15-410
“...1337C0DE...”

Project 4 - “PebPeb”
Nov. 17, 2019

Dave Eckhardt
Dave O'Hallaron
Brian Railing
Synchronization

**Deadline reminders**
- P3extra deadline != P4 deadline
  - This is for real, not a bureaucratic accident
- Don't forget about the book report...
  - Hand-in directories have been created
Acknowledgments

Leadership
- Joshua Wise, Wes Filardo, Ben Blum, Reid Long

Reference host, reference guests
- Elly Jones, Michael Sullivan, Reid Long

Payloads
- Josiah Boning, Matt Schnall, Ryan Pearl, Reid Long, Axel Feldman

Inspirational artwork
- Chris Lu

Inspirational quote
- “The most fun I've had doing systems programming since I took 410.” – Jacob Potter
Outline

P4
- Write a hypervisor!

Impossible!?!?
- “Some restrictions apply”

“A picture is worth 1000 words”
- I'll still need to talk fast

Component inventory

Suggestions
Scope

How can 2 students re-invent VMware in 10 days??

- Page-table tracing
- Page-table *shadowing*
  - Shadow page-table *caching*
- Binary translation of POPF
- Decoding/simulation of key instructions
  - `MOVL %EAX, %CR3`
  - `MOVL %EBX, %CR3`
  - ...etc.
- Virtual interrupts and exceptions
- Decoding I/O port operations into device-I/O requests
Scope

How can 2 students re-invent VMware in 10 days??

- Page-table tracing
- Page-table *shadowing*
  - Shadow page-table *caching*
- Binary translation of POPF
- Decoding/simulation of key instructions
  - MOVL %EAX, %CR3
  - MOVL %EBX, %CR3
  - ...etc.
- Virtual interrupts and exceptions
- Decoding I/O port operations into device-I/O requests

You probably can't
Scope

How can 2 students re-invent VMware in 10 days??

- Page-table tracing
- Page-table *shadowing*
  - Shadow page-table *caching*
- Binary translation of POPF
- Decoding/simulation of key instructions
  - `MOVL %EAX, %CR3`
  - `MOVL %EBX, %CR3`
  - ...etc.
- Virtual interrupts and exceptions
- Decoding I/O port operations into device-I/O requests

You probably can't

- Hey, remind me what are the *kinds* of virtualization?
Paravirtualization is smaller!

- **Virtualization**
  - Page-table conversion
  - Page-table shadowing
    - Shadow page-table caching
  - Binary translation of POPF
  - Decoding/simulation of key instructions
    - `MOVL %EAX, %CR3`
    - `MOVL %EBX, %CR3`
    - ...etc.
  - Virtual interrupts and exceptions
  - Decoding I/O port operations into device I/O requests
  - I/O hypercalls
Paravirtualization

- Page-table tracing
- Virtual interrupts and exceptions
- I/O hypercalls
- Ok, “miscellaneous” hypercalls too
Paravirtualization

The game

- Host kernel will execute the guest kernel in user mode
  - Guest kernel will execute guest programs in user mode
  - Guest user programs will be unaware of the deception
    » After all, user programs always run in user mode
- Guest kernel won't manipulate hardware directly
  - It will politely ask the host kernel (hypervisor) to manipulate hardware for it
Paravirtualization

The game

- Host kernel will execute the guest kernel in user mode
  - Guest kernel will execute guest programs in user mode
  - Guest user programs will be unaware of the deception
    » After all, user programs always run in user mode
- Guest kernel won't manipulate hardware directly
  - It will politely ask the host kernel (hypervisor) to manipulate hardware for it
- You will write the host kernel (hypervisor)
  - You will continue to support Pebbles system calls
    » Shell can fork() and exec() guest kernels
  - You will also support PebPeb hypercalls
    » Guest kernels can launch user programs and switch among them
- You will provide virtual I/O devices to guest kernels
  » Including virtual interrupts
Paravirtualization

Paravirtualization - “PebPeb” Hypercalls

- Page-table tracing
  - hv_setpd(), hv_adjustpg()

- Virtual interrupts and exceptions
  - hv_setidt(), hv_disable_interrupts(), hv_enable_interrupts()
  - hv_iret()

- I/O hypercalls
  - hv_print(), hv_cons_set_cursor_pos(), ...

- Ok, “miscellaneous” hypercalls too
  - hv_magic()
  - hv_exit()

Virtual interrupts

- Timer, keyboard
Picture Time!

**Logical ("marketing") pictures**
- Lots of colored boxes, everybody's happy

**Illogical ("physical") pictures**
- Maybe too many boxes
15-410 Virtual Memory Layout

Kernel Data

Kernel Program

Stack

Stack

Stack

Stack

k-stack

k-stack

k-stack

k-stack

Program

Program

Program

Program

4080 MB

16 MB

[Logical]
15-410 Physical Memory Layout

User Memory: 240 MB
Kernel Memory: 16 MB

[Physical]
Host Kernel, Host User Programs

[Logical]

Red: supervisor-only

HK
Host User Stands alone

[Logical]

HU
Guest Kernel “Sees” Guest User
Guest User Stands Alone
\{\text{Host,Guest}\} \times \{\text{Kernel,User}\} \quad \text{[Logical]}

Both kernels live under \text{USER\_MEM\_START}
The “Real” (not Virtual) Picture

[Physical]

Guest User

Guest Kernel

Host User

Host Kernel

16 MB
The “Real” (not Virtual) Picture

Other Stuff?
Guest User
Guest Kernel
Host Kernel

[Physical]

Host User

16 MB
PebPeb Address Spaces

**Host kernel**
- All of kernel is mapped (0..16M)
- One user program is mapped (17..4096M)

**Host user**
- One user program is mapped
- Kernel is present but invisible until a surprise
PebPeb Address Spaces

**Host kernel**
- All of kernel is mapped (0..16M)
- One user program is mapped (17..4096M)

**Host user**
- One user program is mapped
- Kernel is present but invisible until a surprise

**Guest kernel**
- All of guest kernel is mapped (0..16M)
- One guest user program is mapped (17..4080M)

**Guest user**
- One guest user program is mapped
- Guest kernel is [somewhere] until a surprise
- Host kernel is present but invisible until a surprise
PebPeb Address Spaces

Host kernel
- [You know how to do this]

Host user
- [You know how to do this]

Guest kernel
- How can the guest kernel occupy 0..16M?
  - That range is used by the host kernel!

Guest user
- When a guest user program is running, and a surprise happens, the hardware will enter host kernel mode.
  - How do we get into guest kernel mode?
Once Upon a Time...

"Hey, let's launch a guest kernel!"

[Logical]
Once Upon a Time...

"Hey, let's launch a guest kernel!"

exec("pathos", argv);
Guest Kernel in “Boot VM” Mode

Both kernels live under USER_MEM_START??
Segmentation to the Rescue!

Guest kernel runs in custom segments!
- B: 16M
- L: 4080M
  - (or a bit less)

Guest kernel sees memory starting at 0

Surprises switch %CS and %SS so host kernel works

Guest max frame
USER_MEM_START
Guest frame 0
USER_MEM_START
Component Inventory

What are the pieces?

- Virtual consoles – independent, not tricky
- Creating shifted segments – not very hard
- “Boot VM” page-table generator
- Tweaked guest-kernel ELF loader
- HV console-output – straightforward
- Basic hv_iret()
- Delivery of virtual interrupts to guests
  - Timer, then keyboard – this part is significant
  - Result: “P1 guests” run: cool!
Component Inventory

What are the pieces?

- Virtual consoles – independent, not tricky
- Creating shifted segments – not very hard
- “Boot VM” page-table generator
- Tweaked guest-kernel ELF loader
- HV console-output – straightforward
- Basic hv_iret()
- Delivery of virtual interrupts to guests
  - Timer, then keyboard – this part is significant
  - Result: “P1 guests” run: cool!
- Page-table compiler
  - Conceptually non-trivial
  - Not huge amounts of code
- Advanced hv_iret(), hv_exit(), “mop up”
  - Result: “P3 guests” run: wow!
Grading

All or nothing???
- P4 is actually two different projects
  - Virtual consoles + hosting P1 guests
  - Virtual consoles + hosting P1 guests + hosting P3 guests

The first version is actually fairly educational!
- We expect to award grades around 80% for solid hosting of paravirtualized P1 guests
Summary

P4 – “PebPeb” paravirtualization
Segmentation to the rescue!
Impossible!?!?

- Ok to aim for “P1 guests + virtual consoles”
- Also educational: “1 P1 guest in 1 console”