Overview of the x86-64 kernel

Andi Kleen, SUSE Labs, Novell
ak@suse.de

Linux Bangalore 2004

What’s wrong?

□x86-64, x86_64

□AMD64

□EM64T

□IA32e

□IA64

□x64, CT

Names

□x86-64, x86_64

□AMD64

□EM64T

□IA32e

□x64

□CT

Basics

- 64bit extended x86 architecture
- Can be used with 32bit OS too
  - But 64bit OS is better
- Originally from AMD
- Shipping by AMD and Intel
  - Servers and desktops and even laptops
- Announced by Transmeta and VIA

History of the Linux port

- SUSE Labs project
- Started on simulators in 2000
  - Fork from i386
- Was running on early silicon by AMD
- First betas in 2002
- Shipping product (SLES8) in 2003
- Merged into 2.4 in 2002

Long mode

- 64bit addressing support
- 64bit instructions
- 8 more integer and SSE2 registers
  - eax -> rax
  - r8-r15, xmm8-xmm15
- RIP relative addressing mode
  - Faster shared libraries
- Compat mode to run 32bit
  - Practically no performance penalty compared to 32bit OS

An oops

```
general protection fault: 0000 [1]
CPU 0
Modules linked in: ....
Pid: 7026, comm: insmod Tainted:
RIP: 0010:[<ffffffffa073a000>] <ffffffffa073a000>{:toops3:f2+0}
RSP: 0000:000001000fc79f40  EFLAGS: 00010216
RAX: ffffffffa073a010 RBX: ffffffff803c4da0 RCX: 0000000000101000
RDX: 0000000000000000 RSI: feedbabedeadbeef RDI: feedbabedeadbeef
RBP: ffffffffa073a500 R08: 00000100018af010 R09: 000001001ff6d560
R10: 000001001ff6d570 R11: 0000000000000000 R12: ffffffff803c4cc0
R13: ffffffff803c4cc0 R14: 000000000000000f R15: ffffffff8013cb00
FS:  0000002a9588f4c0(0000) GS:ffffffff804c6480(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 000000000051b000 CR3: 0000000000101000 CR4: 00000000000006e0
Process insmod (pid: 7026, threadinfo 000001000fc78000, task 000001001d7610b0)
Stack: ffffffffa073a019 000001001d7610b0 ffffffff80110e47 ffffffff8013cb00
       000000000000000f ffffffff803c4cc0 ffffffff803c4cc0 ffffffffa073a500
       ffffffff803c4da0 0000000000000000
Call Trace:<ffffffffa073a019>{:toops3:crash+9} <ffffffff80110e47>{child_rip+8}
       <ffffffff8013cb00>{msleep+0} <ffffffffa073a010>{:toops3:crash+0}
       <ffffffff80110e3f>{child_rip+0}


Code: c6 07 01 c3 66 66 66 90 66 66 66 90 66 66 66 90 48 83 ec 08
RIP <ffffffffa073a000>{:toops3:f2+0} RSP <000001000fc79f40>
 done

   0:   c6 07 01                movb   $0x1,(%rdi)
   3:   c3                      retq
```

Some myths

- 64bit is bigger
  - Depends on what CPU you optimize for
  - Normally <~10% difference
  - Sometimes code is even smaller

- 64bit is slower
  - Additional registers
  - New modern ABI
  - SSE2

- I don’t need 64bit, I have less than 4GB of RAM
  - 32bit limit in practice around 2GB
  - Virtual address space fragments (e.g. thread stacks)
  - IO memory hole needs physical space below 4GB

Basics

- Started as a copy of arch/i386, include/asm-i386
- Low level assembly code rewritten
- Code heavily changed for 64bit
  - And only support modern chipsets
- Lots of old cruft removed
  - Workarounds for old hardware bugs
  - No DMI checks so far
  - No APM, no vm86, ...
- Some code shared: MTRR, cpufreq, swiotlb, ...

New features

- NUMA
  - Based on generic NUMA infrastructure in VM
  - Originally for Opteron only, now also supports ACPI SRAT
  - NUMA API
- 32bit emulation
  - Based on code from other 64bit ports
- IOMMU
- 4level page tables
  - Before that 512GB limit per process
- Redesigned machine check handling

Current state

- □Widely used
- □2.4 in maintenance mode
- □2.6 production and development

Porting: basics

- Code must be 64bit clean
- long is 64bit now, int stays 32bit
- Pointers in long, not int
  - different from WIN64
- -Wall cleanness is a good start

Porting in userspace: /lib64

- All 64bit libraries are in lib64
  - 32bit stays in lib
  - Special compat packages for old libraries
- Makefiles often need to be fixed
  - configure --enable-lib-suffix=64
- Not perfect: no include64, bin64
  - Best to have separate library RPMS
  - RPM versions should match

Porting: IOMMU basics

- Some devices cannot address all memory
  - Kick your hardware people if it happens with new hardware

- Driver must map buffers before passing them to hardware
  - Replaces __va, virt_to_bus
  - And free them of course
  - Should be used always

- Explicit cache flushing

- Only works for devices with at least 4GB address space
  - Smaller ones need pci_alloc_consistent()

Porting: IOMMU implementation on x86-64

- AMD AGP GART IOMMU
  - Not a real IOMMU...
  - Uses AGP GART functionality in the CPU northbridge
  - Reuses half of the AGP aperture by default
  - Size depends on BIOS or can be mapped over memory
- Slower swiotlb on Intel
  - And some buggy AMD chipsets
  - Does memory copies
  - Slow
- Remap space is limited
  - Sometimes only 64MB
  - Can be tuned with kernel command line options and in BIOS
  - Best to limit yourself and handle overflows

Porting: IOMMU functions

- pci_set_dma_mask
- pci_alloc_consistent for IO memory
  - pci_free_consistent
- pci_map_sg/pci_map_single for dynamic mappings
  - Need 4GB dma mask or better
- pci_dma_sync_{single,sg}_for_{device,cpu}

Porting: IOMMU notes

- Check and handle errors
  - Especially in block drivers!
  - pci_map_sg returns 0 on error
  - pci_dma_mapping_error for pci_map_single

- dma_* can be used too for generic bus support
  - pci_alloc_consistent -> dma_alloc_coherent
  - pci_map_single -> dma_map_single
  - pci_map_sg -> dma_map_sg
  - pci_dma_mapping_error -> dma_mapping_error

- Documentation/DMA-mapping.txt

Porting: 32bit emulation basics

- 32bit has separate libraries in user space
- 32bit and 64bit always run in different processes
- Kernel has a 32bit emulation layer
- Kernel converts all system calls
  - {fs,net,kernel}/compat.c
- ioctls in drivers need special conversion
- Avoid message passing over read/write

Porting: 32bit ioctl handler

- Needed for x86_64, ppc64, s390x, ia64, mips64, parisc64

- Kernel does it centrally for most of its own ioctls
  - fs/compat-ioctl.{c,h}

- Drivers can register own ioctl handler
  - register_ioctl32_conversion

- Passed through if compatible or converted

- Conversion of structures on user stack
  - Converted from 64bit to compat_* types
  - Access using normal *_user functions

What needs conversion?

- long
- pointers
- long long / u64 without natural alignment
  - Different from RISC ports!
- Some fundamental types
  - dev_t, inode_t, time_t, ...

ioctl conversion functions

- #include <linux/compat.h>
- register_ioctl32_conversion()
  - Need unique number
  - Use _IO* macros to define ioctls
- copy_in_user()
- compat_alloc_userspace()
- sys_ioctl()
- compat_ptr()

32bit conversion example

```
#include <linux/compat.h>

struct ppp_idle32 {
        compat_time_t xmit_idle;
        compat_time_t recv_idle;
};

#define PPPIOCGIDLE32           _IOR(’t’, 63, struct ppp_idle32)

static int ppp_gidle(unsigned int fd, unsigned int cmd, unsigned long arg)
{
        struct ppp_idle __user *idle;
        struct ppp_idle32 __user *idle32;
        __kernel_time_t xmit, recv;
        int err;

        idle = compat_alloc_user_space(sizeof(*idle));
        idle32 = compat_ptr(arg);

        err = sys_ioctl(fd, PPPIOCGIDLE, (unsigned long) idle);

        if (!err) {
                if (get_user(xmit, &idle->xmit_idle) ||
                    get_user(recv, &idle->recv_idle) ||
                    put_user(xmit, &idle32->xmit_idle) ||
                    put_user(recv, &idle32->recv_idle))
                        err = -EFAULT;
        }
        return err;
}
```

References

□/usr/src/linux/arch/x86_64, include/asm-x86_64/

□http://www.x86-64.org

□/usr/src/linux/Documentation/DMA-mapping.txt

□discuss@x86-64.org

□Questions?

Backups

Porting issues: 32bit code with 64bit apps

□ Direct linking not possible

□ All conversion is in the kernel

□ Recommended method: several processes, RPC

□ Make sure your RPC encoding doesn’t assume wordsize

□ Example: Konqueror using 32bit plugins with DCOP