - » But benchmarks done with few processes so this didn't affect architecture much!
  - Failure of user-level OS component can be damaging:
    - » E.g., microkernel virtual memory pager running as a user-level process introduces risk of *liability inversion* if pager hangs

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 isolated in practice?
- Xen: focus thus on *dividing* up resources, not managing them!
- Parallax is a file system that runs in another VM domain, more like a mounted file system
  - Avoids liability inversion problem

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?