Formal requirements for virtualizable third generation architectures

Lei Yan
POCS 21

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

- Safety
  - Isolate guests using virtual memory
  - Isolate VMM from guests using privilege modes
- Equivalence
  - Nothing we can do for unprivileged instructions. What about privileged instructions?
    - They trap so VMM can emulate them.
    - Example: how to emulate a syscall from a guest app?
  - Isolate guest OS from guest app?
- Efficiency

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 changes the amount of resources (R) or the processor mode (M) (and does not cause memory trap)
  - 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'21 Recitation

Lei Yan

Slides adopted from Mark Sutherland

Xen

- Full virtualization has drawbacks
  - Full virtualization of un-modified OS requires a virtualizable architecture
    - Commodity x86 architectures are **not** P&G virtualizable
  - In many scenarios, exposing a subset of physical resources is desirable
    - Optimizing page placement for cache locality
    - Use real time to handle TCP timeouts correctly

- Solution: Paravirtualization

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

Full-virtualization Paging

- “shadow page tables”
  - Trap on page table updates and reflect the updates to the hardware page table

Full-virtualization of Paging - II

- Maintaining Shadow page table is costly
  - Syncing the virtual page table seen by the guest and the “shadow” table seen by hardware
    - E.g., propagation of the dirty bit.

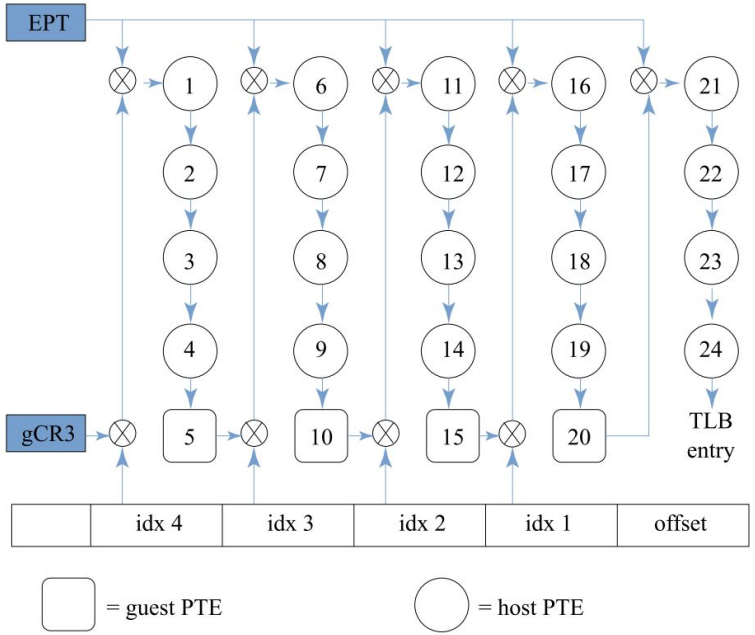
Paravirtualization Abstraction

- 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



Modern Developments of I/O virt: 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 of I/O virt: 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