kexec'ing xen guests
or redesigning the xen domain builder

Gerd Hoffmann
SUSE Labs / Novell Inc.
kraxel@suse.de

September 5, 2006
Overview

- short kexec introduction.
- domain builder redesign.
- booting using kexec.
- kexec kernel internals.
kexec introduction
kexec system call

struct kexec_segment {
    const void *buf;
    size_t bufsz;
    const void *mem;
    size_t memsz;
};

long kexec_load(void *entry, unsigned long nr_segments,
    struct kexec_segment *segments, unsigned long flags)
{
    return syscall(__NR_kexec_load, entry,
        nr_segments, segments, flags);
}

long kexec_reboot(void)
{
    return syscall(__NR_reboot, LINUX_REBOOT_MAGIC1,
        LINUX_REBOOT_MAGIC2,
        LINUX_REBOOT_CMD_KEXEC, 0);
}
kexec userspace bits

- Most of the magic happens here.
- native hardware: /sbin/kexec utility.
- Xen option one: add xen support to /sbin/kexec.
- Xen option two: add kexec support to xen tools.
  - xen domain builder does a very similar job.
  - current code structure makes reusing it very hard.
  - redesigned domain builder ...

© Novell Inc, Confidential & Proprietary
redesigning the domain builder
Why rewrite?

• historically grown, time to redesign from ground up.
• problems of the current domain builder:
  - still visible it was derived from the dom0 builder, making it unnecessary complicated.
  - xen hypercalls sprinkled all over the place, especially map/unmap, making it hard to reuse for something else.
  - lot of cut&paste programming for different architectures.
Design overview

- Keep all the domain info in a struct, dropping lots of local variables and parameters.
- The domain is built in a contiguous piece of main memory, no need to map/unmap pages all the time.
- Separate out code which does hypercalls, so most code runs fine in other contexts too (kexec).
- Separate out arch-specific code, so the core code just calls another set of functions.
- Pluggable binary loaders.
  - Register them self, have a probe() method.
  - The builder just walks the list and tries one by one.
xm_dom_core.c

- core code
  - allocate and init “struct xc_dom_image”
  - parse: find binary loader, parse kernel.
  - load: load kernel & initrd, setup start_info & friends
  - cleanup everything when done.

- helper code
  - memory pool.
  - load & gunzip files.
xc_dom_elf.c

• binary loader for ELF kernels
• can be compiled multiple times
  - required for 32-on-64 bit support.
xc_dom_${arch}.c

- setup architecture specific data structures
  - page tables
  - start_info page
  - vcpu context

- TODO: have multiple of them compiled in too
  - needed for 32-on-64
  - depends on header changes
    - arch-specific structs need a postfix so you can have them all defined without name clashes.
    - postfix should be _${arch} instead of _32 / _64 IMHO
xc_dom_boot.c

- Boot a domain image.
  - Copy pages to the domain
  - Setup p2m map, adjust page tables.
  - Do the hypercalls needed to launch it.

- Only this code actually invokes hypercalls.

- It is the only file which has lots of #ifdefs right now, maybe split off bits into xc_dom_boot_${arch}.c?
xc_dom_compat.c

• Just glue code to make it work as plug-in replacement for the old domain builder code.
Current state

- I use it all day for domain creation and guest kexec.
- Architectures
  - works for x86 (32bit, PAE, 64bit).
  - ia64 could work in theory but is completely untested.
- Operating systems
  - booting linux works fine.
  - booting netbsd seems to work ok too (only briefly tested).
  - it should load everything the old elf loader is able to handle.
- Patches available here:
  - [http://www.suse.de/~kraxel/patches/](http://www.suse.de/~kraxel/patches/)
What else it maybe is useful for?

- Have the domain builder write out suspend images, then boot via “xm restore”.
- Boot new domains on other machines, using the migration protocol.
- Others ... ?
boot using kexec
xen mini kboot – building blocks

• /init – main kboot script
  - setup block devices, activate raid, activate lvm
  - mount rootfs (and optionally /boot) read-only
  - load kernel & initrd, cleanup, boot kernel.

• select-kernel
  - uses dialog to ask user if multiple kernels are present.
  - called by /init if needed.

• mkinitramfs
  - create file list for CONFIG_INITRAMFS_SOURCE
  - kernel build creates a kernel with kboot initramfs compiled in.
xen mini kboot -- usage

- reads arguments from the kernel command line
- root=
  - root filesystem
- boot=
  - /boot filesystem (optional)
- kernel=
  - kernel image (optional). If not specified kboot will search in /boot for kernels. If multiple are present it builds a menu.
- ramdisk=
  - ramdisk (optional). If not specified kboot will look for one matching the kernel name.
xen mini kboot – features and limits

• Features
  - can boot from raid.
  - can boot from lvm.

• Limits
  - no config file from the guest disk is read.
xen mini kboot – current state

• Changes and fixes for xen and backend drivers have been merged into unstable recently (11404+).
  - Any xen-3.0.3 host system should be able to handle guest domains doing kexec, so kboot should work too.

• kexec has limits
  - domains with lots of memory likely cause problems.
  - kexec'ing non-linux kernels is completely untested.
  - In theory anything the domain builder can boot should be bootable via kexec too.

• try yourself!
  - http://www.suse.de/~kraxel/kexec/
guest kexec: the kernel internals
kernel: how kexec works

- At load time:
  - copy segments into kernel memory
  - create a list with temporary and final place of the pages.
- At boot time:
  - disable interrupts, ...
  - identity-map trampoline page, jump to it.
  - turn of paging or activate identity mapping.
  - copy pages using the list created at load time, creating the final memory layout.
  - jump to entry point.
kernel: paravirtualized != native

- running in protected mode, with paging enabled all the time.
- special rules for page table access.
- different segment handling.
- magic pages (console, xenstore).
- state info to maintain
  - start_info page
  - phys_to_machine_mapping
kernel: passing xen state info

- old kernel, arch-specific code
  - pf 0x10: old start_info
  - pf 0x11: xenstore page
  - pf 0x12: console page
  - p2m map: temporary copy, ptr in old start_info
    - FIXME: temporary p2m map must be contigious memory right now.
- helper code (loaded by xc_kexec)
  - pf 0x20+: code and data
  - xc_kexec passes some info by filling variables (vcpu, ...)
  - helper code moves / copies magic pages & p2m map to the final location.