Lguest

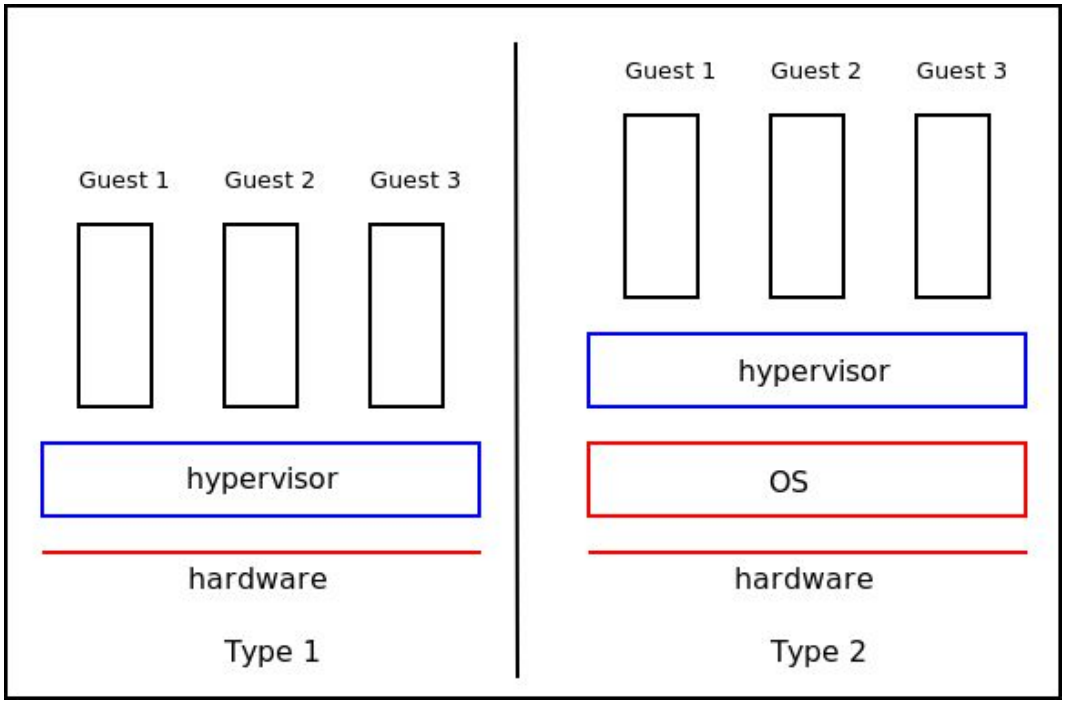
A journey of learning the Linux kernel internals

Daniel Baluta <daniel.baluta@intel.com>

What is lguest?

- “Lguest is an adventure, with you, the reader, as a Hero”
- Minimal 32-bit x86 hypervisor
- Introduced in 2.6.23 (October, 2007) by Rusty Russel & co
- Around 6000 lines of code
- “Relatively” “easy” to understand and hack on
- Support to learn linux kernel internals
- Porting to other architectures is a challenging task

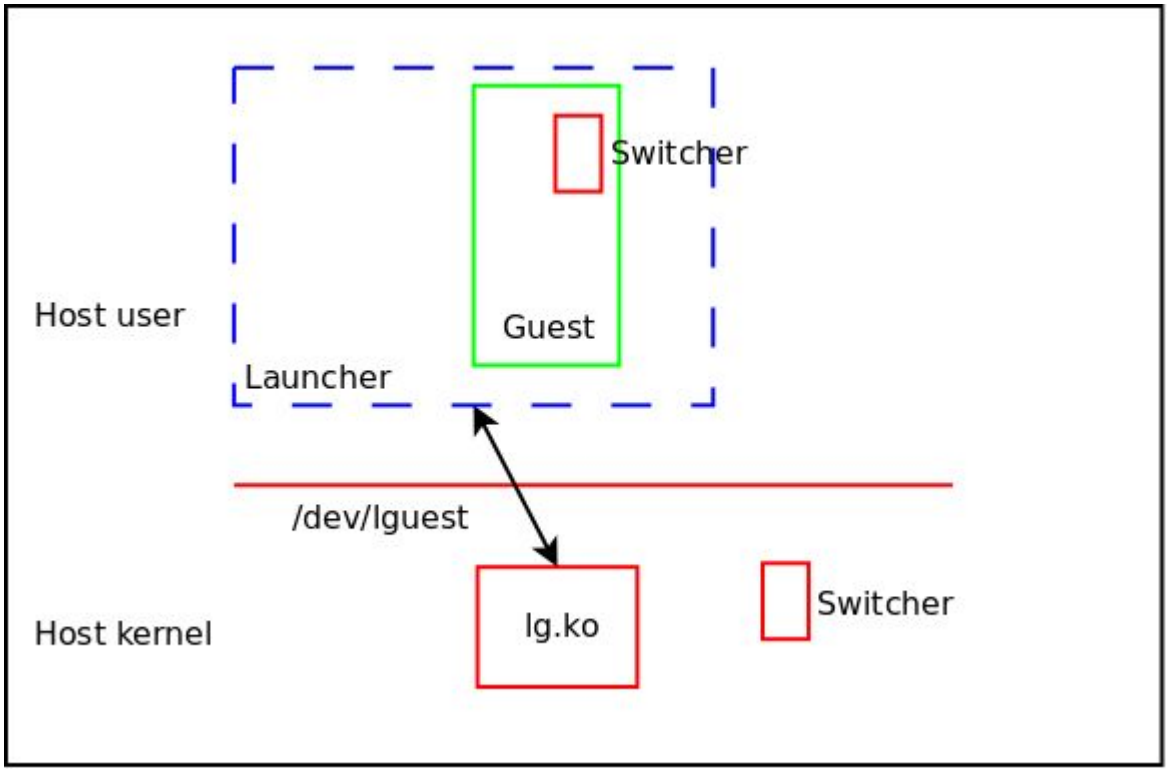
Hypervisor types



make Preparation

- Guest
  - Life of a guest
- Drivers
  - virtio devices: console, block, net
- Launcher
  - User space program that sets up and configures Guests
- Host
  - Normal linux kernel + lg.ko kernel module
- Switcher
  - Low level Host <-> Guest switch
- Mastery
  - What’s next?

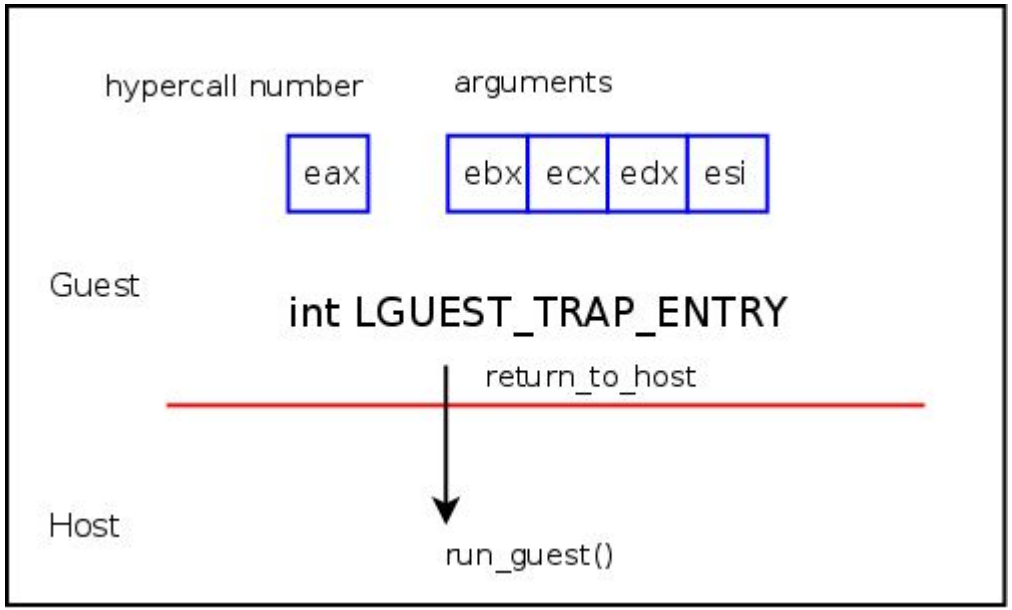
Lguest overview



make Guest

- “Simple creature, identical to the Host [..] behaving in simplified ways”
  - Same kernel as Host (or at least, built from the same source)
- booting trace (when using bzImage)
  - arch/x86/boot/compressed/head_32.S: startup_32
    - uncompresses the kernel
  - arch/x86/kernel/head_32.S : startup_32
    - initialization , checks hardware_subarch field
  - arch/x86/lguest/head_32.S
    - LGUEST_INIT hypercall, jumps to lguest_init
  - arch/x86/lguest/boot.c
    - once we are here we know we are a guest
    - override privileged instructions
    - boot as normal kernel, calling **i386_start_kernel**

Hypercalls


hypercall number
arguments
eax
ebx
ecx
edx
esi
Guest
int LGUEST_TRAP_ENTRY
return_to_host
Host
run_guest()

struct lguest_data

- second communication method between Host and Guest
- LHCALL_GUEST_INIT
  - hypercall to tell the Host where lguest_data struct is
- both Guest and Host publish information here
- optimize hypercalls
  - irq_enabled
  - blocked_interrupts
  - cr2, virtual address of the last page fault
  - wallclock time, set by Host
  - async hypercall ring for batching

Paravirt ops

- Hooks that allow guest to override certain operations
- Interrupt related operations
  - lguest_save_fl
  - lguest_irq_disable
  - lg_irq_enable
  - lg_restore_fl
- CPU instructions
  - lguest_load_gdt
  - etc
- Page table management
  - lguest_set_pte
  - etc

Interrupt control operations

- most commonly called functions from paravirt ops
- implementing using hypercalls is too slow
- Use *lguest_data.irq_enabled* field
  - guest updates this with a single instruction
  - host checks this before it tries to deliver an interrupt
- save_flags
  - Linux cares only about the interrupt flag, this returns irq_enabled
- restore_flags
  - movl %eax, lguest_data + LGUEST_DATA_irq_enabled
- irq_enable / irq_disable
  - movl $X86_EFLAGS_IF, lguest_data + LGUEST_DATA_irq_enabled
  - lguest_data.irq_enabled = 0

Interrupt Descriptor Table

- IDT tells the processor what to do when an interrupt comes in
- IDT entry
  - 64 bit descriptor
  - privilege level, interrupt handler
- hosts controls the real IDT
  - guest asks host to make the IDT entry change
  - LHCALL_LOAD_IDT_ENTRY hypercall
- guests also have a copy of the IDT

Global Descriptor Table

- holds the characteristics for various segments
  - (base, size, access privilege)
- lgdt
  - load global descriptor table
- LHCALL_LOAD_GDT_ENTRY hypercall
- every 8-byte entry in GDT is a descriptor
  - local descriptor table
    - Linux uses this only for strange apps like Wine
    - provide an empty stub to avoid seg fault
  - task state segment
    - used for switching in the past
    - override with do-nothing to avoid seg fault

cpuid

- query CPU identity and its features
- 28 pages in Intel manual!
- lguest_cpuid
  - use native “cpuid”
    - only just to turn features off
    - e.g. : KVM_CPUID_SIGNATURE, PAE, etc.
- cpuid is not privileged
- cat /proc/cpuinfo
  - GenuineIntel

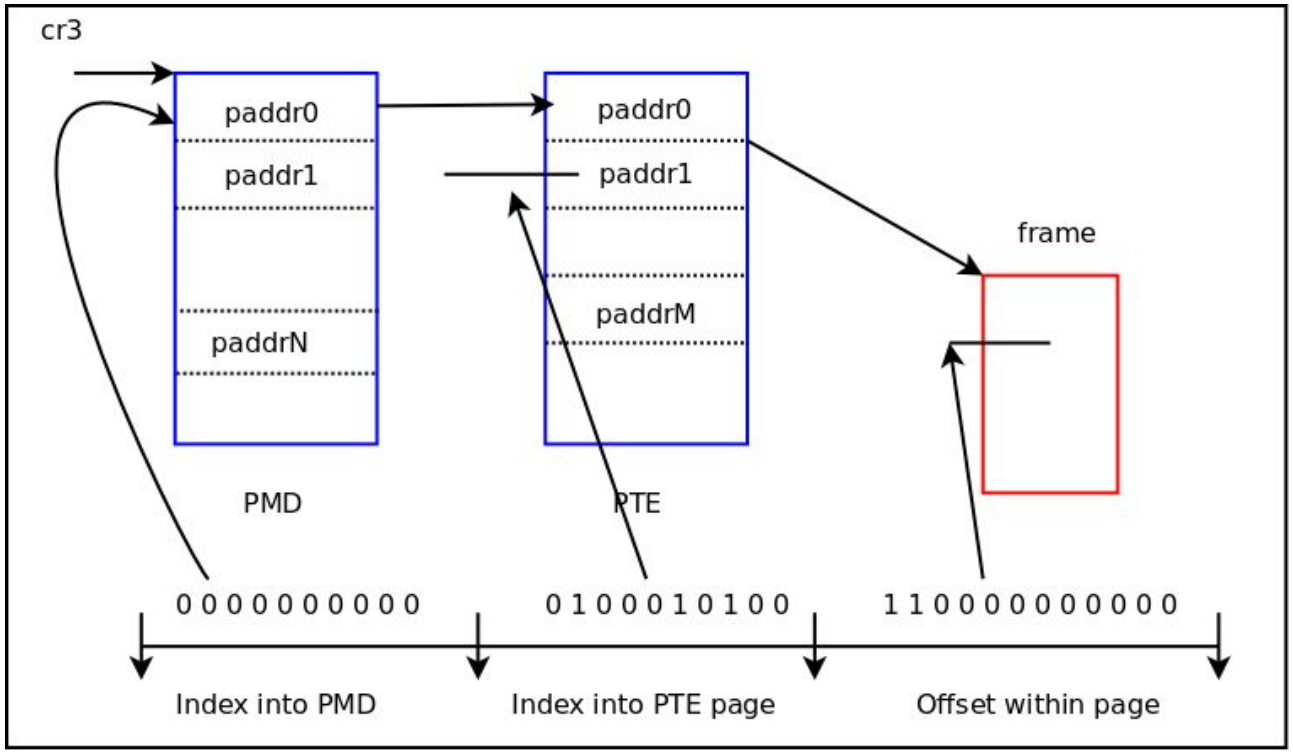
Control registers (cr)

- Host needs to know when the Guest wants to change them
- cr0
  - turns on/off all kind of features, linux only cares of TS (Task Switched)bit
  - lguest_write_cr0, lguest_read_cr0
- cr2
  - virtual address of the last page fault
  - lguest_write_cr2, lguest_read_cr2
- cr3
  - current top level pagetable page
- cr4
  - enable and disable PGE/PAE but we don’t care

Page Table Handling

- Memory is divided into “pages” of 4096 bytes each
- CPU maps virtual addresses to physical addresses using “page tables”
  - we could use one huge index of 2^32 / 2^12 = 2^20 (1 million entries)
  - but most virtual addresses are not used
- Use a 3-level page table to save space
- … but this is not the whole story
  - Intel added Physical Address Extension (PAE) to allow access to 64GB of memory (36 bits)
  - Four level page table
- Only one format available at runtime
  - CONFIG_X86_PAE
- cr3 holds the physical address of top level “page directory” page

Virtual address translation



pv_mmu_ops

- write_cr3
  - same as cr0, keep a local copy and tell host when it changes
  - LHCALL_NEW_PGTABLE
- flush_tlb
  - little cache of page table entries kept by the CPU
  - LHCALL_FLUSH_TLB
- set_pmd
  - LHCALL_SET_PMD
- set_pte
  - LHCALL_SET_PTE

UPIC

- Unadvanced Programmable Interrupt Controller
- lguest_data.blocked_interrupts
  - tell Host what interrupts we want blocked
- struct irq_chip lguest_irq_controller
  - irq_mask
  - irq_unmask
- lguest_setup_irq
  - allocate irq descriptors
- lguest_init_IRQ
  - sets up the IDT entry for each hardware interrupt
  - except 128 (0x80) which is used for system calls

Time

- Guest doesn’t have its own clock
  - Host gives Guest time on every timer interrupt
  - lguest_get_wallclock, lguest_data.time
- clocksource
  - lguest_tsc_khz(), Time Stamp Counter as a clocksource
  - “lguest_clock”, uses lguest_data.time given by the Host
- clockevent
  - deliver interrupts at a specific time in the future
  - LHCALL_SET_CLOCKEVENT
- setup timer interrupt (0)
  - calls in clockevent infrastructure

make Drivers

- “Guest finds its voice and becomes useful”
- no access to real devices
- we emulate a PCI bus with virtio devices on it
  - console
  - block
  - network
- Launcher sets up devices in Guest physical memory
  - PCI_VENDOR_ID 0x1af4
- lguest used to test virtio 1.0 standard

make Launcher

- “Trace back to the creation of a Guest and begin our understanding of the Host”
- Launcher is the Host userspace program
  - sets up, runs and services the Guest
  - does all the device handling for the Guest
- Host kernel interface which the Launcher uses to control the Guest
  - read/write from a (misc) character device
  - /dev/lguest device node
- Launcher executable is in tools/lguest
  - ./lguest --block=<filename> --initrd=<filename> <mem-in-mb> vmlinux [args…]

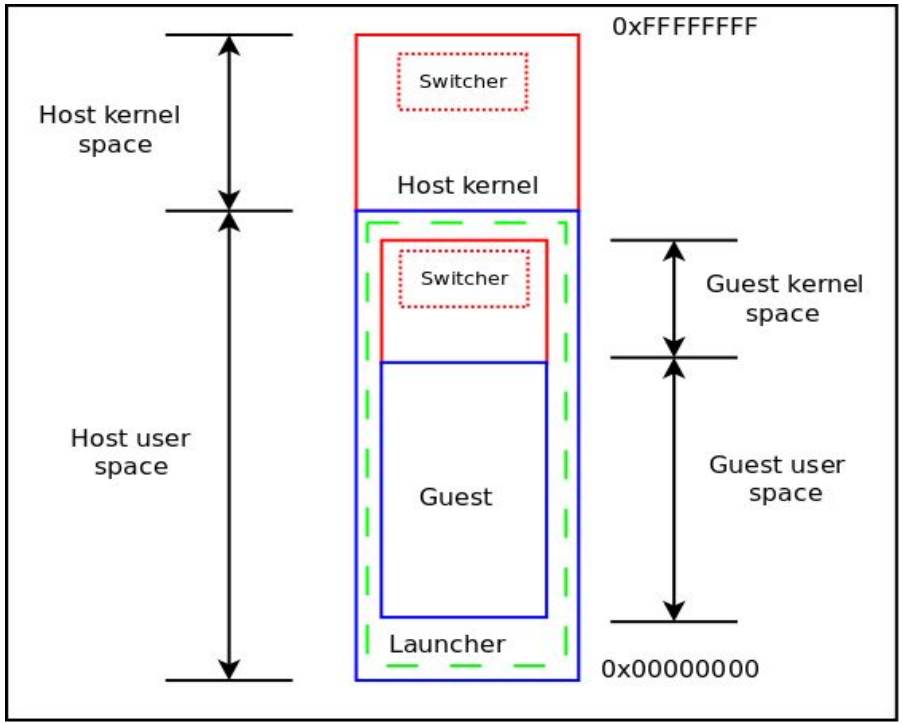
write() on /dev/lguest

- first operation Launcher must do is write() to setup a guest
  - LHREQ_INITIALIZE
  - base, pfn_limit, start, device_limit
  - initialize: clock, registers, pagetables
- other actions
  - send interrupts
    - LHREQ_IRQ, irq_no
  - get/set Guest registers values
    - LHREQ_GETREG/LHREQ_SETREG
  - deliver trap
    - LHREQ_TRAP

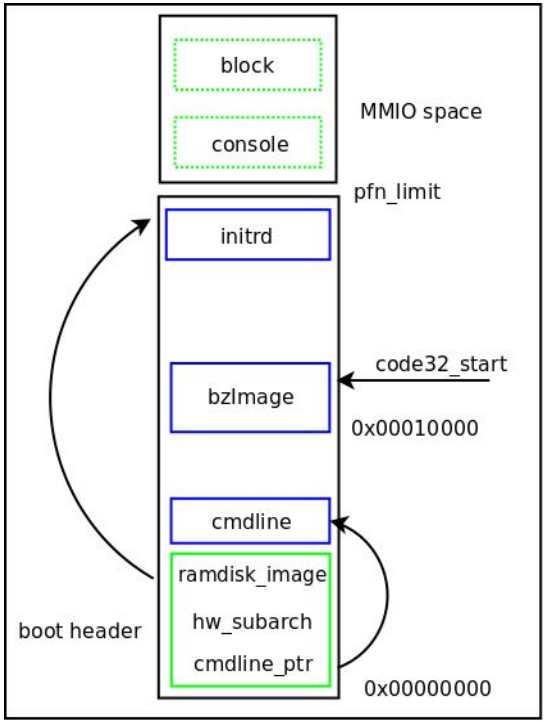
read() on /dev/lguest

- once Lguest is initialized we make it run by reading /dev/lguest
- keep going around until something interesting happens
- run_guest()
  - emulate_insn()
  - emulate_mmio()
  - restart_guest()
- offset is overloaded
  - indicates which CPU number we are dealing with
  - first step in adding SMP support :)

Lguest memory layout



Guest physical memory



Launcher main routine

- initialize devices
  - console, block, network
  - all devices are on PCI bus
- load_kernel
  - vmlinux or bzImage
- load_initrd()
- tell_kernel to initialize the guest()
  - guest_base, guest_limit, start_ip, guest_mmio
- initialize boot header
- run_guest()

make Host

- CONFIG_LGUEST_GUEST
- drivers/lguest/
- normal Linux kernel + lg.ko kernel module
- lg.ko
  - put the switcher up in high virtual memory
  - reserve 2 lguest_pages (after switcher page)
    - lguest_regs
    - host state we need to restore when we switch back
  - reserve interrupt vector
  - register /dev/lguest misc char device

map_switcher()

- few hundred bytes of assembly code
  - assume switcher code is less than a page
  - change CPU to run guest
  - change back to Host when a trap or interrupt occurs
- needs to be mapped at a high virtual address
  - must be at the same virtual address in the Guest as in the Host
- mapped at 0xFFC00000
- we also map the 2 pages for lguest state info

run_guest()

- called by Launcher reading /dev/lguest
- do the hypercalls
- tell Launcher about traps
- delivers pending interrupts
- jump into guest
  - lguest_arch_run_guest()
  - run_guest_once()
- look at why guest exited
  - traps: GPF, PF, hardware interrupts, hypercalls are handled in host
  - the rest is delivered back to the Guest

run_guest_once

- restore guest state
  - copy guest specific information into struct lguest_pages
- push EFLAGS on stack
- do lcall
  - far procedure call
  - change from using the kernel code segment to using the dedicated lguest segment
  - jumps into the switcher

make Switcher

- each CPU has 2 pages visible to Guest when it runs on that CPU
  - copy_in_guest_info
  - has to contain the state of the Guest before we run the guest
- each Guest has “changed” flags
  - contains what changed since last run
- switch_to_guest
- return_to_host
  - due to a trap
- deliver_to_host
  - due to an external interrupt

What's next?

- Mastery
  - guest page faulting
  - add SMP support
- port on other archs
  - x86_64, Steven Rostedt, Glauber de Oliveira Costa
  - arm
  - mips
- short demo booting lguest
- beer!

Q & A

Resources

- http://lguest.ozlabs.org/
- lguest: Implementing the little Linux hypervisor
- http://www.ibm.com/developerworks/library/l-hypervisor/

Kernel address space