Virtual Machines

Tyler Harter
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** (get rid of this)

**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

P1  P2  P3

Linux

VMM

Hardware
Now

Virtual Machine Monitor

Hardware

Linux

OS X

Windows
Now

Hardware

P1  P2  P3  P1  P2  P3  P1  P2  P3

Linux  OS X  Windows

Hypervisor

Hardware
Now

Hardware

Linux

OS X

Windows

Hypervisor

Hardware

guests
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.
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!

User Mode:
- Process
- Process
- Process

Kernel Mode:
- OS
- OS
**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

- User Mode:
  - Process
  - OS
  - Process
  - OS
  - Process

- Kernel Mode:
  - VMM
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
```

- **Process P**
- **RAM**
- **trap-table index for syscalls**
movl $6, %eax;   int $64

trap-table index for syscalls
RAM

Process P
Example

How to emulate an `lidt` call.

Review IDT table...
Example

How to emulate an `lidt` call.

Review IDT table…

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

Memory:

idt
VMM
create table
lidt
switch to guest

H/W

Guest OS

Memory:

user mode
create table
lidt
switch to guest

user mode

create table

Memory:

idt
create table
lidt
switch to guest

user mode

create table
lidt

Memory:

idt

???
VMM

create table
lidt
switch to guest

H/W

user mode

Guest OS

create table
lidt

kernel mode

Memory:

idt
VMM
create table
lidt
switch to guest

H/W
user mode
create table
lidt

Guest OS
guest idt

Memory:
store guest idt addr
guest idt

idt

time
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
switch to guest

user mode

create table lidt

store guest idt addr

kernel mode

Memory:

vmm timer

guest timer
Timer Interrupt Handlers

**Host Trap Handler**

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

**Guest OS Trap Handler**

```c
tick() {
    maybe switch process;
    return-from-trap;
}
```
timer interrupt!
Hypervisor

Linux

OS X

Windows
Hypervisor decides to keep running Linux
Linux tries to return-from-trap to P2, H/W intercepts and switches to Hypervisor.
Linux

OS X

Windows

Hypervisor
Hypervisor switches to P2 for Linux.
timer interrupt!
Linux

OS X

Windows

Hypervisor
timer interrupt!

- Linux
- OS X
- Windows

Hypervisor
Hypervisor decides to switch to Windows.
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 tries to return-from-trap to P2. Privileged?
System Calls

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

system call:
trap to OS

time
Process

- system call: trap to OS

Guest OS

VMM

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

Guest OS

- OS trap handler:
  - decode trap, exec syscall
  - return-from-trap
- resume execution:
  - (@PC after trap)

VMM

- OS tried return-from-trap:
  - do real return-from-trap
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 part of the x86 architecture that contains flags used by the processor. Here is a table showing the bits and their 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>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 (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>Reserved, always 1 on 8086 and 186, always 0 on later models</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## EFLAGS Register

<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></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PF</td>
<td>Parity flag</td>
<td>Status</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Reserved</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>Reserved</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?**

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: \textit{popf}: don't trap
- OS: \textit{popf}: need trap

Kernel Mode: VMM

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

OS Source → OS Binary → Hardware
Fixing x86

OS Source

paravirtualization

OS Binary

binary translation

Hardware
Fixing x86

- OS Source
  - paravirtualization

- OS Binary
  - Binary translation

- Hardware
  - 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 on every execution
- advantage: supports all operating systems
popf by OS

process

OS

cropf

VMM
popf by OS
popf by OS
popf by process
popf by process

process

VMM

OS
Translation Optimizations

Example rdtsc
  • used to
system call by process

process

getpid

OS

VMM
system call by process

process

getpid

OS

VMM
system call by process

how to eliminate extra transitions?
TODO: rdtsc

process

popf

keep running

OS

VMM
Fixing x86

- OS Source
- paravirtualization

- OS Binary
- binary translation

- Hardware
- hardware assist
Fixing x86

- OS Source
- Hardware
- Binary translation
- Hardware assist
- Paravirtualization
Two Modes

User Mode:

- Process
- OS

Kernel Mode:

- VMM
Two Modes

User Mode: Process

Guest Mode: OS

Host Mode: VMM
TODO: vm-run, vm-exit, save/restore
TODO: BT vs H/W assist
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. How? (assume software-managed TLB)
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

<table>
<thead>
<tr>
<th>Virt Addr Space</th>
<th>“Physical” Memory</th>
<th>Machine Memory</th>
</tr>
</thead>
<tbody>
<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></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS Page Table</td>
<td>VMM Page Table</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>VPN 0 =&gt; PFN 2</td>
<td>PFN 0 =&gt; MFN 1</td>
<td></td>
</tr>
<tr>
<td>VPN 1 =&gt; PFN 0</td>
<td>PFN 2 =&gt; MFN 4</td>
<td></td>
</tr>
<tr>
<td>VPN 3 =&gt; PFN 5</td>
<td>PFN 5 =&gt; MFN 2</td>
<td></td>
</tr>
</tbody>
</table>

Strategy: store VPN => MFN mapping in TLB.
Strategy: store VPN => MFN mapping in TLB.

- OS tries to insert VPN => PFN to TLB
- VMM intercepts it, looks up in its PT, inserts VPN => MFN

Examples…
Strategy: store $\text{VPN} \Rightarrow \text{MFN}$ mapping in TLB.

- OS tries to insert $\text{VPN} \Rightarrow \text{PFN}$ to TLB
- VMM intercepts it, looks up in its PT, inserts $\text{VPN} \Rightarrow \text{MFN}$

**Examples…**

**Timeline…**

<table>
<thead>
<tr>
<th>OS Page Table</th>
<th>VMM Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN 0 =&gt; PFN 2</td>
<td>PFN 0 =&gt; MFN 1</td>
</tr>
<tr>
<td>VPN 1 =&gt; PFN 0</td>
<td>PFN 2 =&gt; MFN 4</td>
</tr>
<tr>
<td>VPN 3 =&gt; PFN 5</td>
<td>PFN 5 =&gt; MFN 2</td>
</tr>
<tr>
<td>Process</td>
<td>Guest OS</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Mem load</td>
<td></td>
</tr>
<tr>
<td>TLB miss: trap</td>
<td></td>
</tr>
</tbody>
</table>
Process
Mem load
TLB miss: trap

Guest OS

VMM
Call OS TLB handler (reducing privilege)
Mem load
TLB miss: trap

Extract VPN from VA.
Do page table lookup.
Get PFN, update TLB

Call OS TLB handler (reducing privilege)
Process
Mem load
TLB miss: trap

Guest OS
Extract VPN from VA.
Do page table lookup.
Get PFN, update TLB

VMM
Call OS TLB handler (reducing privilege)

Unprivileged code trying to update TLB! Tried to install VPN-to-PFN.
Insert VPN-to-MFN.
Jump back to OS.
Mem load
TLB miss: trap

Extract VPN from VA.
Do page table lookup.
Get PFN, update TLB

Unprivileged code trying to update TLB! Tried to install VPN-to-PFN. Insert VPN-to-MFN. Jump back to OS.

return from trap

Call OS TLB handler (reducing privilege)
Mem load
TLB miss: trap

Extract VPN from VA.
Do page table lookup.
Get PFN, update TLB

Unprivileged code trying to update TLB! Tried to install VPN-to-PFN.
Insert VPN-to-MFN.
Jump back to OS.

return from trap

Call OS TLB handler (reducing privilege)

Return from trap.
Process
Mem load
TLB miss: trap

Guest OS
Extract VPN from VA.
Do page table lookup.
Get PFN, update TLB

VMM
Call OS TLB handler (reducing privilege)

Unprivileged code trying to update TLB! Tried to install VPN-to-PFN. Insert VPN-to-MFN. Jump back to OS.

Resume execution: (@PC of instruction)

Return from trap.
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);
    }
}
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);
}
Waste 2 (kalloc.c)

```c
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);
}
```

How does the VMM know to give pages to another OS?
// 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;
}
// 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;
}

How does OS know page is already zeroed?
Summary

VM’s have **overheads**.

The existing **H/W interface** is restrictive.

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