Virtual Machine (VM)

“A VM is an efficient, isolated duplicate of a real machine” [Popek&Goldberg 74]

- **Duplicate**: VM should behave identically to the real machine
  - Programs cannot distinguish between real or virtual hardware
  - Except for
    - Fewer resources (potentially different between executions)
    - Some timing differences (when dealing with devices)
- **Isolated**: Several VMs execute without interfering with each other
- **Efficient**: VM should execute at speed close to that of real hardware
  - Requires that most instruction are executed directly by real hardware

Hypervisor aka virtual machine monitor (VMM): Software layer implementing the VM

Types of Virtualisation

- **Platform VM or System VM**
- **Operating System**
- **Process**
- **Virtualisation Layer**

Why Virtual Machines?

- Historically used for easier sharing of expensive mainframes
- Run several (even different) OSES on same machine
  - called guest operating system
- Each on a subset of physical resources
- Can run single-user single-tasked OS in time-sharing mode
- Legacy support

Obsolete by 1980s

Why Virtual Machines?

- Heterogeneous concurrent guest OSes
  - e.g. Linux + Windows
- Improved isolation for consolidated servers: QoS & Security
  - Total mediation/encapsulation
  - Replication
  - Migration/consolidation
  - Checkpointing
  - Debugging
- Uniform view of hardware

Would not be needed if OSES provided proper security & resource management!
Why Virtual Machines: Cloud Computing

- Increased utilisation by sharing hardware
- Reduced maintenance cost through scale
- On-demand provisioning
- Dynamic load balancing though migration

Hypervisor aka Virtual Machine Monitor

- Software layer that implements virtual machine
- Controls resources
- Partitions hardware
- Schedules guests
  - "world switch"
- Mediates access to shared resources
  - e.g. console, network

Implications:
- Hypervisor executes in privileged mode
- Guest software executes in unprivileged mode

Native vs Hosted Hypervisor

- Hosted VMM besides native apps
  - Sandbox untrusted apps
  - Convenient for running alternative OS on desktop
  - Leverage host drivers

Overheads:
- Double mode switches
- Double context switches
- Host not optimised for exception forwarding

Virtualisation Mechanics: Instruction Emulation

- Traditional trap-and-emulate (T&E) approach:
  - guest attempts to access physical resource
  - hardware raises exception (trap), invoking HV’s exception handler
  - hypervisor emulates result, based on access to virtual resource

Guest

VMM

Most instructions do not trap
- prerequisite for efficient virtualisation
- requires VM ISA (almost) same as processor ISA

Trap & Emulate Requirements

- Privileged instruction: when executed in user mode will trap
- Privileged state: determines resource allocation
  - Incl. privilege mode, PT ptr, exception vectors...
- Sensitive instruction:
  - control sensitive: change privileged state
  - behaviour sensitive: expose privileged state
  - eg privileged instructions which NO-OP in user state
- Innocuous instruction: not sensitive

"Impure" Virtualisation

- Support non-T&E hardware
- Improve performance

- Modify binary: binary translation (VMware)
- Modify hypervisor "ISA" para-virtualisation

No-op is insufficient!

T&E virtualisable HW:
All sensitive instructions are privileged
Virtualisation vs Address Translation

Two levels of address translation!
Must implement with single MMU translation!

Virtualisation Mechanics: Shadow Page Table

Hypervisor must shadow (virtualise) PT updates by guest:
• trap guest writes to guest PT
• translate guest PA in guest (virtual) PTE using memory map
• insert translated PTE in shadow PT

Shadow PT has TLB semantics (i.e. weak consistency) => Update at synchronisation points:
• page faults
• TLB flushes

Mechanics: Lazy Shadow Update

On guest PT access must translate (virtualise) PTEs:
• store: guest "PTE" → real PTE
• load: real PTE → guest "PTE"
Mechanics: Optimised Guest Page Table

- Guest translates PTE on read from PT
- Linux PT-access wrappers help
- Guest batches PR updates
- hypcalls to reduce overhead

Example: Interrupt-enable in virtual PSR
- guest and VMM agree on VPSR location
- VMM queues guest IRQs when disabled in VPSR

Mechanics: Guest Self-Virtualisation

Minimise traps by holding some virtual state inside guest

Example:
```
mov r1,#VPSR
ldr r0,[r1]
orr r0,r0,#VPSR_ID
sto r0,[r1]
```

Mechanics: Device Models

- Emulated
- Split (para-virtualised)
- Pass-through

Mechanics: Emulated Device

Each device access must be trapped and emulated
- unmodified native driver
- high overhead!
- may not actually work, violate device timing constraints

Mechanics: Split Driver

Simplified, high-level device interface
- small number of hypcalls
- new (but very simple) driver
- low overhead
- must port drivers to hypervisor

Mechanics: Driver OS (Xen Dom0)

Leverage native drivers
- no driver porting
- must trust complete driver guest!
- huge trusted computing base (TCB)!
Mechanics: Pass-Through Driver

- Unmodified native driver
  - Must trust driver (and guest) for DMA
  - Can't share device between VMs
  - Except with hardware support: recent NICs

Direct device access by guest!

x86 Virtualisation Extensions: VT-x

- New processor mode: VT-x root mode
  - orthogonal to protection rings
  - Entered on virtualisation trap

Root

Ring 0

Ring 3

Ring 2

Ring 1

Ring 0

Guest Kernel

VMM

Kernel entry

VM exit

Hypervisor

Arm Virtualisation Extensions (1)

- EL2 aka "hyp mode"
  - New privilege level
    - Strictly higher than kernel (EL1)
    - Virtualizes or traps all sensitive instructions
    - Presently only available in Arm TrustZone "normal world"

Arm Virtualisation Extensions (2)

- Configurable Traps
  - Can configure traps to go directly to guest OS

Arm Virtualisation Extensions (3)

- Emulation
  - No x86 equivalent
  - HW decodes instruction
    - No L1 miss
    - No software decode
  - SW emulates instruction
    - Usually straightforward

Emulation

1) Load faulting instruction:
   - Compulsory L1-D miss!
2) Decode instruction
   - Complex logic
3) Emulate instruction
   - Usually straightforward

Emulation
### Arm Virtualisation Extensions (4)

**2-stage translation**

- x86 similar (EPTs)
- Hardware PT walker traverses both PTs
- PT walker loads combined (guest-virtual to physical) mapping into TLB
- eliminates "virtual TLB"

- On page fault walk twice number of page tables!
- Can have a page miss on each, requiring PT walk
- \( O(n^2) \) misses in worst case for n-level PT
- Worst-case cost is massively worse than for single-level translation!

**Trade-off:**
- fewer traps
- simpler implementation
- higher TLB-miss cost up to 50% of run-time!

### Arm Virtualisation Extensions (5)

**Virtual Interrupts**

- 2-part IRQ controller
  - global "distributor"
  - per-CPU interface
- New HW "virt. CPU interface"
  - Mapped to guest
  - Used by HV to forward IRQ
  - Used by guest to acknowledge
  - Halves hypervisor invocations for interrupt virtualization

- x86: issue only for legacy level-triggered IRQs

### Arm Virtualisation Extensions (6)

**System MMU (I/O MMU)**

- Devices use virtual addresses
- Translated by system MMU
  - elsewhere called I/O MMU
  - translation cache, like TLB
  - reloaded from I/O page table

- Can do pass-through I/O safely
  - guest accesses device registers
  - no hypervisor invocation

### Hybrid Hypervisor-OSes

**Non-Root**

- Guest apps
- Guest apps
- Linux kernels
- Linux kernels
- Linux apps
- Native Linux apps

**Root**

- Linux kernels
- Linux kernels
- Linux drivers

**Idea:** Turn OS into hypervisor by running in VT-x root mode, pioneered by KVM

**Huge TCB, contains full Linux system (kernel and userland)!**

**Often falsely called a "Type-2" hypervisor**

**Ideas:**
- Reuse Linux drivers!
Fun and Games with Hypervisors

- Time-travelling virtual machines [King '05]
  - debug backwards by replay VM from checkpoint, log state changes
- SecVisor: kernel integrity by virtualisation [Seshadri '07]
  - controls modifications to kernel (guest) memory
- Overshadow: protect apps from OS [Chen '08]
  - make user memory opaque to OS by transparently encrypting
- Turtles: Recursive virtualisation [Ben-Yehuda '10]
  - virtualize VT-x to run hypervisor in VM
- CloudVisor: mini-hypervisor underneath Xen [Zhang '11]
  - isolates co-hosted VMs belonging to different users
  - leverages remote attestation (TPM) and Turtles ideas

... and many more..