

Lecture 8: March 27

*Lecturer: Alessandro Pellegrini**Scribe: Anzhelo Xhebraj*

Why is `__syscall_return` needed? A syscall can either fail or succeed. To let the user know what happened that macro sets `errno`. Signed arithmetic is more costly than unsigned therefore making the check unsigned the compiler will generate unsigned arithmetic instructions making the check faster.

Only 8 registers are available: ESP cannot be used, EAX is used for the code of the syscall, EBP must be saved before used.

The dispatcher takes a complete snapshot of CPU registers for two reasons: the dispatcher will mangle the content of the registers so it has to save them; when the trap is issued the calling convention does not say anything about what registers could be modified by the kernel.

The registers are pushed on stack and then a call to the syscall is performed. If the snapshot is modified then a different context will be given back to the user process.

Resizing the syscall table (static variable vector) requires to reshuffle the whole compilation process since Bootmem must have a new bitmap etc.

`sysenter/sysexit` are fast system call paths since by using `int $0x80` too much overhead was needed (enter IDT, GDT, change TR etc).

stdlib introduced `syscall()` interface but then later the instructions above were introduced in the kernel.

Dispatcher

Model specific register speedup the entering of kernel mode since they store the informations needed to start executing kernel code. Doesn't do any check.

Virtual Dynamic Shared Object: implements assembly code to activate/issue the

8.1 Multi cores synchronisation issues

Changes in some data structures in the system we must ensure that all cores are aligned to same view of data structures and resources. Some issues are addressed by firmware, other not.

Requests other cores to perform some action. Firmware generates them but software processes them.

INTER PROCESSOR INTERRUPTS (IPI)

References

- [a20()] A20 line. URL https://wiki.osdev.org/A20_Line.
- [car()] Writing a bootloader from scratch. URL <https://www.cs.cmu.edu/~410-s07/p4/p4-boot.pdf>.
- [con()] Context switching. URL https://wiki.osdev.org/Context_Switching.
- [des(a)] Descriptor cache, a. URL https://wiki.osdev.org/Descriptor_Cache.
- [des(b)] Interrupt descriptor table, b. URL https://wiki.osdev.org/Interrupt_Descriptor_Table.
- [osd()] "8042" ps/2 controller. URL https://wiki.osdev.org/8042_PS2_Controller.
- [rma()] The workings of: x86-16/32 realmode addressing. URL https://web.archive.org/web/20130609073242/http://www.osdever.net/tutorials/rm_addressing.php?the_id=50.
- [ser()] Interrupt service routines. URL https://wiki.osdev.org/Interrupt_Service_Routines.
- [tas()] Task state segment. URL https://wiki.osdev.org/Task_State_Segment.
- [vec()] Interrupt vector table. URL https://wiki.osdev.org/Interrupt_Vector_Table.
- [wra()] Who needs the address wraparound, anyway? URL <http://www.os2museum.com/wp/who-needs-the-address-wraparound-anyway/>.
- [x86()] Why doesn't linux use the hardware context switch via the tss? URL <https://stackoverflow.com/questions/2711044/why-doesnt-linux-use-the-hardware-context-switch-via-the-tss>.
- [wik(2017)] Task state segment, Jun 2017. URL https://en.wikipedia.org/wiki/Task_state_segment.
- [wik(2018a)] x86 memory segmentation, Mar 2018a. URL https://en.wikipedia.org/wiki/X86_memory_segmentation#cite_note-Arch-1.
- [wik(2018b)] A20 line, Feb 2018b. URL https://en.wikipedia.org/wiki/A20_line.
- [Bovet and Cesati(2006)] Daniel P. Bovet and Marco Cesati. *Understanding the Linux kernel*. O'Reilly, 2006.
- [Brouwer()] Andries E. Brouwer. A20 - a pain from the past. URL <https://www.win.tue.nl/~aeb/linux/kbd/A20.html>.
- [Collins(a)] Robert Collins. A20 - reset anomalies, a. URL <http://www.rcollins.org/Productivity/A20Reset.html>.
- [Collins(b)] Robert Collins. The segment descriptor cache, b. URL <http://www.rcollins.org/ddj/Aug98/Aug98.html>.
- [Duarte(2008a)] Gustavo Duarte. Memory translation and segmentation, Aug 2008a. URL <https://manybutfinite.com/post/memory-translation-and-segmentation/>.
- [Duarte(2008b)] Gustavo Duarte. Cpu rings, privilege, and protection, Nov 2008b. URL <https://manybutfinite.com/post/cpu-rings-privilege-and-protection/>.

[Intel()] Intel. *Intel 64 and IA-32 Architecture Software Developer's Manual*, volume 3A.

[Oostenrijk(2016)] Alexander van Oostenrijk. Writing your own toy operating system: Jumping to protected mode, Sep 2016. URL <http://www.independent-software.com/writing-your-own-toy-operating-system-jumping-to-protected-mode/>.