Fast Reboots with kload

- Russell Cattelan

  Kernel Engineer

  - cattelan@digitalelves.com
  - http://git.digitalelves.com/?p=FreeBSD_kload.git

Digital ELVES

digitalelves.com

Isilon Storage Systems a division of EMC

Sponsored by and funded

How does this help?

- Avoid reseting system
  - BIOS/ POST takes along time
    - Varies from system to system
- New kernel is loaded into memory before reboot
- Modeled on Linux's kexec / kdump

Building on userboot.so

- userboot.so developed for the BHyVe project
  - FreeBSD Hyper Visor.
- Separates the guts of /boot/loader into a library that is usable userspace utilities
  - Forth interpreter
  - elf loader
- Reads and sets up elf images in memory.
  - Primary kernel image
  - Kernel modules i.e. if_re.ko
- Temporary main memory image
- Populates kernel parameter page (kenv)

/sbin/kload

- Loads / configures / adjusts
  - Reads kernel / modules – sets up memory image
  - Builds smap by querying existing kernel via sysctl
    - Map of system memory
- Command Line options
  - -k flag add / override kenv parameters
  - -e execute **NOW** skip shutdown routines
  - -r sends kill signal to all processes – normal reboot
  - -h alternate / loads a kernel not installed in /<arg>/boot
- Initiates kload syscall once everything is ready to go

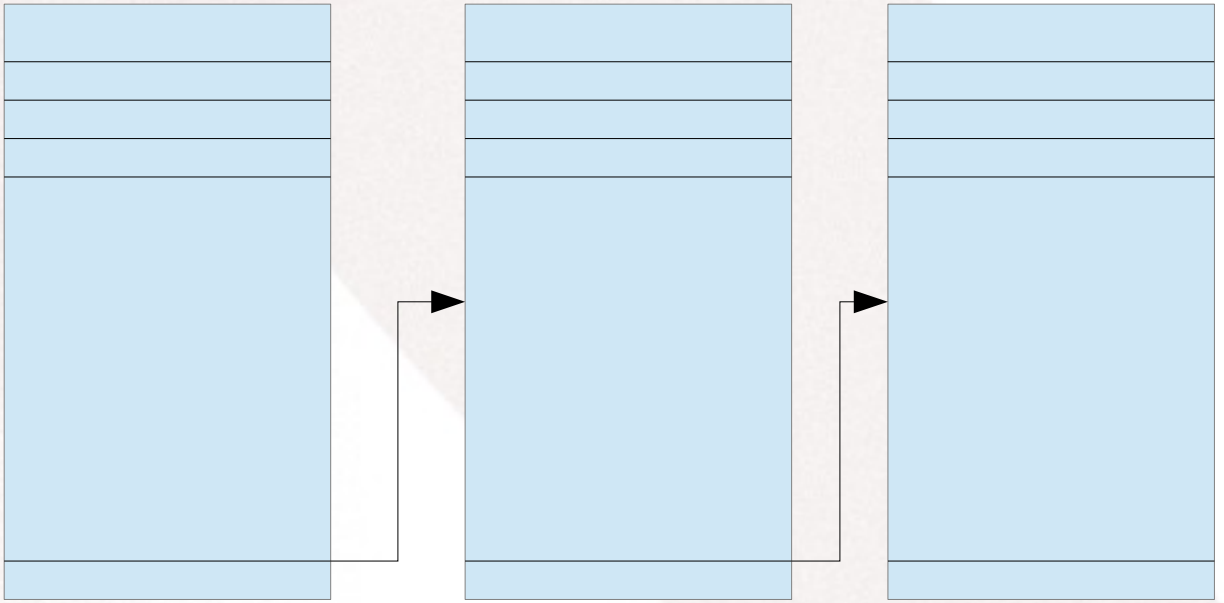
The running kernel is the loader

- userboot.so / kload does the image loading / setup but it's not in the right spot in memory.
  - First step is to allocate temporary pages in the lower 1 Gig of memory. (This is sometimes a performance issue)
  - Build a very simple scatter gather list of identity mapped pages -- physical address + KERNBASE
  - Setup simple GDT / pagetable
  - Allocate code / stack / control pages
  - Handle AP / interrupt shutdown

Scatter gather list of pages

Simple page list structure

- page_list – series of page address last address in each page being a ptr to next page list



System clean up / shutdown.

- Mostly the same as normal shutdown.
  - Hooks into shutdown chain right before last event when kernel is loaded
  - Last shutdown event is to reset cpu / power down

```
#define SHUTDOWN_PRI_FIRST      EVENTHANDLER_PRI_FIRST
#define SHUTDOWN_PRI_DEFAULT    EVENTHANDLER_PRI_ANY
#define SHUTDOWN_PRI_LAST       EVENTHANDLER_PRI_LAST
#define SHUTDOWN_PRI_KLOAD      EVENTHANDLER_PRI_LAST - 100
  /* hook into the shutdown/reboot path so we end up here before cpu reset */
  EVENTHANDLER_REGISTER(shutdown_final,
                kload_shutdown_final, NULL, SHUTDOWN_PRI_KLOAD);
```

kload_final

- Send Inter Processor Interrupt IPI to cpu 1 – X Application Processors (APs) telling them to suspend
  - Mask lapic on each cpu especially timer interrupts
    - This code needs to be reconciled with suspend / resume
  - Not ported to i386 yet
- De-install all ioapic interrupts (system wide interrupts routed to a particular lapic)
- Mask lapic (Local Advanced Programmable Interrupt Controller) on cpu 0 Board Support Processor (BSP) disable cpu interrupts
- At this point system is ready to replace kernel

Replace old kernel image

- relocate_kernel
  - Relocates itself and running stack so as to not clobber itself (long jump)
  - Turn off processor interrupts again :-)
  - Install simple GDT with a writeable code segment (CS) and writeable data segment (DS)
  - Install identity mapped page table – entire address space maps to first 1 Gig of memory
  - Walk list of pages copy over the existing kernel pages starting at KERNBASE
  - Push kernel start address on to stack long jump to it

```
        /* first install the new page table */
        movq 32(%rcx), %rax /* page table */
        movq 40(%rcx), %r9  /* address of control_page with new PT */
        movq %rax, %cr3

        movq $(X86_CR4_PSE | X86_CR4_PAE), %rax
        movq %rax, %cr4


        /* then move the stack to the end of control page */
        lea     4096(%r9), %rsp
        /*
         * now move to the code page
         * should have been passed code_page based
         * on new page table
         */
        movq %rdx, %r8
        addq  $(identity_mapped - relocate_kernel), %r8
        /* offset of code segment in new gdt */
        pushq$0x08
        pushq%r8
        /* jump to this spot in the new page */
        lretq
identity_mapped:
        /* Do the copies */
.
.
.
        pushq16(%r9)            /* physfree */
        movq 8(%r9), %rax       /* modulep */
        salq   $32, %rax
        pushq%rax


        pushq$0x8
        pushq48(%r9)            /* entry # kernel entry pt */
        /* jump to kernel entry pt */
        lretq
```

Lets try it out

Known Issues

- Drivers need to correctly shutdown the hardware
  - Realtek driver needs a reset added to re_shutdown
- kmem_alloc_attr sometimes takes a long time to return memory. Even to the point were it would probably be faster to do a normal boot
  - Memory is pre-allocated for now
- Debugging is very hard in asm code – needs to be done with bochs
- Does not have a kload --unload option – memory can not be released.

More gritty details

- Intel x86 emulator Bochs !!!!
  - This project would have happened without this tool as there is now way to debug things without hardware level instruction debugging / stepping
- GDT page table mirrors what /boot/loader sets up
  - These differ for amd64 and i386
  - PAE kernel not supported?
- Could do tricks with page tables to reduce the number of times the images needs to be copied

More gritty details

- Intel x86 emulator Bochs !!!!
  - This project would have happened without this tool as there is now way to debug things without hardware level instruction debugging / stepping
- GDT / page table same as /boot/loader
  - These differ for amd64 and i386
  - PAE kernel not supported?
- Could do tricks with page tables to reduce the number of times the images needs to be copied

GDT / IDT / TSS / Page Table

- Global Descriptor Table
  - Old way of loading multiple programs by segmenting memory. Not used by paging sytems but still need to setup a minimal GDT.
  - Interrupt Descriptor Table.
  - Create empty IDT table just to make sure, but interrupts should be disabled.
  - Task State Segment – not used during kload should already set to ring0 (full privileges)

  Create identity mapped page table

Identity mapped pagetable

- Page size set to 2 meg / page tables are built using 2 meg pages
- Amd64 / PAE used level 3 page tables
  - 512 64 bit / 8byte address per 4k page for a total of 1Gig of memory per page
- I386 uses level 2
  - 1024 32 bit / 4 byte address per 4k page
- Used the same code as the loader to set up page tables as to keep things as simple as possible

Page table code

- Map all of the address space to the first 1GB

```
for (i = 0; i < 512; i++) {
  /* Each slot of the level 4 pages points to the same level 3 page */
  PT4[i] = (pt_entry_t)(vtophys(PT3));
  PT4[i] |= PG_V | PG_RW | PG_U;

  /* Each slot of the level 3 pages points to the same level 2 page */
  PT3[i] = (pt_entry_t)(vtophys(PT2));
  PT3[i] |= PG_V | PG_RW | PG_U;

  /* The level 2 page slots are mapped with 2MB pages for 1GB. */
  PT2[i] = i * (2 * 1024 * 1024);
  PT2[i] |= PG_V | PG_RW | PG_PS | PG_U;
}
```

a/sys/amd64/amd64/intr_machdep.c
a/sys/amd64/amd64/kload.c
a/sys/amd64/amd64/kload_exec.S
a/sys/amd64/amd64/machdep.c
a/sys/amd64/amd64/mp_machdep.c
a/sys/amd64/conf/KLOAD
a/sys/amd64/conf/KLOAD-CAMDEBUG
a/sys/amd64/include/apicvar.h
a/sys/amd64/include/intr_machdep.h
a/sys/boot/Makefile
a/sys/boot/common/Makefile.inc
a/sys/boot/common/load_elf.c
a/sys/boot/ficl/Makefile
a/sys/boot/i386/libi386/amd64_tramp.S
a/sys/boot/userboot/Makefile
a/sys/boot/userboot/ficl/Makefile
a/sys/boot/userboot/test/Makefile
a/sys/boot/userboot/userboot.h
a/sys/boot/userboot/userboot/Makefile
a/sys/boot/userboot/userboot/bootinfo64.c
a/sys/boot/userboot/userboot/conf.c
a/sys/boot/userboot/userboot/main.c
a/sys/boot/userboot/userboot/userboot_cons.c
a/sys/conf/files
a/sys/conf/files.amd64
a/sys/conf/kern.pre.mk
a/sys/conf/options
a/sys/dev/re/if_re.c
a/sys/i386/i386/kload.c
a/sys/i386/i386/kload_exec.S
a/sys/kern/init_sysent.c
a/sys/kern/kern_kload.c
a/sys/kern/kern_module.c
a/sys/kern/syscalls.c
a/sys/kern/syscalls.master
a/sys/kern/systrace_args.c
a/sys/sys/eventhandler.h
a/sys/sys/kload.h
a/sys/sys/reboot.h
a/sys/sys/syscall.h
a/sys/sys/syscall.mk
a/sys/sys/sysproto.h
a/sys/x86/x86/local_apic.c
a/sys/x86/x86/nexus.c
a/usr.sbin/kload/Makefile
a/usr.sbin/kload/kload.c

- From osdevwiki.org each must follow this format
- http://wiki.osdev.org/GDT
  - Very hard to just decode

31 16	15 0
Base 0:15	Limit 0:15

63 56	55 52	51 48	47 40	39 32
Base 24:31	Flags	Limit 16:19	Access Byte	Base 16:23

```
void
setup_freebsd_gdt(uint64_t *gdtr)
{
  gdtr[GUEST_NULL_SEL] = 0x0000000000000000;
  gdtr[GUEST_CODE_SEL] = 0x0020980000000000;
  gdtr[GUEST_DATA_SEL] = 0x0000920000000000;
}
```

References

- http://wiki.freebsd.org/BHyVe
- http://wiki.osdev.org
- http://bochs.sourceforge.net
- http://bhyve.org