Linux x86-64 port

Andi Kleen, SuSE Labs
ak@suse.de

Overview of X86-64 I

- X86 / SSE2 based
- Long mode / Compatibility mode / Legacy mode
- 8 additional integer registers (R8-15)
- 8 additional SSE2 register (XMM8-15)
- 64bit registers with zero extension

Overview of X86-64 II

- RIP relative memory access.
- 43bit address space / 48bit in architecture
- Stack always 64bit aligned.

Overview of X86-64 III - System

- Segment bases and limits are ignored -> Segmentation gone.
- FS/GS stay as a kind of address register
- Interrupt stack / interrupt priority support
- 4 level page tables similar to PAE

Overview of X86-64 IV - Things dropped

- 16bit segments are gone (support for 16bit programs in wine gone)
- Task switching dropped
- vm86 dropped (dosemu gone in long mode)

Overview of X86-64 V - Instructions gone.

These are all single byte instructions.

Ascii Adjust: AAl, AAD, AAM, AAS

BCD Adjust: DAA, DAS

Rarely used bounds checking: INTO, BOUND

LAHF, SAHF, SALC

PUSHA, POPA

PUSH/POP segment register (multibyte equivalents still exists)

Segment cruft: LDS, LES, JMPF immediate, CALLF immediate, ARPL

New ABI I

- Modern ABI optimized for code size
- Code size comparable to 32bit code.
- register arguments, including stdargs
- Natural alignment everywhere.
- Uses SSE2 registers fully

New ABI II

- Most registers are callee saved to save code space.
- Requires prototypes for floating point
- Non prototyped calls are a bit slower because they must handle stdargs
- double is always 64bit, only long double uses the x87 FPU stack.

New ABI III

- Stack is always 64bit aligned
- Stack redzone
- No frame pointer; uses unwind tables instead
- dwarf native debugging format

Code models

- Pointers are always 64bit, this just changes how addresses of linked objects are loaded in the code.

- Small

Code/static data limited to 2GB range, references in code RIP relative. Smallest and fastest code. Should be used by most programs.

- Medium

Code limited to 2GB, data references full 64 bit.

- Large

Support full 64bit data/code references. Bigger and slower code.

- Kernel

Negative small model. Exploits wrapping and sign extension in EA calculation for efficient kernel code.

- GCC & binutils

- x86-64 backend based on i386 backend
- SSE2 support implemented
- i386/x86_64 is merged (-m32 and -m64 work both from the same executable)
- Is stable enough for development
- gcc merged in gcc 3.1; binutils into official binutils tree.

Kernel: New port:

- Based on the i386 port.
- Ambitional port: trying to exploit new features instead of just trying to get it running.
- Started in August 2000

Kernel: Things removed

- Gone: support for old CPUs
- Gone: APM
- Gone: Lots of old bug workarounds (like F00F)
- Gone: FPU emulation
- Gone: support for non PAE

4K pages

- x86-64 has 4K pages.
- Linux allocator cannot reliable get more than two continuous pages.
- page table allocation failure is fatal.
- 3 level pagetable with 1 page each -> 39 bits.
- 8K kernel stacks -> interrupt stacks

Memory management

- Uses similar structures as modern x86 (3 level PAE), with minor changes.
- Only 3 level of 4 pagetables used by Linux ATM (= 39 bits/process),
- fourth level hidden from generic code.
- Kernel space negative
- User mode positive
- Kernel code mapped to upper part of negative space, for kernel code model.

Processor Data Area (PDA) I

- Every CPU has an per processor area
- It is always pointed to by %gs when the kernel runs.
- Needed for syscall and for interrupt stacks.
- Saves memory because padding is minimized.

Processor Data Area (PDA) II

- PDA cheaper to access than CPU number indexed arrays.
- Work still needed to put generic data structures into the PDA also.
- Hopefully other architectures will follow.

Split stackframe I

- System call entry is very critical
- Saves only callee clobbered integer registers on normal syscall or interrupt.
- Program pointer/stack pointer/etc. are saved into PDA
- Signals/exec/fork/clone/ptrace save full stack frame with special stubs.

Split stackframe II

- Exceptions save full frame.
- stack frame on most system calls is valid, but many fields are undefined(including rip)
- Interrupts see interrupt frame and all non callee saved in ptregs arg.
- Not clear if it’s really worth it.

Interrupt stacks I

- Stack limit of two pages (8K) due to VM limitations.
- 64bit code needs more stack than 32bit.
- Uses interrupt stacks to stay in limit.

Interrupt stacks II

- Interrupt stacks implemented in software as the hardware mechanism doesn’t support nested interrupts easily.
- Getting the current process via stack pointer does not work anymore.
- Uses the PDA for that instead.
- Allows to use cache colouring allocation for task_struct to get better cache usage in the scheduler.

vsyscalls

- gettimeofday is a very critical system call
- It can be implemented in user context with some kernel support using the CPU timestamp counters.
- vsyscalls map code into user space at a fixed address
- Can be called with the overhead of a system call.
- Problems with exception handling: needs an unwind table that has to be supplied by the user

Context switch

- Has to save more registers and they are twice as big (->slower)
- Manages 64bit segment registers lazily, because rdmsr/wrmsr is slow.
- Lazy FPU context switch.
- More efficient kernel entry saves some overhead again.

IA32 emulation

- Translates system calls and ioctl that pass data structures with pointers or long.
- Based on previous sparc64/ia64 code.
- Sits as an layer between the 32bit syscall entry (int 0x80) and the normal kernel calls.

IA32 emulation details

- Shares the same stack frame with 64bit calls
- 32bit Syscall instruction not supported.
- A lot of unix system calls can be directly mapped with zero extension.
- System calls that need sign extended arguments (e.g. lseek) need to be mapped.

IA32 emulation split

- Currently rather monolithic.
- Most of it portable code and needed by at least 6 architectures.
- Plan to make it generic for 2.5 and move it into subsystems.
- Drivers should translate their own ioctls with register_ioctl

Status

- Kernel works for 32bit and 64bit executables.
- Stable enough for user space development
- Currently based on2.4.7.

People

Kernel: Andrea Arcangelli, Pavel Machek, Andi Kleen, Karsten Keil

Glibc: Andreas Jaeger

Gcc/Binutils: Jan Hubicka, Bo Thorsen

GDB: Jiri Smid

XFree86: Egbert Eich

URLs

http://www.x86-64.org

Kernel patches at ftp://ftp.x86-64.org/pub/linux-x86_64/v2.4/

Getting things via CVS:

cvs -d :pserver:anoncvs@cvs.x86-64.org:/cvs/Repository login

Password: anoncvs

cvs -z4 checkout <module>

Some module: linux, gcc, binutils, x86-64-ABI, ...