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

**Cloud**: Amazon ec2, Microsoft Azure, DigitalOcean
Virtualization Software

**Desktop**: VMware, VirtualBox

**Cloud**: Amazon ec2, Microsoft Azure, DigitalOcean

Demos…
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

P1 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P3
---|----|----|----|----|----|----|----|----
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`?

**Guest**: What should a hypervisor do when a guest OS tries to call something like `lidt`?
Virtual CPU
Example

How to emulate an `lidi`t call.
Example

How to emulate an `lidt` call.

Review IDT table...
movl $6, %eax;  int $64
movl $6, %eax; int $64

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

trap-table index for syscalls
Process P

RAM

keyboard

timer

segfault
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

Memory:
VMM

create table
lidt

idt
create table
lidt
switch to guest
create table
lidt
switch to guest

user mode

create table

Memory:

VMM

H/W

Guest OS
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

Memory:

idt

lidt

idt
create table lidt
switch to guest

user mode

create table lidt

kernel mode

Memory:
create table
lidt
switch to guest

user mode
create table
lidt

store guest idt addr

kernel mode

Memory:

VMM

H/W

Guest OS
create table lidt
switch to guest

user mode
create table lidt
kernel mode
store guest idt addr

Memory:
create table lidt
switch to guest

user mode
create table lidt

kernel mode
store guest idt addr

Memory:

vmm timer

guest timer
**Timer Interrupt Handlers**

**Host Trap Handler**

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

**Guest OS Trap Handler**

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

- Linux
- OS X
- Windows

Hypervisor
Hypervisor decides to keep running Linux
Linux tries to return-from-trap to P2, H/W intercepts and switches to Hypervisor.
P1  P2  P3  P1  P2  P3  P1  P2  P3

Linux  OS X  Windows

Hypervisor
Hypervisor switches to P2 for Linux.
Linux

OS X

Windows

Hypervisor
timer interrupt!

- P1
- P2
- P3

- Linux
- OS X
- Windows

Hypervisor
Linux

OS X

Windows

Hypervisor
timer interrupt!

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
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…
system call: trap to OS
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

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)
Virtual Memory
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)
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
<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>
</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 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…
Timeline…
Process
Mem load
TLB miss: trap

Guest OS

VMM

time
Mem load
TLB miss: trap

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

Call OS TLB handler (reducing privilege)

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

return from trap

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

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.

Return from trap.

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.

return from trap

Resume execution:
(@PC of instruction)

Return from trap.
Problems
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);
    }
}
Waste 1 (proc.c)

```c
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);
}
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;
}
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.

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

More fun…