Virtualization II

Adam Belay <abelay@mit.edu>
Plan for today

• Last lecture: Virtualization basics
  • A VMM is an operating system that maintains a machine-like interface instead of a process interface
  • Many compelling reasons to use virtualization
  • Originally, virtualization wasn’t believed to be possible on x86
  • VMware introduced binary translation solution

• Today: Recent developments
  • More detailed discussion of HW support for virtualization
  • Safe user-level access to privileged CPU features
Intel VT-x

• Makes x86 hardware “virtualizable” under Popek and Goldberg definition

• Goal: Direct execution of most privileged instructions

• Introduces two CPU modes, kind of like ring protection
  • VMX Root Mode: For running VMM (host)
  • VMX Non-root Mode: For running VMs (guest)
  • But each mode has its own rings (CPL0 – CPL3)

• In-memory structure called VMCS stores privileged register state and control flags
Intel VT-x

VMX Non-Root Mode

%CR0-8 %EFLAG CPL 3
GDT IDT CPL 0

VMX Root Mode

%CR0-8 %EFLAG CPL 3
GDT IDT CPL 0

CONTROL
GUEST STATE
HOST STATE
VMCS
Intel VT-x: VM Enter

VMX Non-Root Mode
- CPL 3
- CPL 0
- %CR0-8
- %EFLAG
- GDT
- IDT

VMX Root Mode
- CPL 3
- CPL 0
- %CR0-8
- %EFLAG
- GDT
- IDT

VM Enter

VMCS
- CONTROL
- GUEST STATE
- HOST STATE
Intel VT-x: VM Exit

VMX Non-Root Mode

VMX Root Mode

VM Exit

VMCS
VM Enter and VM Exit

• Transitions between VMX Root Mode and VMX Non-root Mode

• VM Exit
  • VMCALL instruction, EPT Page Faults, some trap and emulate (configured in VMCS)

• VM Enter
  • VMLAUNCH instruction: Enter VMX Non-root Mode for a new VMCS
  • VMRESUME instruction: Enter VMX Non-root Mode for the last VMCS (faster)

• Typical VM Exit/Enter is ~200 cycles on modern HW
Intel EPT (nested paging)

- Goal: **Direct execution** of guest page table interactions
  - Reads and write to page table in memory
  - `mov %eax, %cr3, INVLPD, etc.`

- Idea: Maintain two layers of paging translation
  - Normal page table: Guest-virtual to guest-physical
  - EPT: guest-physical to host-physical

---

**Diagram:**

- Guest Virtual Address
- Guest Physical Address
- Host Physical Address
- PT
- EPT
Intel EPT

TLB

GVA | HPA
GVA | HPA
GVA | HPA

Page Walker

MMU

PGTBL

PGDIR

EPGTBL

PGDIR

Guest VA -> Guest PA

Guest Kernel

VMM

Guest PA -> Host PA
Q: What’s the worst case page walk time with EPT enabled?
Q: What’s the worst case page walk time with EPT enabled?

• \(O(N^2)\): Each page table level could require an EPT page walk

• But in practice CPU hardware caches the first couple levels of page table and EPT, so usually \(O(N)\)
Q: What’s faster EPT or Shadow Page Tables?
SR-IOV + IOMMU

• Goal: Allows direction execution of I/O device access

• Challenge #1: How to partition a single device into multiple instances
  • SR-IOV: Allows a PCIe device to expose multiple, separate memory-mapped I/O regions

• Challenge #2: How to prevent DMA from overwriting memory belonging to VMM or another guest
  • IOMMU: Provides paging translation across PCIe bus
IOMMU

CPU 0
CPU 1
CPU 2
CPU 3

MMU

RAM

PCIe Device

bus
Big picture

• Direct execution reduces overhead
  • Avoids VM exits, trap-and-emulate, binary translation

• Enabled by three microarchitectural changes:
  • Intel VT-x: direct execution of most privileged instructions (e.g. IDT, GDT, ring protection, EFLAG, etc.)
  • Intel EPT: direct execution of page table manipulation
  • IOMMU + SRIOV: direct execution of I/O interactions (e.g. disk, network, etc.)
Operating systems today
What if you could give a process access to raw hardware?

1. Access to full hardware capabilities
2. Access to all existing Linux abstractions
Could build new OS on top of Linux

1. Access to full hardware capabilities
2. Access to all existing Linux abstractions

Key idea: Using Linux means access through system calls
But still have to maintain process isolation

1. Access to full hardware capabilities
2. Access to all existing Linux abstractions
Dune

• Key Idea: Use VT-x, EPT, etc. to support Linux processes instead of virtual machines
• Dune is loadable kernel module, makes it possible for an ordinary Linux process to switch to “Dune mode”
• Dune mode processes can run along side ordinary processes. Within a process, some threads can be in Dune mode even if others aren’t.
A dune process

• Is still a process
  • has memory, can make Linux system calls, is fully isolated, etc.

• But isolated with VT-x Non-root mode
  • Rather than with CPL=3 and page table protections

• memory protection via EPT
  • Dune configures EPT so process can only access the same physical pages it would normally have access to
Why isolate a process with VT-x?

• Process can access all of Linux environment while also directly executing most privileged instructions
• User code now runs at CPL 0
• Process can manage its own page table via %CR3
• Fast exceptions (e.g. page faults) via shadow IDT
  • Kernel crossings eliminated
• Can run sandboxed code at CPL 3
  • So process can act like a kernel!
How to perform a Linux system call in a Dune process?

• INT $80 just traps inside process at handler specified in shadow IDT
How to perform a Linux system call in a Dune process?

• INT $80 just traps inside process at handler specified in shadow IDT

• VMCALL instruction forces a VM Exit
  • Dune module vectors exit into kernel system call table

• Challenge: Compatibility
  • Existing code and libraries don’t use VMCALL

• Solution: Shadow IDT handler forwards the system calls it catches using VMCALL
How to perform a Linux system call in a Dune process?

![Diagram showing the process of performing a system call in a Dune process.](image)
Microbenchmarks: Overheads

- Two sources of overhead
  - VM exits and VM enters
  - EPT translations

<table>
<thead>
<tr>
<th>(cycles)</th>
<th>Getpid</th>
<th>Page fault</th>
<th>Page walk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>138</td>
<td>2,687</td>
<td>36</td>
</tr>
<tr>
<td>Dune</td>
<td>895</td>
<td>5,093</td>
<td>86</td>
</tr>
</tbody>
</table>
Microbenchmarks: Speedups

• Large opportunities for optimization
  – Faster system call interposition and traps
  – More efficient user-level virtual memory manipulation

<table>
<thead>
<tr>
<th></th>
<th>ptrace (getpid)</th>
<th>trap (TRAP, PROT1, UNPROT)</th>
<th>Appel 1 (PROTN, TRAP, UNPROT)</th>
<th>Appel 2 (PROTN, TRAP, UNPROT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>27,317</td>
<td>2,821</td>
<td>701,413</td>
<td>684,909</td>
</tr>
<tr>
<td>Dune</td>
<td>1,091</td>
<td>587</td>
<td>94,496</td>
<td>94,854</td>
</tr>
</tbody>
</table>
Example: Sandboxed execution

• Suppose your browser wants to run a plugin
  • It could be buggy or malicious

• Need a way to execute plugin but limit system calls and memory access

• Using Dune:
  • Could create a page table with PTE_U mappings for allowed access and ~PTE_U for prohibited access
  • Run browser in CPL0 and plugin in CPL3
  • Plugin can run system calls but they trap into browser
  • Browser filters or emulates system calls
Sandboxing diagram

sh

Web Browser (Dune CPL 0)

Plugin (CPL 3)

Plugin (CPL 3)

Linux Kernel

Hardware
Sandbox: SPEC2000 performance

- Only notable end-to-end effect is EPT overhead
- Can be eliminated through use of large pages
Example: Garbage collection (GC)

• GC is mostly about tracing pointers to find live data
  • set a mark flag in every reached object
  • Any object not marked is dead and can be freed

• Boehm collector is concurrent GC:
  • Mutator runs in parallel with tracer -- with no locks
  • At some point the tracer has followed all pointers
    • But mutator might modify pointers in already traced objects
    • Solution: pause mutator briefly, look at all pages modified since tracer has started

• How does Dune help?
  • Clear all PTE dirty bits (PTE_D) at start of GC
  • Scan for set PTE dirty bits to detect written pages
Example: Garbage collection (GC)
More thoughts on use cases

• Dune provides similar benefits to Exokernel
  • Raw access to paging hardware for Appel + Li paper
  • Speed improvements alone may make some ideas more feasible (GC, DSM, etc.)

• Each Dune thread can have a different page table!
  • E.g. sthreads, a mechanism for least privilege
Conclusion

- VT-x, EPT, and SR-IOV/IOMMU enable direct execution of guest instructions
- Dune implements processes with VT-x and EPT rather than ordinary ring protection
- Dune processes can use both Linux system calls and privileged HW
  - Enables fast access to page table and page faults
  - Enables processes to build kernel-like functionality
    - E.g. sandboxing untrusted plugins in CPL3
    - Hard to do this at all in Linux let alone efficiently