Formal requirements for virtualizable third generation architectures

Lei Yan
POCS 20

Theorem 1

For any conventional 3rd generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

Subject

Hardware model:

- Mode (User/Supervisor)
- Virtual memory (base and bound)
- Traps
- Still the essence of modern architectures

States:

- S = (E, M, P, R)

Property

A VMM can be constructed that meets the following requirements:

- Equivalence
  - Running unmodified OS
  - OS has no idea if it runs on real or virtual machine
- Safety
  - Resource control
- Efficiency
  - Direct execution

Property

VMM construction: trap and emulate architecture

- What (VMM/guest OS/guest app) runs in which mode (user/supervisor)?
- What happens for privileged instruction?
  - run by guest OS?
  - attempt by guest app (e.g., syscall)?

Precondition

The set of sensitive instructions for that computer is a subset of the set of privileged instructions

- Privileged instructions
  - Instruction I is privileged if it traps in user mode but does not trap in supervisor mode
- Sensitive instructions
  - Control sensitive: instruction does not trap and changes the amount of resources or the processor mode (M and R)
  - Behavior sensitive: instruction behaves differently depending on M and R
- Innocuous

Precondition

What if control sensitive instructions do not trap?

- Breaks safety

What if behavior sensitive instructions do not trap?

- Breaks equivalence

What about innocuous?

- Breaks nothing

Modern architectures (e.g., x86) is non-virtualizable, how does existing techniques (hard-/software) workaround this?

- VT-x (http://www.cs.columbia.edu/~cdall/candidacy/pdf/Uhlig2005.pdf)
- Binary translation
- Paravirtualization

Is it always beneficial to pursue full-virtualization, i.e., equivalence?

- VMM cannot take advantage of high level information in VM:
  - Cannot deschedule a core of VM that waits for lock
- VMM provides abstraction over physical resources
  - Recall exokernel

Xen and the Art of Virtualization

POCS'20 Recitation
Mark Sutherland

Motivation for Xen

- Full virtualization of un-modified OS requires a virtualizable architecture
  - Commodity x86 architectures are **not** P&G virtualizable (why?)

- In many scenarios, exposing a subset of physical resources is desirable
  - E.g., disk block locations for faster disk scheduling, network Tx/Rx rings for direct I/O
  - Where have we seen this argument before?

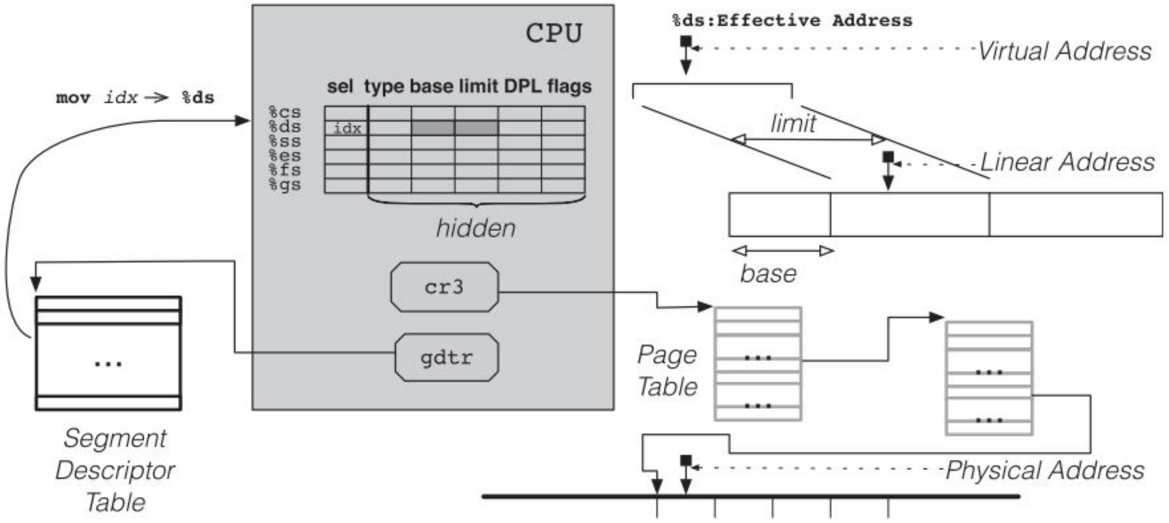
- Which of the P&G properties does Xen's choices directly change?

Full/Para-virtualized Machine Abstractions

	Full Virtualization	Paravirtualized
CPU	- Trap-and-emulate - Syscalls emulated before passed to the guest OS	- Guest OS runs in de-privileged mode - Interrupts/exceptions go through VMM - Syscalls can be short-cut into guest OS
Memory	- Guest has the illusion of the entire contiguous physical memory - VMM manages all relocations	- Guest allocates/manages its own pages - Page table updates go through VMM

X86 Address Translation Review

- Protected mode uses both segmentation and paging
- Paging from linear to physical address is invisible



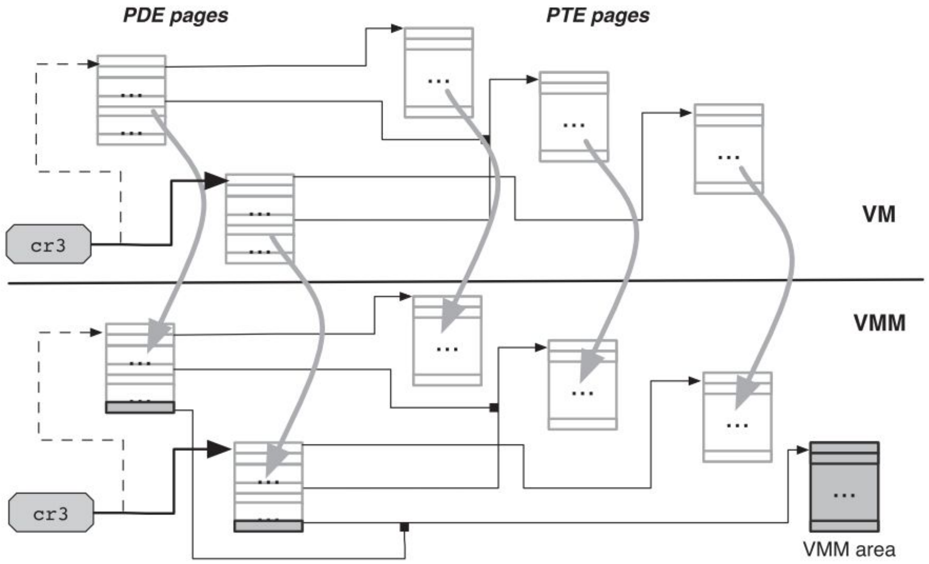
Virtualizing Paging

- Recall the set of privileged state: {cr3, PTEs themselves}

- What's a naive first idea?
  - Trap on every PTE access, redirect it to the correct place in real hardware memory
- What's the problem with this idea?
  - Performance overhead! An emulated PTE access can cost 2000 cycles [Bugnion, TOCS'12]

Virtualizing Paging - II

- What can we do about this?
  - Hint: where do instructions modify the PTEs so control can be vectored to the VMM?
- This approach is called “shadow page tables”

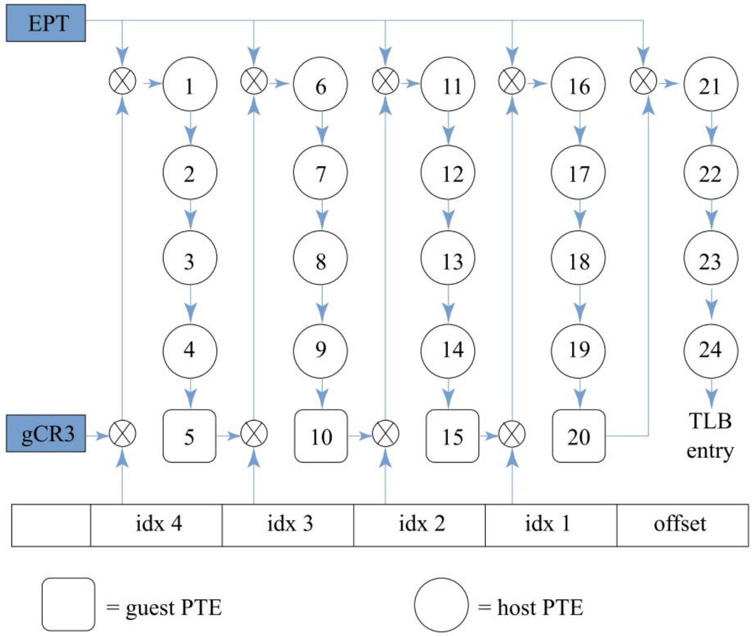


Paravirtualization Abstraction

- What changes to the VM abstraction could allow the guest OSes themselves to modify their page tables?
- How would this remove the need for “shadow” page tables?

Modern Developments: Extended Page Tables

- Almost all CPUs now have a feature called EPT
- EPT works by defining a "nested page walk" for each level of the guest PT



I/O Abstraction Comparison

	Full Virtualization	Paravirtualized
I/O	- Identical interfaces are re-exposed to all guests - All interactions emulated by VMM	- Network/disk abstractions are completely modified (e.g., VBD, VIF) - Replace emulation with hypercall

- Can you see a potential issue with even the Xen approach?
  - Hint: think back to week 2

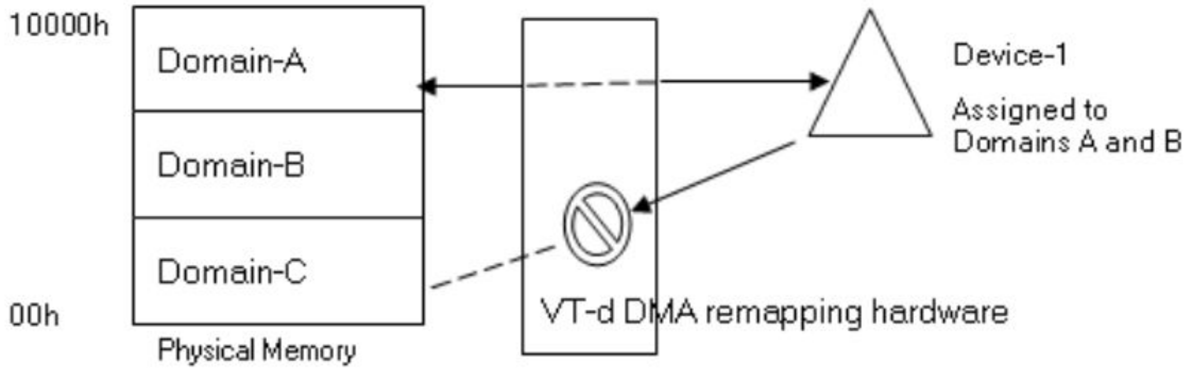
Modern Developments: IOV

- In Xen’s approach, how many layers are there in the I/O procedure?
  - Device → HW-visible I/O ring → Xen I/O ring → Guest → User
- Do you see a performance problem here?

- Today’s devices support native I/O Virtualization (IOV)
  - Multiple HW-visible rings, interrupt descriptors, etc…
  - Device → HW-visible I/O ring → Guest → User
  - Software also exists to remove the guest OS from that path

Modern Developments: VT-d

- Fundamental job of I/O: bring data blocks in and out of memory (DMA)
- How does this interact with paging?
  - Device → HW-visible I/O ring → Guest → User
- Do you see any problems here related to isolation?
  - Hint: think about who puts addresses onto the HW-visible rings


10000h
00h
Domain-A
Domain-B
Domain-C
Physical Memory
VT-d DMA remapping hardware
Device-1
Assigned to
Domains A and B