Lguest:
Implementing The Little Linux Hypervisor

Rusty Russell
OzLabs

Contents

Motivation

Theory

Getting to the sash prompt

Segment protection v paging

Performance

Documentation

Future Work

Motivation

Motivation

A learning experience for paravirt_ops

Motivation

A learning experience for paravirt_ops

A playground for virtualization

Motivation

A learning experience for paravirt_ops

A playground for virtualization

Simplicity

Motivation

A learning experience for paravirt_ops

A playground for virtualization

Simplicity

Linux on Linux, no ABI, all in-tree.

KVM vs lguest

KVM vs lguest

KVM:

lguest:

KVM vs lguest

KVM:

- Full Virtualization

lguest:

KVM vs lguest

KVM:
- Full Virtualization

lguest:
- Paravirtualization

KVM vs lguest

KVM:
- Full Virtualization
- Suspend/Resume

lguest:
- Paravirtualization

KVM vs lguest

KVM:
- Full Virtualization
- Suspend/Resume

lguest:
- Paravirtualization
- Puppies

KVM vs lguest

KVM:
- Full Virtualization
- Suspend/Resume
- Live Migration

lguest:
- Paravirtualization
- Puppies

KVM vs lguest

KVM:	lguest:
- Full Virtualization	- Paravirtualization
- Suspend/Resume	- Puppies
- Live Migration	- Puppies

KVM vs lguest

KVM:
- Full Virtualization
- Suspend/Resume
- Live Migration
- x86-64 support

lguest:
- Paravirtualization
- Puppies
- Puppies

KVM vs lguest

KVM:	lguest:
- Full Virtualization	- Paravirtualization
- Suspend/Resume	- Puppies
- Live Migration	- Puppies
- x86-64 support	- Puppies

KVM vs lguest

KVM:	lguest:
- Full Virtualization	- Paravirtualization
- Suspend/Resume	- Puppies
- Live Migration	- Puppies
- x86-64 support	- Puppies
- “Features”	

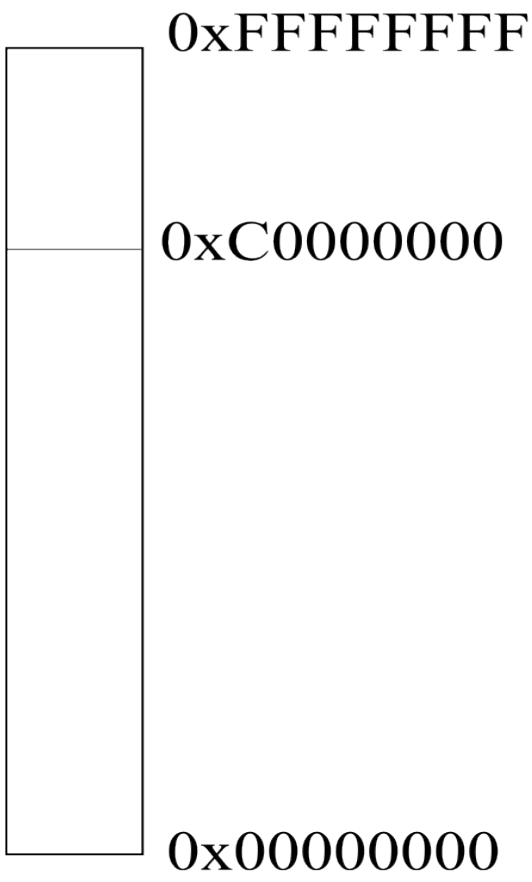
KVM vs lguest

KVM:
- Full Virtualization
- Suspend/Resume
- Live Migration
- x86-64 support
- “Features”

lguest:
- Paravirtualization
- Puppies
- Puppies
- Puppies

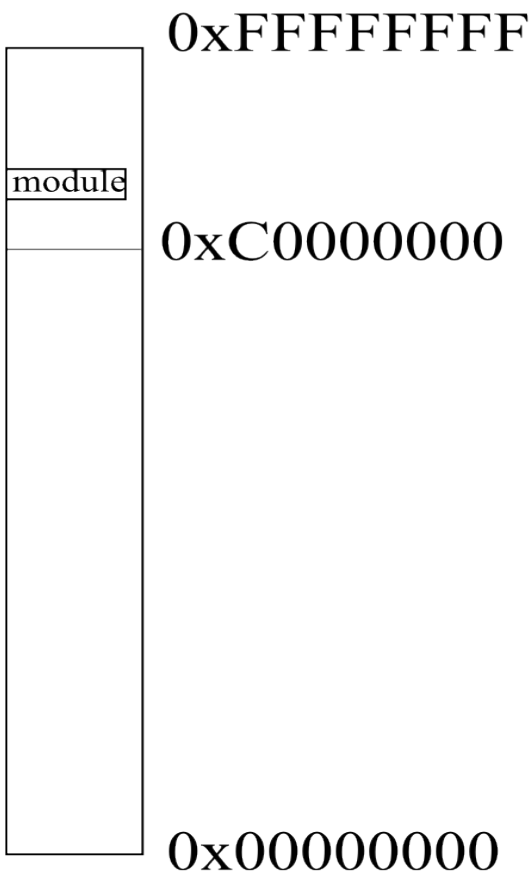
Theory

Theory



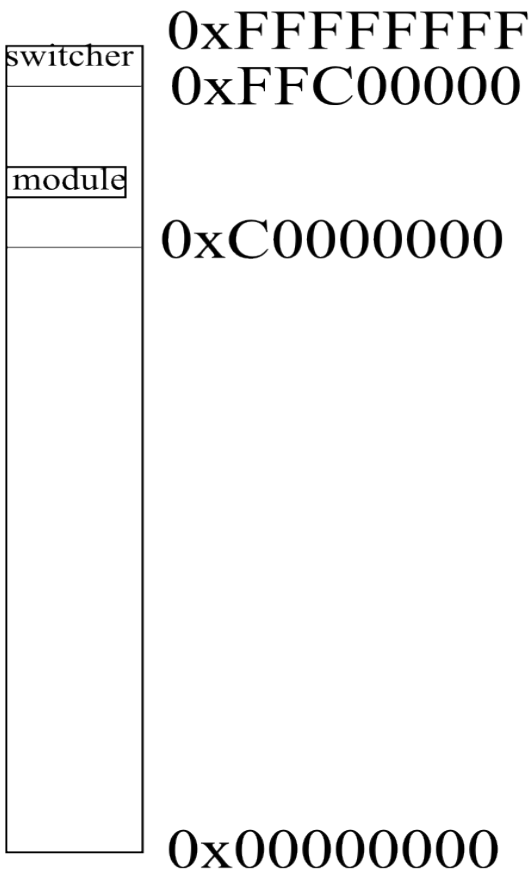
Theory

Kernel module “lg.ko”



Theory

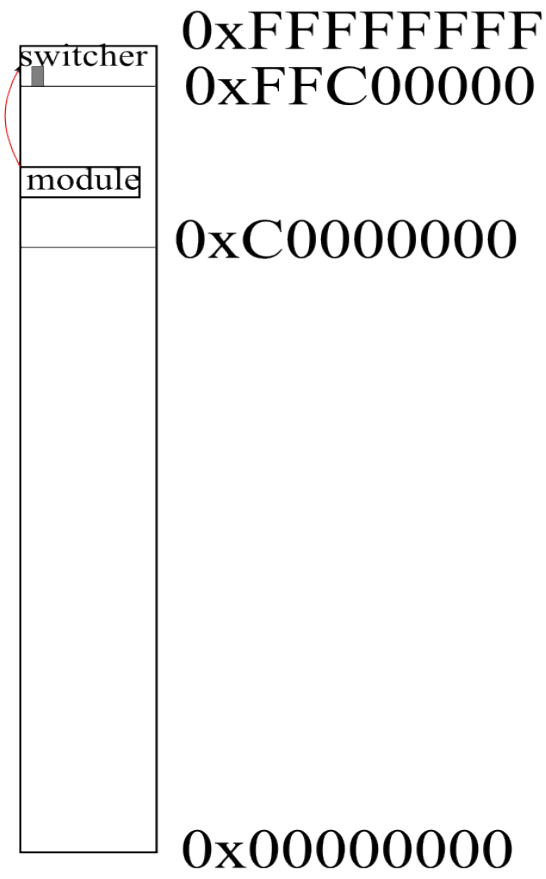
Map “switcher” at -4MB



Theory

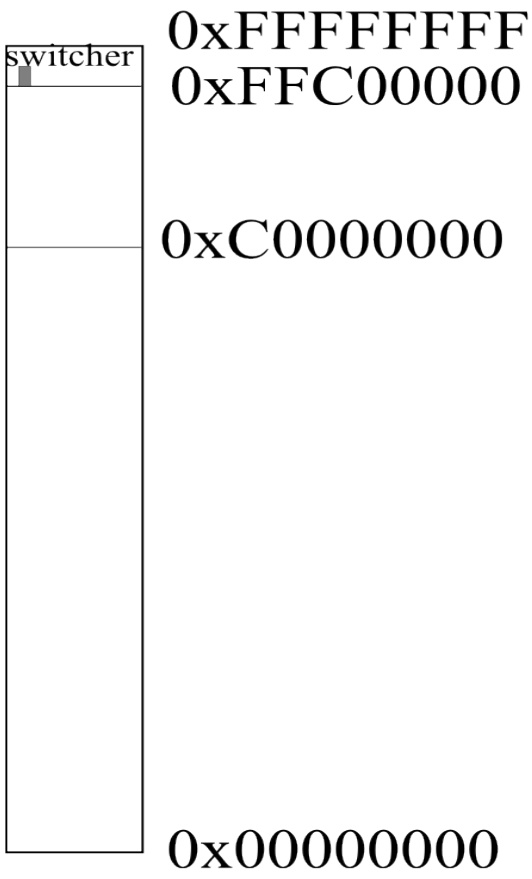
Put regs in switcher area

Jump to switcher code



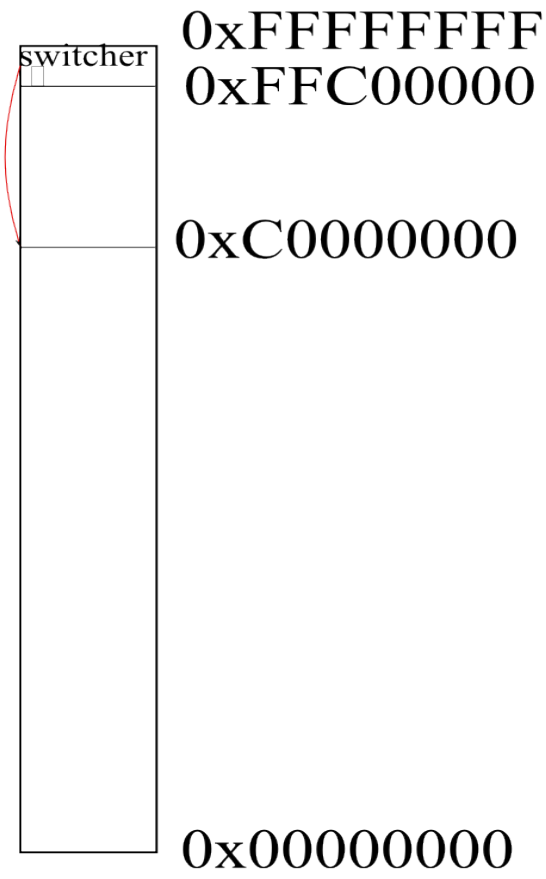
Theory

Switch to guest page tables



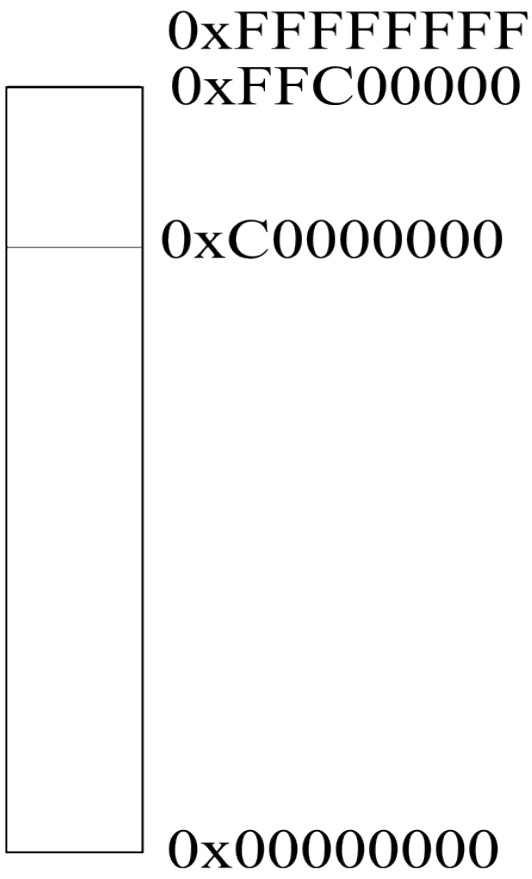
Theory

Pop guest regs, jump into guest



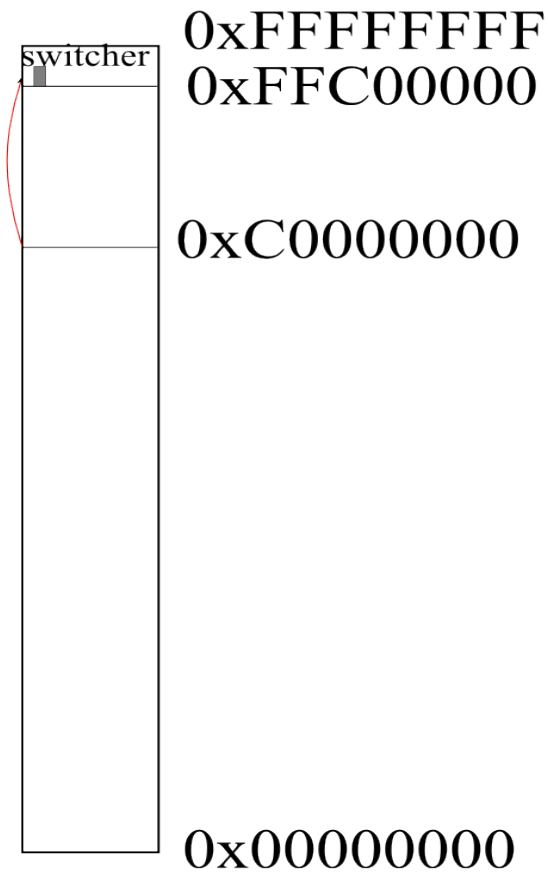
Theory

Guest (PL 1) segments can't reach switcher



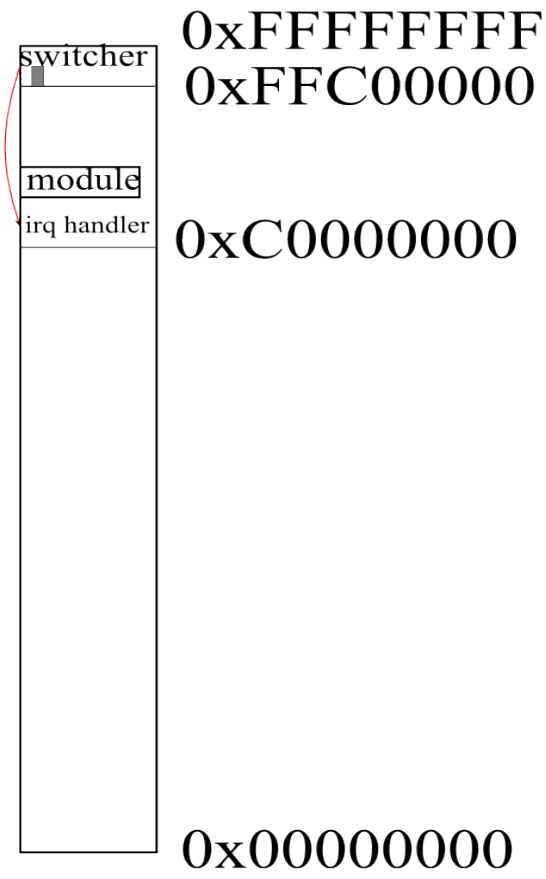
Theory

Interrupt jumps to switcher, saves regs



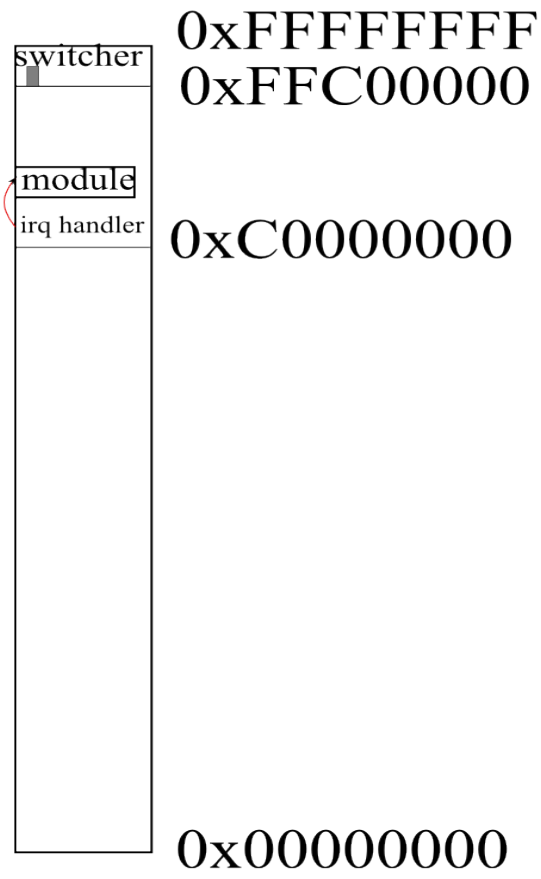
Theory

Switch back to host pagetables, jump to interrupt handler



Theory

Interrupt handler returns to lg.ko



Getting to the sash prompt

Embryonic Switcher

1) Map high address and copy in asm switcher

Embryonic Switcher

1) Map high address and copy in asm switcher
2) switch_to_guest

Embryonic Switcher

1) Map high address and copy in asm switcher
2) switch_to_guest
    1) Save host regs
    2) Change to switcher IDT
    3) Pop guest regs, iret into “guest” loop

Embryonic Switcher

1) Map high address and copy in asm switcher
2) switch_to_guest
    1) Save host regs
    2) Change to switcher IDT
    3) Pop guest regs, iret into “guest” loop
3) switch_to_host

Embryonic Switcher

1) Map high address and copy in asm switcher
2) switch_to_guest
    1) Save host regs
    2) Change to switcher IDT
    3) Pop guest regs, iret into “guest” loop
3) switch_to_host
    1) Save guest regs
    2) Restore host IDT
    3) Jmp to host interrupt handler

Switcher In Full

switch_to_guest:

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)
4) Clear host GDT TSS “used” bit

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)
4) Clear host GDT TSS “used” bit
5) Change to guest page table

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)
4) Clear host GDT TSS “used” bit
5) Change to guest page table
6) Pop guest regs

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)
4) Clear host GDT TSS “used” bit
5) Change to guest page table
6) Pop guest regs
7) iret into guest

Switcher In Full

switch_to_guest:

1) Save host regs
2) Change to guest IDT
3) Change to guest GDT (segments)
4) Clear host GDT TSS “used” bit
5) Change to guest page table
6) Pop guest regs
7) iret into guest
   - Turns interrupts back on
   - Sets privilege level to 1
   - Switches segments to truncated entries

Switcher in Full

switch_to_host:

Switcher in Full

switch_to_host:

1) Stub pushes interrupt number on stack

Switcher in Full

switch_to_host:

1) Stub pushes interrupt number on stack
2) Save guest regs on stack

Switcher in Full

switch_to_host:

1) Stub pushes interrupt number on stack
2) Save guest regs on stack
3) Switch back GDT, IDT, page tables

Switcher in Full

switch_to_host:

1) Stub pushes interrupt number on stack
2) Save guest regs on stack
3) Switch back GDT, IDT, page tables
4) Either:
    1) iret directly back to module (faults), or
    2) jmp to interrupt handler (interrupts)

Guest

Guest

Now we set up page tables to map guest kernel, and actually switch into guest

Guest

Now we set up page tables to map guest kernel, and actually switch into guest

Crash!

Guest

Now we set up page tables to map guest kernel, and actually switch into guest

Crash!

Implement that paravirt_op

- needs hypercall mechanism (int 0x81)

Guest

Now we set up page tables to map guest kernel, and actually switch into guest

Crash!

Implement that paravirt_op

- needs hypercall mechanism (int 0x81)

Repeat...

Page Table Handling

Page Table Handling

“Shadow” page tables

Page Table Handling

“Shadow” page tables

- Guest maintains page tables
- Host maintains real page tables

Page Table Handling

“Shadow” page tables

- Guest maintains page tables
- Host maintains real page tables
  - Switcher nailed in permanently

Page Table Handling

“Shadow” page tables

- Guest maintains page tables
- Host maintains real page tables
  - Switcher nailed in permanently
- paravirt_ops hooks on all PT operations:
  Tell Host to throw away everything

Page Table Handling

"Shadow" page tables

- Guest maintains page tables
- Host maintains real page tables
  - Switcher nailed in permanently
- paravirt_ops hooks on all PT operations:
  Tell Host to throw away everything
- On page fault, Host re-reads Guest PTEs and updates real PTE.

Page Table Handling

"Shadow" page tables

- Guest maintains page tables
- Host maintains real page tables
  - Switcher nailed in permanently
- paravirt_ops hooks on all PT operations:
  Tell Host to throw away everything
- On page fault, Host re-reads Guest PTEs and updates real PTE.

Optimized later to minimize updates.

Guest

- Added primitive “write” hypercall for console

Guest

- Added primitive “write” hypercall for console

```
Checking if this processor honours the WP bit
  even in supervisor mode..
```

Interrupts

Need interrupt delivery!

Interrupts

Need interrupt delivery!

First implement interrupt state ops:

- irq_disable / irq_enable
- save_flags / restore_flags

Interrupts

Need interrupt delivery!

First implement interrupt state ops:

- irq_disable / irq_enable
- save_flags / restore_flags

"init" hypercall tells Host where our "struct lguest_page" is.

- lguest_page contains "irq_enabled" field.

Interrupts

Need interrupt delivery!

First implement interrupt state ops:

- irq_disable / irq_enable
- save_flags / restore_flags

"init" hypercall tells Host where our "struct lguest_page" is.

- lguest_page contains "irq_enabled" field.

Simply manipulate that word.

Interrupts

load_idt_entry hypercall:

Host remembers guest interrupt handlers

set_kernel_stack hypercall:

Host needs to know stack for interrupts

Interrupts

load_idt_entry hypercall:

- Host remembers guest interrupt handlers

set_kernel_stack hypercall:

- Host needs to know stack for interrupts

Host “reflects” traps destined for guest:

- Push old stack, RPL etc on guest kernel stack
- Change program counter (eip) to handler
- May disable interrupts for guest

Interrupts

load_idt_entry hypercall:

- Host remembers guest interrupt handlers

set_kernel_stack hypercall:

- Host needs to know stack for interrupts

Host “reflects” traps destined for guest:

- Push old stack, RPL etc on guest kernel stack
- Change program counter (eip) to handler
- May disable interrupts for guest

iret hypercall:

- pops off stack, restores interrupt state etc.

Interrupts

- Later lguest added direct traps
  - Especially useful for system call performance
- Also removed iret hypercall: normal iret (almost) works.

Emulation

- PCI code uses in/out instructions to try to find bus

Emulation

- PCI code uses in/out instructions to try to find bus
  - Guest is PL 1, General Protection Fault

Emulation

- PCI code uses in/out instructions to try to find bus
  - Guest is PL 1, General Protection Fault
  - These are not overridden by paravirt_ops

Emulation

- PCI code uses in/out instructions to try to find bus
  - Guest is PL 1, General Protection Fault
  - These are not overridden by paravirt_ops
- => Emulate in/out instructions

Timers

Timers

```
Calibrating delay using timer specific
  routine..
```

Timers

```
Calibrating delay using timer specific
   routine..
```

Now we need a timer interrupt!

Requires registering a “struct irq_chip”

Timers

Checkin 16:

Timers

Checkin 16:

- Interrupt support, and timer interrupt.
- Implement hlt instruction.
- Simple GDT updates (we reload entire table).
- Godawful clock source and wallclock function.
- Panic handler so we kill process on panic.
- Emulation of another I/O instruction.
- Fix io_bitmap_base: previously random I/O ports were allowed (IDE access!)
- Don't try to init lhype module when running under paravirt already.

Timers

Checkin 20:

Boots to sash prompt in initramfs.

Segments v Paging

Segments v Paging

We originally truncated all segments which the Guest can access, so we could protect the Switcher.

Segments v Paging

We originally truncated all segments which the Guest can access, so we could protect the Switcher.

glibc really wants to use 4G segments

Segments v Paging

We originally truncated all segments which the Guest can access, so we could protect the Switcher.

glibc really wants to use 4G segments

- Nasty tricks to allow Guest userspace to use 4G segments, but not Guest kernel.

Segments v Paging

We originally truncated all segments which the Guest can access, so we could protect the Switcher.

glibc really wants to use 4G segments

- Nasty tricks to allow Guest userspace to use 4G segments, but not Guest kernel.

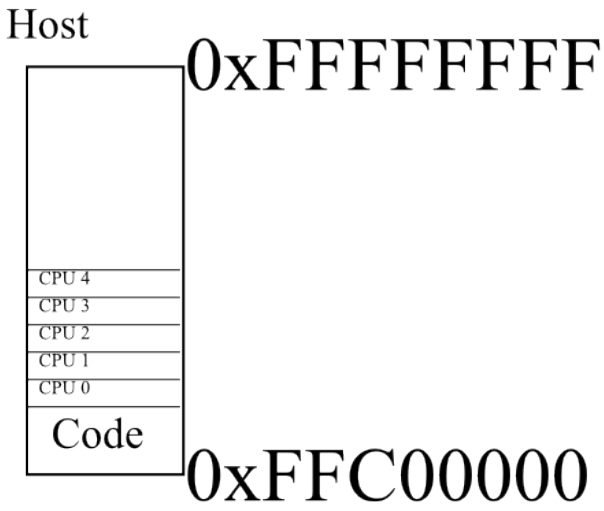
x86-64 can't use segments for isolation anyway.

Segments v Paging

Instead we can use page protection to protect Switcher:

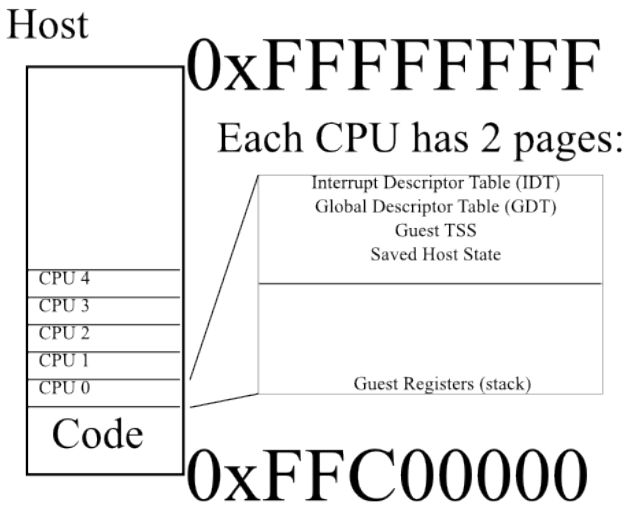
Segments v Paging

Instead we can use page protection to protect Switcher:



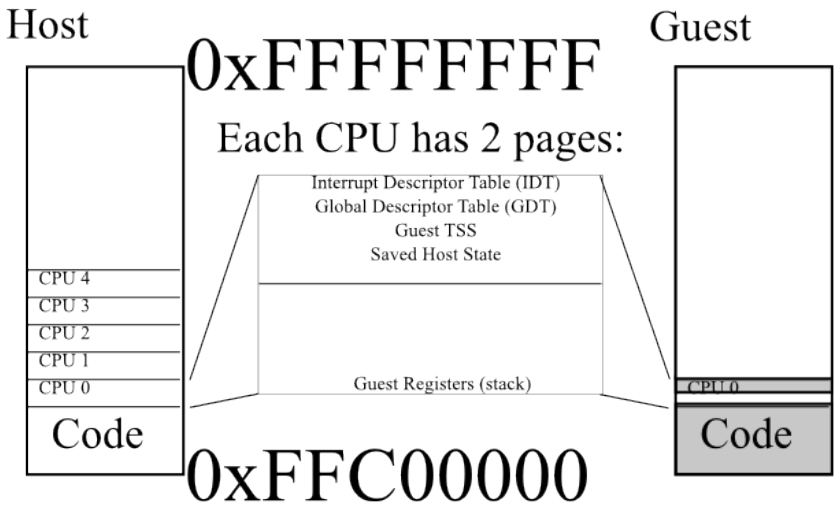
Segments v Paging

Instead we can use page protection to protect Switcher:



Segments v Paging

Instead we can use page protection to protect Switcher:



Segments v Paging

We copy IDT and GDT for this Guest into this CPU's page before we run it

Segments v Paging

We copy IDT and GDT for this Guest into this CPU's page before we run it

- Optimize the common case where nothing has changed

Segments v Paging

We copy IDT and GDT for this Guest into this CPU's page before we run it

- Optimize the common case where nothing has changed

Copy Guest's registers in and out of CPU's register page

Segments v Paging

We copy IDT and GDT for this Guest into this CPU's page before we run it

- Optimize the common case where nothing has changed

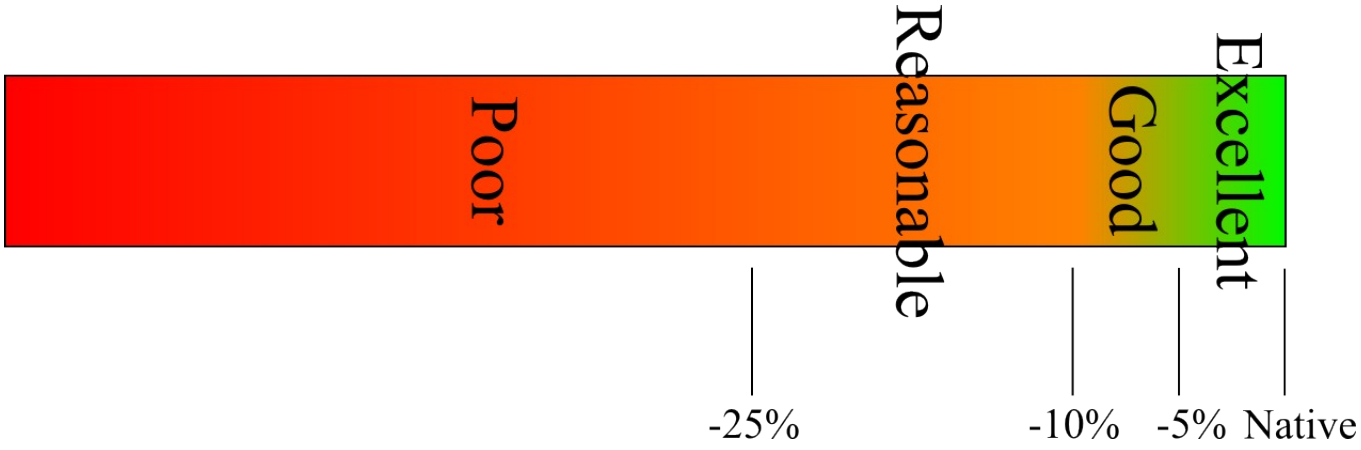
Copy Guest's registers in and out of CPU's register page

- Optimized to just map Guest's registers over CPU's register page in Guest page tables

Performance

Performance

Performance level-setting:



Performance

Micro (virtbench, times in ns)

Performance

Micro (virtbench, times in ns)

Native:

Time for one context switch via pipe: 2349 (2345 - 2359)

Time for one Copy-on-Write fault: 3722 (3570 - 4876)

Time to exec client once: 284296 (279828 - 356812)

Time for one fork/exit/wait: 96500 (88578 - 130750)

Performance

Micro (virtbench, times in ns)

Native:

Time for one context switch via pipe: 2349 (2345 - 2359)

Time for one Copy-on-Write fault: 3722 (3570 - 4876)

Time to exec client once: 284296 (279828 - 356812)

Time for one fork/exit/wait: 96500 (88578 - 130750)

Lguest:

Time for one context switch via pipe: 4839 (4778 - 4949)

Time for one Copy-on-Write fault: 75023 (14218 - 399609)

Time to exec client once: 767500 (749718 - 1268312)

Time for one fork/exit/wait: 362062 (359468 - 432437)

Performance

Page fault speed sucks.

- Every pagefault goes through hypervisor twice (6x slower than native)

Performance

Page fault speed sucks.

- Every pagefault goes through hypervisor twice (6x slower than native)

It would be possible to handle this in the Switcher...

Performance

Macro (kernel compile 512M memory)

Performance

Macro (kernel compile 512M memory)

Native:

real	user	sys
7m21s	5m49s	0m38s
6m53s	5m50s	0m36s
7m13s	5m50s	0m37s

Performance

Macro (kernel compile 512M memory)

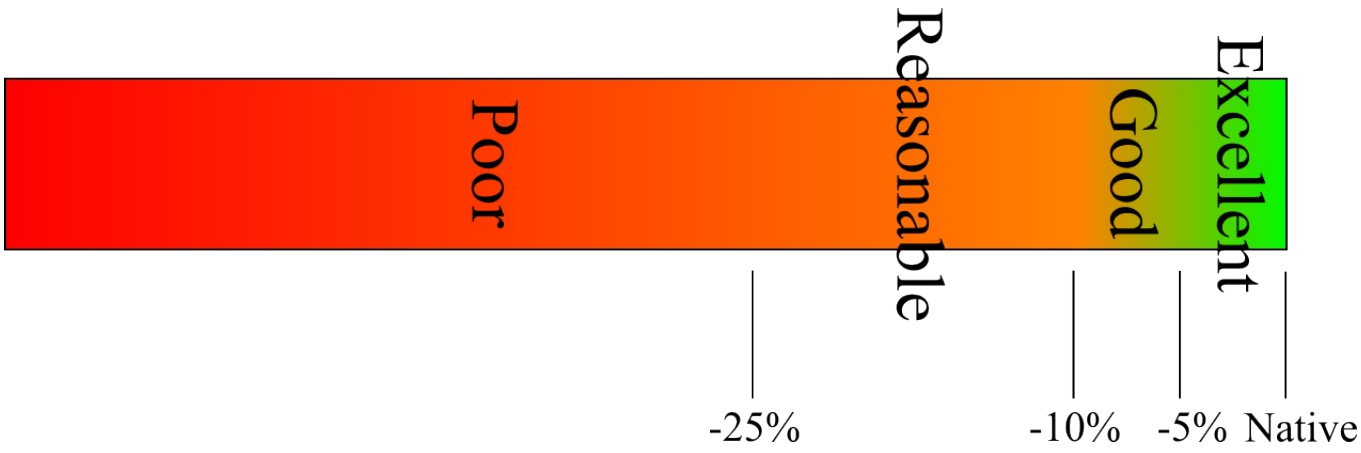
Native:

real	user	sys
7m21s	5m49s	0m38s
6m53s	5m50s	0m36s
7m13s	5m50s	0m37s

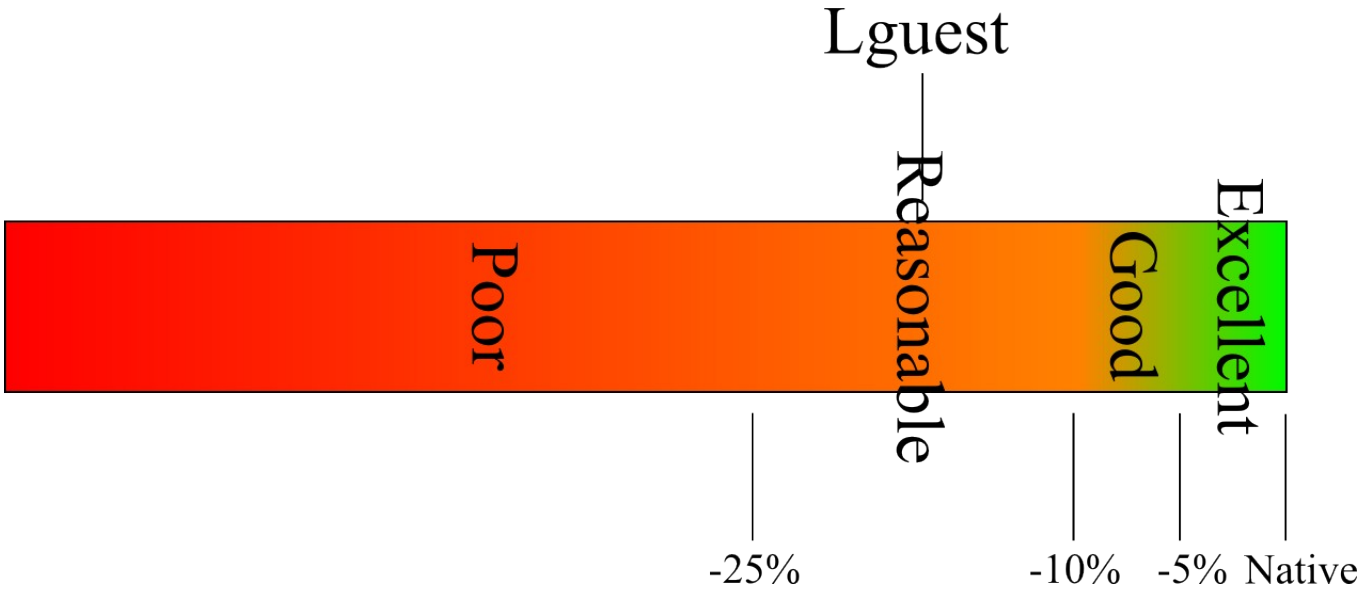
lguest (w/ virtio blk patch):

real	user	sys
8m27s	6m29s	1m26s
8m9s	6m31s	1m21s
8m19s	6m33s	1m23s

Performance



Performance



Performance

Guest wc -l:

Performance

Guest wc -l:

```
  615 drivers/lguest/lguest.c
   56 drivers/lguest/lguest_asm.S
  148 drivers/lguest/lguest_bus.c
  275 drivers/block/lguest_blk.c
  354 drivers/net/lguest_net.c
  102 drivers/char/hvc_lguest.c
   85 include/linux/lguest.h
   48 include/linux/lguest_bus.h
   73 include/linux/lguest_launcher.h
 1756 total
```

Performance

Host wc -l:

```
  463 drivers/lguest/core.c
  184 drivers/lguest/hypercalls.c
  268 drivers/lguest/interrupts_and_traps.c
  415 drivers/lguest/io.c
  238 drivers/lguest/lguest_user.c
  262 drivers/lguest/lg.h
  411 drivers/lguest/page_tables.c
  125 drivers/lguest/segments.c
  159 drivers/lguest/switcher.S
 2525 total
```

Performance

Launcher wc -l:

1012 Documentation/lguest/lguest.c

Performance

Launcher wc -l:

1012 Documentation/lguest/lguest.c

Total: 5293 lines

Documentation

Documentation

User as *hero*

-- Kathy Sierra, LCA 2007 Keynote

Documentation

User as *hero*

-- Kathy Sierra, LCA 2007 Keynote

drivers/lguest/README

Future Work

Today:

- virtio
- native bzImage execution

Future Work

Today:

- virtio
- native bzImage execution

Tomorrow:

- extensible cmdline
- suspend/resume
- x86-64
- non-root launcher
- guest SMP

Fun Experiments!

Fun Experiments

- fork() hypercall

Fun Experiments

- fork() hypercall
- NUMA simulation

Fun Experiments

- fork() hypercall
- NUMA simulation
- grub-like bootloader

Fun Experiments

- fork() hypercall
- NUMA simulation
- grub-like bootloader
- nested lguests

Fun Experiments

- fork() hypercall
- NUMA simulation
- grub-like bootloader
- nested lguests
- puppies!

References

http://lguest.ozlabs.org

- 2.6.21 patch (includes documentation)
- Code (not documentation) is in -mm

Legal Statement

This work represents the views of the author(s) and does not necessarily reflect the views of IBM Corporation.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries: IBM (logo). A full list of U.S. trademarks owned by IBM may be found at http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.