KVM,
Dune
Yiying Zhang
Outline

- Recap on hardware-assisted virtualization
- KVM
- Dune
Lec3: Hardware-Assisted CPU Virtualization (Intel VT-x)

• Two new modes of execution (orthogonal to protection rings)
  
  • VMX root mode: same as x86 without VT-x
  
  • VMX non-root mode: runs VM, sensitive instructions cause transition to root mode, even in Ring 0

• New hardware structure: VMCS (virtual machine control structure)
  
  • One VMCS for one virtual processor
  
  • Configured by VMM to determine which sensitive instructions cause VM exit
  
  • Specifies guest OS state
Lec3: Comparison of Pre VT-x and Post VT-x

Hardware w/o VT-x

<table>
<thead>
<tr>
<th>Guest Applications</th>
<th>VMX non-root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS</td>
<td>Ring 1</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>VMX root Ring 0</td>
</tr>
</tbody>
</table>

Hardware w/ VT-x

<table>
<thead>
<tr>
<th>Guest Applications</th>
<th>VMX non-root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS</td>
<td>Ring 0</td>
</tr>
<tr>
<td>Host Applications</td>
<td>VMX root Ring 0</td>
</tr>
<tr>
<td>Hypervisor</td>
<td></td>
</tr>
</tbody>
</table>
Lec3: VMX Mode Transition with Intel VT-x

- VM exit/entry (to/from root mode)
  - Registers and address space swapped in one atomic operation
  - Guest- and host-states saved and loaded to VMCS during transitions
- Whenever possible, sensitive instructions only affect states within the VMCS instead of always trapping (VM exit)
- VM exit
  - `vmcall` instruction
  - EPT page faults
  - Interrupts
  - Some sensitive instructions (configured in VMCS)
- VM enter
  - `vmlaunch` instruction: enter with a new VMCS
  - `vmresume` instruction: enter for the last VMCS
- Typical vm exit/enter taks ~200 cycles on modern CPU
Lec4: Hardware-Assisted Memory Virtualization

• Hardware support for memory virtualization

  • Intel EPT (Extended Page Table) and AMD NPT (Nested Page Table)

• EPT: a per VM table translating PPN -> MPN, referenced by EPT base pointer

• EPT controlled by the hypervisor, guest page table (GPT) controlled by guest OS (both exposed to hardware)

• Hardware directly walks GPT + EPT (for each PPN access during GPT walk, needs to walk the EPT to determine MPN)

• No VM exits due to page faults, INVLP�, or CR3 accesses
Lec 4: EPT Translation: Details
Lec4: EPT Increases Memory Access

One memory access from the guest VM may lead up to **20 memory accesses**!

*sPA*: machine address  
*nCR3*: root of EPT table  
*nL_k*: EPT table level
KVM: Linux-based Virtualization

Avi Kivity
avi@qumranet.com

Columbia University Advanced OS/Virtualization course
At a glance

- KVM – the Kernel-based Virtual Machine – is a Linux kernel module that turns Linux into a hypervisor
- Requires hardware virtualization extensions
- Supports multiple architectures: x86 (32- and 64-bit), s390 (mainframes), PowerPC, ia64 (Itanium)
- Competitive performance and feature set
- Advanced memory management
- Tightly integrated into Linux
The KVM approach

- Reuse Linux code as much as possible
- Focus on virtualization, leave other things to respective developers
- Integrate well into existing infrastructure, codebase, and mindset
- Benefit from semi-related advances in Linux
Xen

- Domain 0
  - Driver
- User VM
  - Driver
- User VM
- User VM

Hypervisor

Hardware
KVM

Ordinary Linux Process

User VM

User VM

User VM

Modules

Linux

Driver

Driver

Driver

Hardware
KVM model benefits

- Reuse scheduler, memory management, bringup
- Reuse Linux driver portfolio
- Reuse I/O stack
- Reuse management stack
KVM Execution Model

- Three modes for thread execution instead of the traditional two:
  - User mode
  - Kernel mode
  - Guest mode
- A virtual CPU is implemented using a Linux thread
- The Linux scheduler is responsible for scheduling a virtual cpu, as it is a normal thread
KVM Execution Model

- **Userspace**
  - Userspace exit handler
  - ioctl()

- **Kernel**
  - Switch to Guest Mode
  - Kernel exit handler

- **Guest**
  - Native Guest Execution
KVM Execution Model

- Guest code executes natively
  - Apart from trap'n'emulate instructions
- Performance critical or security critical operations handled in kernel
  - Mode transitions
  - Shadow MMU
- I/O emulation and management handled in userspace
  - Qemu-derived code base
  - Other users welcome
KVM Memory Model

- Kernel Address Space
- User Address Space
- VMM userspace code and data
- Guest physical address space
KVM Memory Model

- Guest physical memory is just a chunk of host virtual memory, so it can be:
  - Swapped
  - Shared
  - Backed by large pages
  - Backed by a disk file
  - COW'ed

- The rest of the host virtual memory is free for use by the VMM:
  - Low bandwidth device emulation
  - Management code
Linux Integration

- Preemption (and voluntary sleep) hooks: preempt notifiers
- Swapping and other virtual memory management: mmu notifiers
Preempt Notifiers

- Linux may choose to suspend a vcpu's execution
- KVM runs with some guest state loaded while in kernel mode (FPU, etc.)
- Need to restore state when switching back to user mode
- Solution: Linux notifies KVM whenever it preempts a process that has guest state loaded
  - ... and when the process is scheduled back in
- Allows the best of both worlds
  - Low vmexit latency
  - Preemptibility, sleeping when paging in
Preempt notifiers

External interrupt or trap

Guest

VMM process in host kernel

Scheduler

Other process

Restore host state

Context switch

Restore guest state

Context switch

Copyright © 2007 Qumranet, Inc. All rights reserved.
MMU Notifiers

- Linux doesn't know about the KVM MMU
- So it can't
  - Flush shadow page table entries when it swaps out a page (or migrates it, or ...)
  - Query the pte accessed bit when determines the recency of a page
- Solution: add a notifier
  - for tlb flushes
  - for accessed/dirty bit checks
- With MMU notifiers, the KVM shadow MMU follows changes to the Linux view of the process memory map
Paravirtualization

- Yesterday's hot topic
  - Needed for decent MMU performance without two-dimensional paging
  - Intrusive
- KVM has modular paravirtualization support
  - Turn on and off as needed by hardware
  - Still needs hardware virtualization extensions
- Supported areas
  - Hypercall-based, batched mmu operations
  - Clock
Virtio

- Most devices emulated in userspace
  - With fairly low performance
- Paravirtualized I/O is the traditional way to accelerate I/O
- Virtio is a framework and set of drivers:
  - A hypervisor-independent, domain-independent, bus-independent protocol for transferring buffers
  - A binding layer for attaching virtio to a bus (e.g. pci)
  - Domain specific guest drivers (networking, storage, etc.)
  - Hypervisor specific host support
Architecture: Host/guest device emulation split

Guest device – device model visible to guest

- rtl8139
- Intel e1000
- virtio-net

Decouples hardware emulation from I/O mechanism

- tap
- L2TPv3
- socket

Host device – performs I/O on behalf of guest
KVM Conclusion

- Tight integration with Linux
- The KVM module is relatively simple, with most of the functionalities already implemented in the Linux kernel
- KVM relies on hardware virtualization support (which is prevalent now)
- KVM performance is generally good
- Increasing popularity with AWS making the big move from Xen to KVM
Dune: Safe User-level Access to Privileged CPU Features

Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis
Stanford University
The power of privilege

• Privileged CPU features are fundamental to kernels

• But other, compelling uses:
  – Speed up garbage collection (Azul C4)
    • Page tables provide memory access information
  – Privilege separation within a process (Palladium)
    • MMU hardware isolates compartments
  – Safe native code in web browsers (Xax)
    • System call handler intercepts system calls
Should we change the kernel?

- Problem: stability concerns, challenging to distribute, composability concerns
What about an Exokernel?

- Problem: must replace entire OS stack
What about a virtual machine?

- Problem: virtual machines have strict partitioning
Dune in a Nutshell

- Provide safe user-level access to privileged CPU features
- Still a normal process in all ways (POSIX API, etc)
- Key idea: leverage existing virtualization hardware (VT-x)
How can Dune be Useful?

• Processes can have fast exception handling (e.g., page faults)
  • They can register their own interrupt descriptor table
• Processes can manage their own page tables
  • Useful for many tasks like GC to access and manipulate PTEs (e.g., dirty bits)
• Can have different trust zones (privilege modes) in a process
  • e.g., for sandboxing
Garbage collection in Dune

- Solution: control the page table directly within a process
Available CPU features

• Privilege Modes
  – SYSRET, SYSEXIT, IRET

• Virtual Memory
  – MOV CRn, INVLPg, INVPCID

• Exceptions
  – LIDT, LTR, IRET, STI, CLI

• Segmentation
  – LGDT, LLDT
Dune architecture

- Host mode -> VMX root mode on Intel
- Normally used for hypervisors
- In Dune, we run the kernel here
  - Reason: need access to VT-x instructions
Dune architecture

• Guest mode -> VMX non-root mode on Intel
• Normally used by the guest OS
• In Dune, we run ordinary processes here
  – Reason: need access to privileged features
• Dune Module (~2500 LOC)
  – Configures and manages virtualization hardware
  – Provides integration with the rest of the kernel in order to support a process abstraction
  – Uses Intel VT-x (could easily add AMD SVM)
**Dune architecture**

- **libDune (∼6,000 LOC)**
  - A utility library to help applications manage privileged hardware features
  - Completely untrusted
  - Exception handling, system call handling, page allocator, page table management, ELF loader
Providing a process abstraction

• Memory management
• System calls
• POSIX Signals
Memory management in Dune

- Configure the EPT to provide process memory
- User programs can then directly access the page table
System calls in Dune

• SYSCALL will only trap back into the process
• Use VMCALL (i.e. a hypercall) to perform normal kernel system calls
But SYSCALL is still useful

- Isolate untrusted code by running it in a less privileged mode (i.e. ring 3 on x86)
- Leverage the ‘supervisor’ bit in the page table to protect memory
Signals in Dune

• Signals should only be delivered to ring 0
• What happens if process is in ring 3?
• Possible solution: have the Dune module manually transition the process to ring 0
  – Works but slow and somewhat complex
• Our solution: deliver signals as injected interrupts
  – Hardware automatically switches to ring 0
  – Can use CLI and STI to efficiently mask signals
Many implementation challenges

• Reducing VM exit and VM entry overhead
• Pthread and fork were tricky to integrate with the Linux kernel
• EPT does not support enough address space
• Check the paper for details
Evaluation

• How much overhead does Dune add?
• What potential does Dune create for optimization?
• What is Dune’s performance in end-to-end use cases?
Overhead analysis

- Two sources of overhead
  - VMX transitions
  - EPT translations

<table>
<thead>
<tr>
<th></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>
Optimization analysis

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

<table>
<thead>
<tr>
<th>(cycles)</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>
End-to-end case studies

• We built and evaluated three systems
  • Application sandbox (~1300 LOC)
    – Constrained the system calls performed by an untrusted binary
  • Garbage collection (less than 100 LOC change)
    – Improved dirty page detection through direct access to dirty bits
  • Privilege separation (~750 LOC)
    – Supported several protection domains within a single process through use of multiple page roots (with TLB tagging)
Conclusions

• Applications can benefit from access to privileged CPU features
• Virtualization hardware allows us to provide such access safely
• Dune creates new opportunities to build and improve applications without kernel changes
• Dune has modest performance overhead
• Download Dune at http://dune.scs.stanford.edu
Discussion

- How is Dune different from VMs?
- How is Dune different from Unikernels (as VM) and Unikernels as process?
- Can we extend Dune to utilize hardware I/O virtualization features (SR-IOV)?
- Dune: just a nice research idea or something that could have real use?