[537] Virtual Machines

Tyler Harter
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Virtual Machines

**Goal**: run an OS over an OS

Who has done this?

Why might it be useful?
Virtual Machines

**Goal**: run an OS (guest) over an OS (host)

Who has done this?

Why might it be useful?
Motivation

**Functionality**: want Linux programs on Mac OS X

**Consolidation**: avoid light utilization

**Cloud computing**: fast scalability

**Testing/Development**: for example, xv6
Virtualization Software

**Desktop**: VMware, VirtualBox, qemu

**Cloud**: Amazon ec2, Microsoft Azure, Google Compute
Needs

An OS expects to run on **raw hardware**.

Need to give **illusion** to OS of **private** ownership of H/W.

Didn’t we already virtualize H/W? How is this different?
Process Virtualization

We have done two things:
- given **illusion** of private resources
- provided more **friendly interface**

**The interface** (what **processes** see/use):
- virtual memory (w/ holes)
- most instructions (but not lidt, etc)
- most registers (but not cr3, etc)
- syscalls, files, etc
Process Virtualization

We have done two things:
- given **illusion** of private resources
- provided more **friendly interface**

The interface (what **processes** see/use):
- virtual memory (w/ holes)
- most instructions (but not lidt, etc)
- most registers (but not cr3, etc)
- syscalls, files, etc
Machine Virtualization

We have done two things:
- given illusion of private resources
- provided more friendly interface (get rid of this)

The interface (what guest OS’s see/use):
- “physical” memory (no holes), PT management
- all instructions (even dangerous ones!)
- all registers
- “physical” devices, interrupts, disks, etc
Before

P1  P2  P3

Linux

Hardware
Now

- Linux
- OS X
- Windows

Hypervisor

Hardware
Approach 1

Write a **simulator**.

For example:
- big array for “physical” memory
- run over OS instructions, call function for each
Approach 1

Write a **simulator**.

For example:
- big array for “physical” memory
- run over OS instructions, call function for each

Problems?
Approach 1

Write a **simulator**.

For example:
- big array for “physical” memory
- run over OS instructions, call function for each

Problems? (performance)
Solution?
Approach 1

Write a **simulator**.

For example:
- big array for “physical” memory
- run over OS instructions, call function for each

Problems? (performance)
Solution? Limited Direct Execution!
Approach 2: Limited Direct Execution

Hypervisor runs in *kernel mode* and can do anything.

Processes *and* guest OS’s run in *user mode* when they don’t need to do anything privileged.

LDE is like baby proofing!
Process/Guest Privilege

**Process**: how do processes correctly do privileged ops?
Process/Guest Privilege

**Process**: how do processes correctly do privileged ops?

**Guest**: why can’t guest OS’s do the same?
Process/Guest Privilege

**Process**: how do processes correctly do privileged ops?

**Guest**: why can’t guest OS’s do the same?

**Process**: What should an OS do when a process tries to call something like `lidt`?
Process/Guest Privilege

**Process**: how do processes correctly do privileged ops?

**Guest**: why can’t guest OS’s do the same?

**Process**: What should an OS do when a process tries to call something like `lidt`? Kill it.
Process/Guest Privilege

**Process**: how do processes correctly do privileged ops?

**Guest**: why can’t guest OS’s do the same?

**Process**: What should an OS do when a process tries to call something like `lidt`? Kill it.

**Guest**: What should a hypervisor do when a guest OS tries to call something like `lidt`?
**Process/Guest Privilege**

**Process**: how do processes correctly do privileged ops?

**Guest**: why can’t guest OS’s do the same?

**Process**: What should an OS do when a process tries to call something like `lidt`? Kill it.

**Guest**: What should a hypervisor do when a guest OS tries to call something like `lidt`? Emulate it.
Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Classical Virtualization

User Mode: Process

Kernel Mode: OS
Classical Virtualization

**challenge**: operating systems don’t trust each other!
Classical Virtualization

**strategy**: run OS in user mode

- **User Mode**: Process, OS, Process, OS, Process
- **Kernel Mode**: VMM
Classical Virtualization

**challenge**: OS thinks it’s in kernel mode

**User Mode**: Process → OS → Process → OS → Process

**Kernel Mode**: VMM
Classical Virtualization

**strategy**: emulate privileged ops
Example

How to emulate an `lidt` call.
Example

How to emulate an `lidt` call.

Review IDT table...
movl $6, %eax;  int $64
struct gatedesc idt[256] (trap.c)

movl $6, %eax;   int $64

trap-table index for syscalls
movl $6, %eax;  int $64

trap-table index for syscalls
Example

How to emulate an `lidt` call.

Review IDT table…
Example

How to emulate an \texttt{lidt} call.

Review IDT table…

Bootup of VMM and guest OS.
VMM
create table
lidt
switch to guest

H/W

Guest OS

user mode

Memory:

idt
VMM
create table
lidt
switch to guest

H/W
user mode

Guest OS
create table

Memory:
idt
VMM
create table
lidt
switch to guest

H/W
user mode

Guest OS
create table
lidt

Memory:

idt

???
create table
lidt
switch to guest

user mode

create table
lidt

Memory:
create table
lidt
switch to guest

user mode
create table
lidt

kernel mode

Memory:
VMM
create table
lidt
switch to guest

H/W
user mode
create table
lidt

kernel mode
store guest idt addr

Guest OS
time

Memory:
create table
lidt
guest idt

(idt)
VMM

create table
lidt
switch to guest

H/W

user mode

store guest idt addr

kernel mode

Guest OS

create table
lidt

Memory:
VMM
create table
lidt
switch to guest

H/W
user mode
create table lidt
kernel mode
store guest idt addr

Guest OS
time

Memory:
vmm timer

vmm timer

guest timer
Timer Interrupt Handlers

**Host Trap Handler**

```c
void tick() {
  if (...) {
    switch OS;
  } else {
    call OS tick;
  }
}
```

**Guest OS Trap Handler**

```c
void tick() {
  maybe switch process;
  return-from-trap;
}
```
timer interrupt!
Hypervisor decides to keep running Linux
Linux tries to return-from-trap to P2, H/W intercepts and switches to Hypervisor.
Hypervisor

Linux

OS X

Windows
Hypervisor switches to P2 for Linux.
timer interrupt!

- P1
- P2
- P3

- Linux
- OS X
- Windows

Hypervisor
Linux

OS X

Windows

Hypervisor
Linux

OS X

Windows

Hypervisor
timer interrupt!

- P1
- P2
- P3

- Linux
- OS X
- Windows

Hypervisor
Hypervisor decides to switch to Windows.
Linux

OS X

Windows

Hypervisor
Windows tries to return-from-trap to P2, H/W intercepts and switches to Hypervisor.
timer interrupt!

Linux  | OS X  | Windows

Hypervisor
Example

How to emulate an `lidt` call.

Review IDT table…

Bootup of VMM and guest OS.
Example

How to emulate an `lidt` call.

Review IDT table...

Bootup of VMM and guest OS.

What if process in guest calls `lidt`?
P1 calls lidt!
Linux

OS X

Windows

Hypervisor
Linux kills P1. Privileged?
Linux

OS X

Windows

Hypervisor
Linux tries to return-from-trap to P2. Privileged?
System Calls

System calls must also have the VMM in the middle…
Process

Guest OS

VMM

system call:
trap to OS
Process

system call:
trap to OS

Guest OS

call os Trap handler
(at reduced privilege)

VMM

process trapped:

time
Process
system call:
trap to OS

Guest OS

VMM

process trapped:
call os Trap handler
(at reduced privilege)

OS trap handler:
decode trap, exec syscall
return-from-trap
Process

- system call:
  - trap to OS

Guest OS

- OS trap handler:
  - decode trap, exec syscall
  - return-from-trap

VMM

- process trapped:
  - call os Trap handler
    - (at reduced privilege)

- OS tried return-from-trap:
  - do real return-from-trap
Process

- system call: trap to OS

Guest OS

OS trap handler:
- decode trap, exec syscall
- return-from-trap

VMM

- process trapped: call os Trap handler (at reduced privilege)
- OS tried return-from-trap: do real return-from-trap

Resume execution:
- (@PC after trap)
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Challenge: x86 behavior

Inconsistent semantics for “privileged” instructions

Desired behavior:
- **kernel mode**: just do it
- **user mode**: trap to kernel mode

Actual behavior:
- **kernel mode**: just do it
- **user mode**: do something different

Example: popf
# EFLAGS Register

The EFLAGS register is a special register in the x86 architecture that contains flags and indicators used to control various aspects of the processor's operation.

## Flag Descriptions

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Abbreviation</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CF</td>
<td>Carry flag</td>
<td>Status</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PF</td>
<td>Parity flag</td>
<td>Status</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>AF</td>
<td>Adjust flag</td>
<td>Status</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ZF</td>
<td>Zero flag</td>
<td>Status</td>
</tr>
<tr>
<td>7</td>
<td>SF</td>
<td>Sign flag</td>
<td>Status</td>
</tr>
<tr>
<td>8</td>
<td>TF</td>
<td>Trap flag (single step)</td>
<td>Control</td>
</tr>
<tr>
<td>9</td>
<td>IF</td>
<td>Interrupt enable flag</td>
<td>Control</td>
</tr>
<tr>
<td>10</td>
<td>DF</td>
<td>Direction enable flag</td>
<td>Control</td>
</tr>
<tr>
<td>11</td>
<td>OF</td>
<td>Overflow flag</td>
<td>Status</td>
</tr>
<tr>
<td>12-13</td>
<td>IOPL</td>
<td>I/O privilege level (286+ only), always 1 on 8086 and 186</td>
<td>System</td>
</tr>
<tr>
<td>14</td>
<td>NT</td>
<td>Nested task flag (286+ only), always 1 on 8086 and 186</td>
<td>System</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Reserved, always 1 on 8086 and 186, always 0 on later models</td>
<td></td>
</tr>
</tbody>
</table>

- **overflow?**
- **zero?**
- **interrupts enabled?**
- **which bits are privileged?**

# EFLAGS Register

![Table of EFLAGS Register](https://en.wikipedia.org/wiki/FLAGS_register)

- **Bit #**
- **Abbreviation**
- **Description**
- **Category**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Abbreviation</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CF</td>
<td>Carry flag</td>
<td>Status</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PF</td>
<td>Parity flag</td>
<td>Status</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>AF</td>
<td>Adjust flag</td>
<td>Status</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ZF</td>
<td>Zero flag</td>
<td>Status</td>
</tr>
<tr>
<td>7</td>
<td>SF</td>
<td>Sign flag</td>
<td>Status</td>
</tr>
<tr>
<td>8</td>
<td>TF</td>
<td>Trap flag</td>
<td>Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(single step)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>IF</td>
<td>Interrupt enable flag</td>
<td>Control</td>
</tr>
<tr>
<td>10</td>
<td>DF</td>
<td>Direction enable flag</td>
<td>Control</td>
</tr>
<tr>
<td>11</td>
<td>OF</td>
<td>Overflow flag</td>
<td>Status</td>
</tr>
<tr>
<td>12-13</td>
<td>IOPL</td>
<td>I/O privilege level (286+ only), always 1 on 8086 and 186</td>
<td>System</td>
</tr>
<tr>
<td>14</td>
<td>NT</td>
<td>Nested task flag (286+ only), always 1 on 8086 and 186</td>
<td>System</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, always 1 on 8086 and 186, always 0 on later models</td>
<td>Status</td>
<td></td>
</tr>
</tbody>
</table>

- overflow?
- zero?
- interrupts enabled?
- which bits are privileged?

pushf/popf example

pushf and popf backup and restore registers on function call and return.

“the effect of the POPF/POPFD instructions on the EFLAGS register changes slightly, depending on the mode of operation of the processor. When the processor is operating in protected mode at privilege level 0 (or in real-address mode, which is equivalent to privilege level 0), all the non-reserved flags in the EFLAGS register except the VIP, VIF, and VM flags can be modified. The VIP and VIF flags are cleared, and the VM flag is unaffected.”

http://x86.renejeschke.de/html/file_module_x86_id_250.html
pushf/popf example

User Mode:
- Process
- OS

Kernel Mode:
- VMM
pushf/popf example

User Mode:
- Process: popf: don't trap
- OS: popf: need trap

Kernel Mode:
- VMM
pushf/popf example

User Mode:
- Process: popf: don't trap
- OS: popf: need trap

Kernel Mode:
- VMM

Options:
- **modify OS** to trigger trap (insert some trapping instruction)
- **modify H/W** to distinguish between process and OS
Fixing x86

- OS Source
- OS Binary
- Hardware
Fixing x86

- OS Source
- paravirtualization
- OS Binary
- Hardware
Fixing x86

- OS Source
- paravirtualization
- OS Binary
- Binary translation
- Hardware
Fixing x86

- OS Source
  - paravirtualization
- OS Binary
- Hardware
  - binary translation
  - hardware assist
Fixing x86

- OS Source
  - paravirtualization

- OS Binary
  - binary translation

- Hardware
  - hardware assist
OS Modification

Hypercalls
- call from OS to VMM (like syscall, which is from process to OS)
- use some instruction guaranteed to trap
- replace `popf` (and similar) with hypercalls

Paravirtualization
- simple: changing C code
- one-time cost
- problem: can’t support generic operating systems (e.g., Windows)

Binary translation
- tricky: x86 code, can’t break addresses
- translate for each OS instance
- advantage: supports all operating systems
Translation Optimizations

Example: **rdtsc** (read timestamp counter)
- used to get high-precision timestamp
- pretty safe, but useful in certain attacks
- OS/VMM can optionally make it privileged, triggering a trap

If VMM is faking **rdtsc** anyway, why bother trapping?

**rdtsc** → **fake_time++**

(hypothetical translation)
rdtsc by OS

process

OS

VMM

rdtsc
rdtsc by OS

process

OS

VMM

rdtsc
fake_time++
The diagram illustrates the process of `rdtsc` by the OS in the context of virtualization (VMM). The OS reads the current time using `rdtsc` and increments `fake_time`. The process continues to run.
popf by OS

process

OS

popf

VMM
popf by OS

process

OS

popf
hypercall

VMM
popf by OS
popf by process
popf by process

process

OS

VMM

popf

keep running
system call by process

process

getpid

OS

VMM
system call by process

process

getpid

OS

VMM
system call by process

remaining challenge: how to eliminate extra transitions?
Fixing x86

- OS Source
  - paravirtualization

- OS Binary
  - binary translation

- Hardware
  - hardware assist
Two Modes

User Mode:
- Process
- OS

Kernel Mode:
- VMM
Three Modes

User Mode: Process

Guest Mode: OS

Host Mode: VMM
Mode Switching Instructions

Between Process and OS
- trap
- return-from-trap

Between OS and VMM
- vm-exit
- vm-run

H/W responsibilities
- change CPU mode
- save/restore state of processes and operating systems
Performance

OS Source

paravirtualization

OS Binary

binary translation

Hardware

hardware assist
Performance

OS Source
paravirtualization

OS Binary

Hardware

binary translation vs.
hardware assist
some traps can be avoided with in-guest emulation

modern hypervisors typically use both techniques
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
How to get more pages?

**Process**: asks politely, with `sbrk` or `mmap syscall`

**OS**: just uses it!

VMM needs to intercept such usage, and automatically allocate the pages.
Virt Addr Space

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

“Physical” Memory

<table>
<thead>
<tr>
<th>PFN</th>
<th>MFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Machine Memory

<table>
<thead>
<tr>
<th>MFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Problem: TLB needs to map virtual addr \( \Rightarrow \) machine addr
Example TLB State

- VPN 0 => MFN 4
- VPN 1 => MFN 1

if process accesses virtual page 3, it will TLB miss. How to update TLB?
Page Table Lookup

(1) S/W managed TLB

(2) H/W virtualization assist

(3) Shadow page table
(1) S/W managed TLB

S/W managed TLBs interrupt OS VMM on miss.

VMM can do lookup on whatever structures it likes, including the nested PT layout.

Note: every read to an OS-level page-table entry or page-directory entry requires doing VMM-level translation.
Page Table Lookup

(1) S/W managed TLB

(2) H/W virtualization assist

(3) Shadow page table
H/W assist gave us extra CPU mode

Modern x86 CPUs also understand nested page tables.
Page Table Lookup

(1) S/W managed TLB

(2) H/W virtualization assist

(3) Shadow page table
(3) Shadow Page Table

Old x86 CPUs only understand regular page tables.

Compute one from the two page tables.
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Virt Addr Space | “Physical” Memory | Machine Memory
---|---|---
0 | 0 | 0
1 | 1 | 1
2 | 2 | 2
3 | 3 | 3
4 | 4 | 4
5 | 5 | 5
### OS Page Table
- VPN 0 => PFN 2
- VPN 1 => PFN 0
- VPN 3 => PFN 5

### VMM Page Table
- PFN 0 => MFN 1
- PFN 2 => MFN 4
- PFN 5 => MFN 2

### Shadow Page Table
- VPN 0 => MFN 4
- VPN 1 => MFN 1
- VPN 3 => MFN 2

---

**Virt Addr Space**

**“Physical” Memory**

**Machine Memory**
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Shadow Page Table
VPN 0 => MFN 4
VPN 1 => MFN 1
VPN 3 => MFN 2

Virt Addr Space
“Physical” Memory
Machine Memory
0 1 2 3
0 1 2 3 4 5 6 7
0 1 2 3 4 5

PTBR
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Shadow Page Table
VPN 0 => MFN 4
VPN 1 => MFN 1
VPN 3 => MFN 2

Virt Addr Space
0
1
2
3

“Physical” Memory
0
1
2
3
4
5
6
7

Machine Memory
0
1
2
3
4
5

OS sets
PTBR
Virt Addr Space      | “Physical” Memory      | Machine Memory
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Shadow Page Table
VPN 0 => MFN 4
VPN 1 => MFN 1
VPN 3 => MFN 2

PTBR
TRAP!
OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Shadow Page Table
VPN 0 => MFN 4
VPN 1 => MFN 1
VPN 3 => MFN 2

Virt Addr Space
0
1
2
3

“Physical” Memory
0
1
2
3
4
5
6
7

Machine Memory
0
1
2
3
4
5
what if OS disables paging in kernel mode?

**OS Page Table**
- VPN 0 => PFN 2
- VPN 1 => PFN 0
- VPN 3 => PFN 5

**VMM Page Table**
- PFN 0 => MFN 1
- PFN 2 => MFN 4
- PFN 5 => MFN 2

**Shadow Page Table**
- VPN 0 => MFN 4
- VPN 1 => MFN 1
- VPN 3 => MFN 2

**PTBR**
what if OS disables paging in kernel mode?

OS Page Table
VPN 0 => PFN 2
VPN 1 => PFN 0
VPN 3 => PFN 5

VMM Page Table
PFN 0 => MFN 1
PFN 2 => MFN 4
PFN 5 => MFN 2

Shadow Page Table
VPN 0 => MFN 4
VPN 1 => MFN 1
VPN 3 => MFN 2

Virtual Address Space

"Physical" Memory

Machine Memory

Challenge: Cleanup

How does VMM know when to free a shadow PT?

These look the same to the VMM:
• OS killed the process
• OS has stopped scheduling the process for a while

Solution: shadow page tables are just a cache
Challenge: Building the Shadow

When should shadow PTEs be populated?

(1) when we context switch to process:
   - interpose on return-from-trap
   - high up-front cost

(2) when the translation is needed:
   - fast context switch
   - how do we switch to the VMM before translation is used?
Challenge: Building the Shadow

When should shadow PTEs be populated?

(1) when we context switch to process:
   • interpose on return-from-trap
   • high up-front cost

(2) when the translation is needed:
   • fast context switch
   • how do we switch to the VMM before translation is used?
   • use protection bits
## Challenge: Building the Shadow

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>PFN</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>rw-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>rw-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>r-x</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>r-x</td>
</tr>
</tbody>
</table>
Challenge: Building the Shadow

### OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

### VMM Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
</tbody>
</table>

### Shadow Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

If x86 does a page table walk, it will trap to OS because everything is protected.
### Challenge: Building the Shadow

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PFN</strong></td>
<td><strong>PFN</strong></td>
<td><strong>PFN</strong></td>
</tr>
<tr>
<td><strong>protection</strong></td>
<td><strong>protection</strong></td>
<td><strong>protection</strong></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>rw-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>rw-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>r-x</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>r-x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>rwx</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>rwx</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>rwx</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>rwx</td>
</tr>
<tr>
<td>0</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>---</td>
</tr>
</tbody>
</table>

process accesses virtual page 2
**Challenge: Building the Shadow**

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>PFN</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>rw-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>rw-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>r-x</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>r-x</td>
</tr>
</tbody>
</table>

H/W finds entry in shadow, traps into VMM
### Challenge: Building the Shadow

#### OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

#### VMM Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>MFN</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>rwx</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>rwx</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>rwx</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>rwx</td>
</tr>
</tbody>
</table>

#### Shadow Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>MFN</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>---</td>
</tr>
</tbody>
</table>

VMM finds that 2 maps to 1, which maps to 100
# Challenge: Building the Shadow

## OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

## VMM Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rwx</td>
</tr>
<tr>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>2</td>
<td>rwx</td>
</tr>
<tr>
<td>3</td>
<td>rwx</td>
</tr>
</tbody>
</table>

## Shadow Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

VMM updates PTE 2
### Challenge: Building the Shadow

#### OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

#### VMM Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
</tbody>
</table>

#### Shadow Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
### Challenge: Building the Shadow

#### OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

#### VMM Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
</tbody>
</table>

#### Shadow Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Later, process accesses virtual page 2
Challenge: Building the Shadow

OS Page Table

<table>
<thead>
<tr>
<th>PFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

VMM Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rwx</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
</tbody>
</table>

Shadow Page Table

<table>
<thead>
<tr>
<th>MFN</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

H/W finds entry in shadow, updates TLB
## Challenge: Building the Shadow

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>PFN</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>rw-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>rw-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>r-x</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>r-x</td>
</tr>
</tbody>
</table>

process continues
Challenge: Invalidating the Shadow

What if OS changes the page table for a process?
  • how will VMM know to update shadow?
Challenge: Invalidating the Shadow

What if OS changes the page table for a process?
- how will VMM know to update shadow?

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>PFN</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>r-x</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Challenge: Invalidating the Shadow

What if OS changes the page table for a process?
- how will VMM know to update shadow?

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>MFN</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>300</td>
</tr>
</tbody>
</table>

OS moves page
**Challenge: Invalidating the Shadow**

What if OS changes the page table for a process?
- how will VMM know to update shadow?

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>protection</td>
<td>PFN</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Challenge: Invalidating the Shadow

What if OS changes the page table for a process?
- how will VMM know to update shadow?

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
<th>Shadow Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PFN</strong></td>
<td><strong>MFN</strong></td>
<td><strong>MFN</strong></td>
</tr>
<tr>
<td><strong>protection</strong></td>
<td><strong>protection</strong></td>
<td><strong>protection</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>?</td>
</tr>
</tbody>
</table>

100 is wrong
What if OS changes the page table for a process?

- how will VMM know to update shadow?

(1) discard page table on **TLB flush** by OS

- if OS changes PT without TLB flush, staleness is possible
- if staleness could happen anyway, shadow doesn’t “make it worse”
- correct, but slow
Challenge: Invalidating the Shadow

What if OS changes the page table for a process?
- how will VMM know to update shadow?

(1) discard page table on TLB flush by OS
- if OS changes PT without TLB flush, staleness is possible
- if staleness could happen anyway, shadow doesn’t “make it worse”
- correct, but slow

(2) trace updates to process page table
- process page table lives in machine memory
- disable writes to that machine memory using protection in VMM PT
- if guest OS updates process PT, H/W will trap to VMM
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Outline

Machine Virtualization Overview
CPU Virtualization (Trap-and-Emulate)
CPU Virtualization (Modern x86)
Memory Virtualization
Performance Challenges
Information Gap

OS’s were not built to run on top of a VMM. (less true than it used to be)

H/W interface does not give VMM enough info about guest OS.

In particular, is the OS using all its resources?
Information Gap

OS’s were not built to run on top of a VMM. (less true than it used to be)

**H/W interface** does not give VMM enough info about guest OS.

In particular, is the OS using all its resources?

Examples of waste from xv6…
void scheduler(void) {
    struct proc *p;
    for(;;){
        // Enable interrupts on this processor.
        sti();
        // Loop over process table looking for process to run.
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE)
                continue;
            ...
        }
        release(&ptable.lock);
    }
}
void scheduler(void) {
    struct proc *p;
    for(;;){
        // Enable interrupts on this processor.
        sti();
        // Loop over process table looking for process to run.
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE)
                continue;
            ...
        }
        release(&ptable.lock);
    }
}

How does the VMM know to give CPU to another OS?
struct {
    struct spinlock lock;
    struct run *freelist;
} kmem;

// first address after kernel loaded from ELF file
extern char end[];

// Initialize free list of physical pages.
void kinit(void) {
    char *p;

    initlock(&kmem.lock, "kmem");
    p = (char*)PGROUNDUP((uint)end);
    for(; p + PGSIZE <= (char*)PHYSTOP; p += PGSIZE)
        kfree(p);
}
struct {
    struct spinlock lock;
    struct run *freelist;
} kmem;

// first address after kernel loaded from ELF file
extern char end[];

// Initialize free list of physical pages.
void kinit(void) {
    char *p;

    initlock(&kmem.lock, "kmem");
    p = (char*)PGROUNDUP((uint)end);
    for(; p + PGSIZE <= (char*)PHYSTOP; p += PGSIZE)
        kfree(p);
}
// Allocate page tables and physical memory to grow process.
// Returns new size or 0 on error.
int allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
    char *mem;
    uint a;
    a = PGROUNDUP(oldsz);
    for(; a < newsz; a += PGSIZE){
        mem = kalloc();
        memset(mem, 0, PGSIZE);
        mappages(pgdir, (char*)a, PGSIZE, PADDR(mem), PTE_W|PTE_U);
    }
    return newsz;
}
Waste 3 (vm.c)

// Allocate page tables and physical memory to grow process.
// Returns new size or 0 on error.
int allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
    char *mem;
    uint a;
    a = PGROUNDUP(oldsz);
    for(; a < newsz; a += PGSIZE){
        mem = kalloc();
        memset(mem, 0, PGSIZE);
        mappages(pgdir, (char*)a, PGSIZE, PADDR(mem), PTE_W|PTE_U);
    }
    return newsz;
}
Summary

VM’s have overheads.

Architecture impacts ease of implementation.

New opportunities for sharing often outweigh the disadvantages, as utilization is improved.
The Turtles Project: Design and Implementation of Nested Virtualization

More fun…