Lguest64 - A new breed of puppies

Glauber de Oliveira Costa
gcosta@redhat.com

Red Hat Inc.

January, 2008

The need for a 64-bit PV

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.
- Not strictly. But we wanted it (HVM enabled x86_64 hardware slightly more common)

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.
- Not strictly. But we wanted it (HVM enabled x86_64 hardware slightly more common)
- Where's the hardware?

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.
- Not strictly. But we wanted it (HVM enabled x86_64 hardware slightly more common)
- Where's the hardware?
- Testbed for the pvops64 patch

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.
- Not strictly. But we wanted it (HVM enabled x86_64 hardware slightly more common)
- Where's the hardware?
- Testbed for the pvops64 patch
- lguest64 - smp from the very beginning

The need for a 64-bit PV

- x86_64 PV not nearly as efficient as i386.
- Not strictly. But we wanted it (HVM enabled x86_64 hardware slightly more common)
- Where's the hardware?
- Testbed for the pvops64 patch
- lguest64 - smp from the very beginning
- Ideas exported into lguest32 (For ex: get rid of the ugly elf loader)

x86_64 - Intrinsically more complicated!

- No segment limit protection

x86_64 - Intrinsically more complicated!

- No segment limit protection
- swapgs all-in-one instruction

x86_64 - Intrinsically more complicated!

- No segment limit protection
- swapgs all-in-one instruction
- syscall instruction always present

x86_64 - Intrinsically more complicated!

- No segment limit protection
- swapgs all-in-one instruction
- syscall instruction always present
- syscalls bounces to hypervisor

x86_64 - Intrinsically more complicated!

- No segment limit protection
- swapgs all-in-one instruction
- syscall instruction always present
- syscalls bounces to hypervisor
- 4-level page tables

x86_64 - Intrinsically more complicated!

- No segment limit protection
- swapgs all-in-one instruction
- syscall instruction always present
- syscalls bounces to hypervisor
- 4-level page tables
- Much room for code sharing, but hard in 2.6.22

No segment limit protection

Forced to use page tables for protection
lguest32 also benefited from it.

host2guest comm

3 pages: (guest perspective)

- HV text - Executable

host2guest comm

3 pages: (guest perspective)

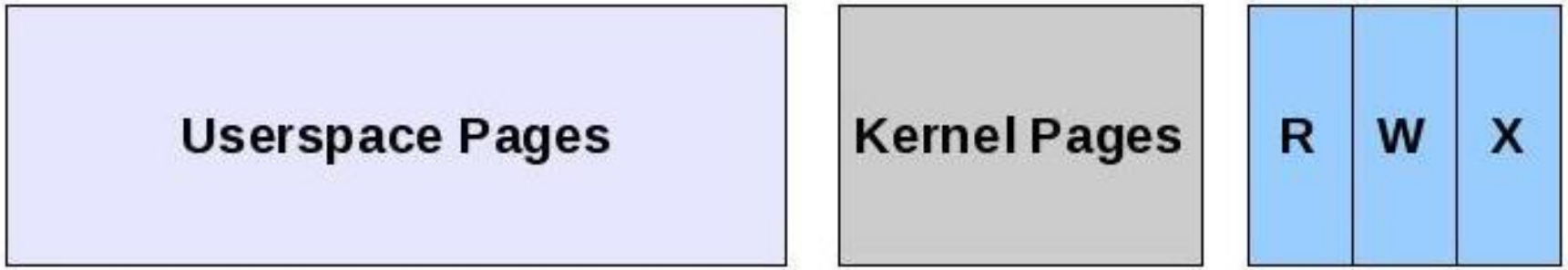
- HV text - Executable
- guest ro area - the vcpu struct, Read Only

host2guest comm

3 pages: (guest perspective)

- HV text - Executable
- guest ro area - the vcpu struct, Read Only
- guest scratch pad - mapped in the same virtual address for all vcpus, RW

What you mean?



Why map in the same virtual address?

Consider the code: (It's guest code)

```
ENTRY(lguest_iret)
        pushl   %eax
        movl    12(%esp), %eax
        movl    %eax,%ss:lguest_data+LGUEST_DATA_irq_enabled
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        popl    %eax
        iret
```

How do you know where to write ? userspace stack, userspace gs, etc

No segment limit protection - Guest kernel

When guest kernel runs: all rw pages can be touched.
Map hypervisor (vcpu_data) RO (with a RW scratch pad - irq state, etc)

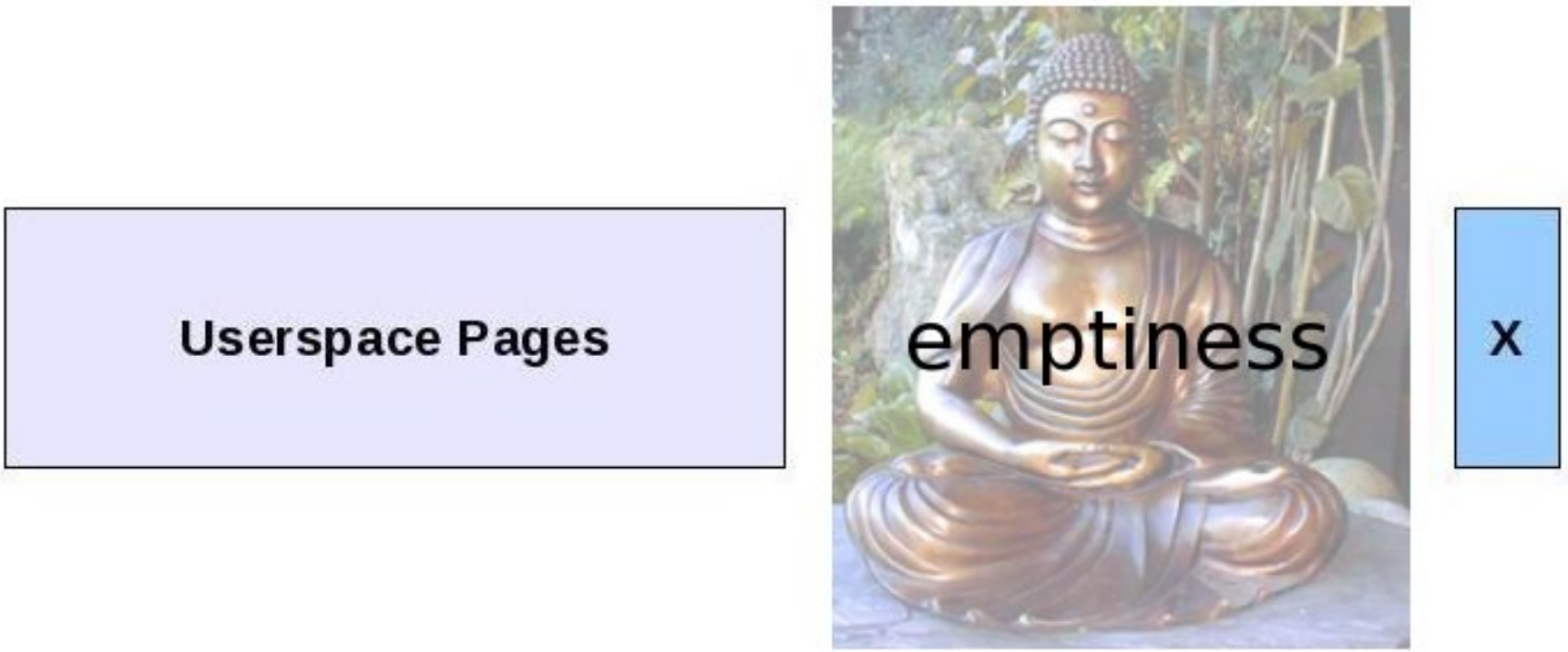
No segment limit protection - switcher

Hypervisor has a lot of updates to do $\rightarrow$ all of them have to happen before cr3 switch

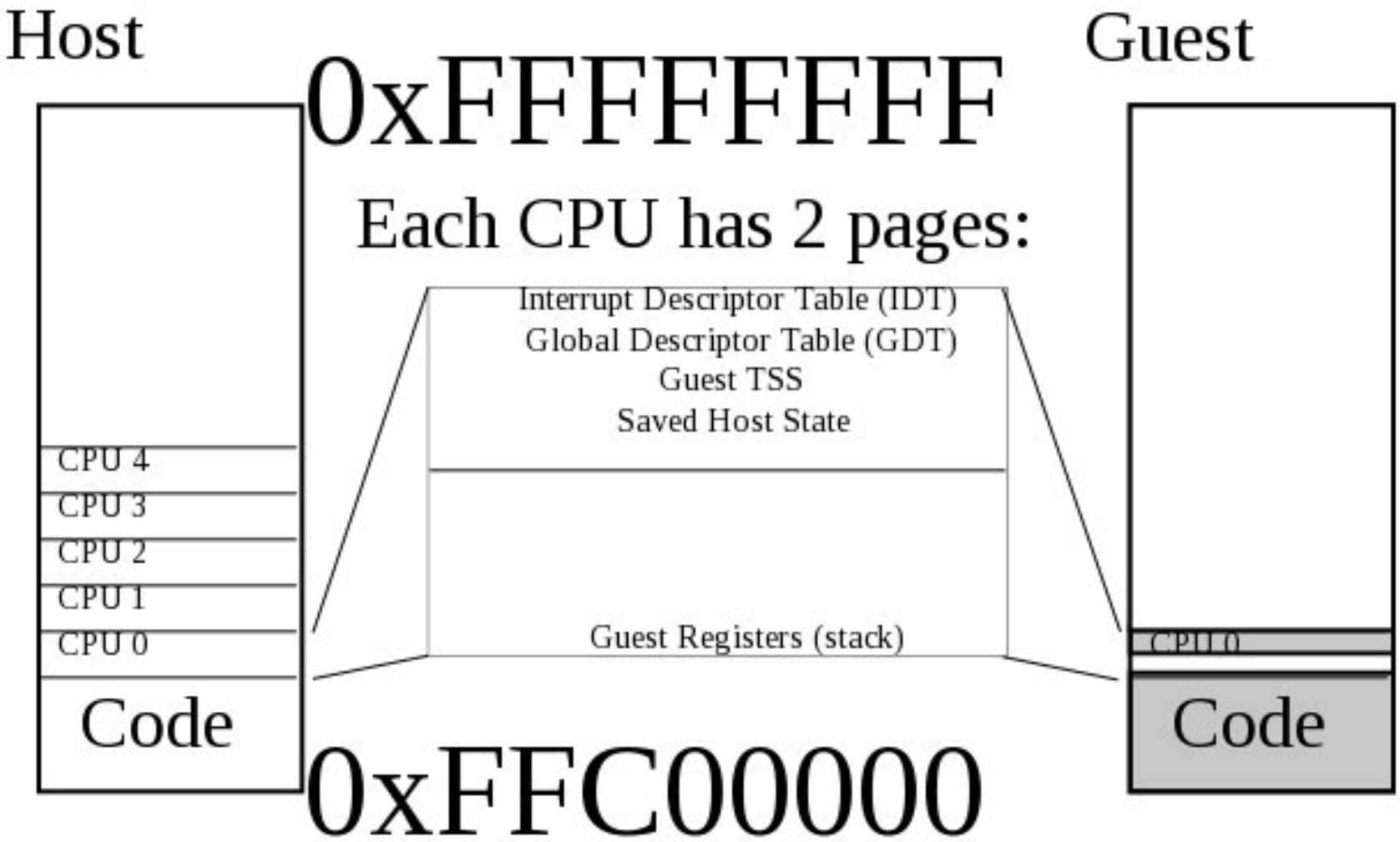
No segment limit protection - userapp

When userspace app runs, no kernel pages are mapped.

Like this:



What does 32-bit do?



Communications

- Extended set of hypercalls over plain lguest
- setup hypercalls use *int 0x80*, switch to *syscall* ASAP.

syscall always present

and always go to privilege 0!

syscall always present

and always go to privilege 0!

- write msr at every run $\longrightarrow$ no mess with userspace host apps

syscall always present

and always go to privilege 0!

- write msr at every run $\longrightarrow$ no mess with userspace host apps
- guest kernel and guest userspace differentiate through a flag

swapgs

Before: Access to kernel data structures
After: Forget about it
(And the other way around too)

swapgs

Before: Access to kernel data structures
After: Forget about it
(And the other way around too)

- Hard to call functions (stack is kernel data)

swapgs

Before: Access to kernel data structures
After: Forget about it
(And the other way around too)

- Hard to call functions (stack is kernel data)
- We made pvops have a symbol that points to syscall after swapgs

swapgs

Before: Access to kernel data structures
After: Forget about it
(And the other way around too)

- Hard to call functions (stack is kernel data)
- We made pvops have a symbol that points to syscall after swapgs
- syscall handler trampoline go straight there

x86_64 system call

```
#define SWAPGS_UNSAFE_STACK swapgs

ENTRY(system_call)
        SWAPGS_UNSAFE_STACK
ENTRY(system_call_after_swapgs)
        movq    %rsp,%gs:pda_oldrsp
        movq    %gs:pda_kernelstack,%rsp
```

4-level page tables

The nastier one: page table updates have to find their corresponding pmd, pud, pgd.
We keep a hash binding to upper level

Other features

- strong statistics
- NMI handling
- But features kill puppies, so no much more.

Current Status

- Long winter due to need of getting pvops64 upstream (x86 merge)
- Strategy is to not even keep trees separated
- Rusty took first part of smp patches (missing the scratch pad)
- Work on progress to make lguest hv functions less 32-bit centric

That's all, Folks!

... Unless you have questions!

Many thanks to Steven Rostedt, who could not unfortunately be here