AMD64 mode switching

Konstantin Belousov kib@FreeBSD.org

June 11, 2021, git date: 2021-06-13

Context Switching

Inter-Thread

- Done only in kernel mode
- Only subset of the whole CPU state is switched

Inter-Protection Domains

- ***User*** $\rightarrow$ ***Kernel***
- ***Kernel*** $\rightarrow$ ***User***

Other

- SMI

When: User->Kernel switch reasons

Synchronous

- syscalls
- exceptions

Asynchronous

- exceptions
- interrupts
- Non-maskable interrupts (NMI, MCE, delayed DB)

What: What is handled on kernel entry

Switched

- General Purpose Registers
- %RFLAGS.DF and %RFLAFS.AC
- TLS base, complicated by WRGSBASE
- Stack
- %CR3 for PTI case

Not switched (by FreeBSD)

- FPU registers (whole XSAVE management), handled on thread context switch
- segment registers, handled on return to usermode

Kernel TLS

- Still based on segmentation, but AMD64 makes it much less functional comparing with IA32
- Segment register %gs based, GSBASE MSR
- Shared with userspace, needs switching
- Before we know anything about kernel context
- Hack: SWAPGS: exchanges GSBASE and KGSBASE
- Critical to only do SWAPGS when needed
- user GSBASE is not neccessary what is set by the SYSARCH syscall, if WRGSBASE is supported. Done to support usermode context switching

Other non-GPR parts

Stack switching

- syscalls and synchronous exceptions, interrupts: RSP0 or reuse
- asynchronous exceptions and non-maskable interrupts: IST

PTI: %CR3

- follows stack logic
- but needs space/stack to not corrupt interrupted context

Invisible CPU state

- NMI disable state
- DB disable (single-instruction)

Problems

- Switching is not atomic against NMI and faults
- Exception and non-maskable interrupt handlers must understand what was interrupted (e.g. avoid unneeded SWAPGS)
- Reuse of stack means that we might nest on very limited PTI stack
- Nested exceptions which use IST corrupt stack
- Not all state is visible: NMI blocking, pending DB, STI interrupt shadowing
- Interrupt shadowing means that DB is not maskable

Return to userspace

Restore user segment registers

- May fault

Stack, %CR3

- Are switched back to userspace

IRETQ

- May fault
- Fault CPU state: %CR3, stack, TLS all userspace, but fault from kernel
- Different between Intel, AMD, and most emulators

What we need

- Atomicity of context switch
- With more elements added to context definition, like GS.base, stack handling
- No interrupt shadowing
- Explicit management for hidden arch state, like NMI blocking
- Would be nice to not make it yet another level of hacks

Problems recognized

Intel

- Flexible Return and Event Delivery (FRED)
- Seems to be available for simics

AMD

- Supervisor Entry Extensions
- much less fleshed than Intel proposal (e.g. no MSR numbers allocated)

FRED: Intel handles AMD mistakes

Removed

- IDT
- FAR CALL, FAR RET, IRETQ
- SWAPGS

Added

- FRED Event Delivery: SYSCALL, INT$\star$, exceptions, and interrupts
- LKGS formally not FRED

FRED

Context

- %CS
- %RIP
- %SS
- %RSP
- %RFLAGS
- %GS.BASE
- stack level (AKA replacement for IST)

Event Delivery

- Whole context is switched atomically
- Stack: stack level selects
- Two events handler entry points: User and Kernel

FRED

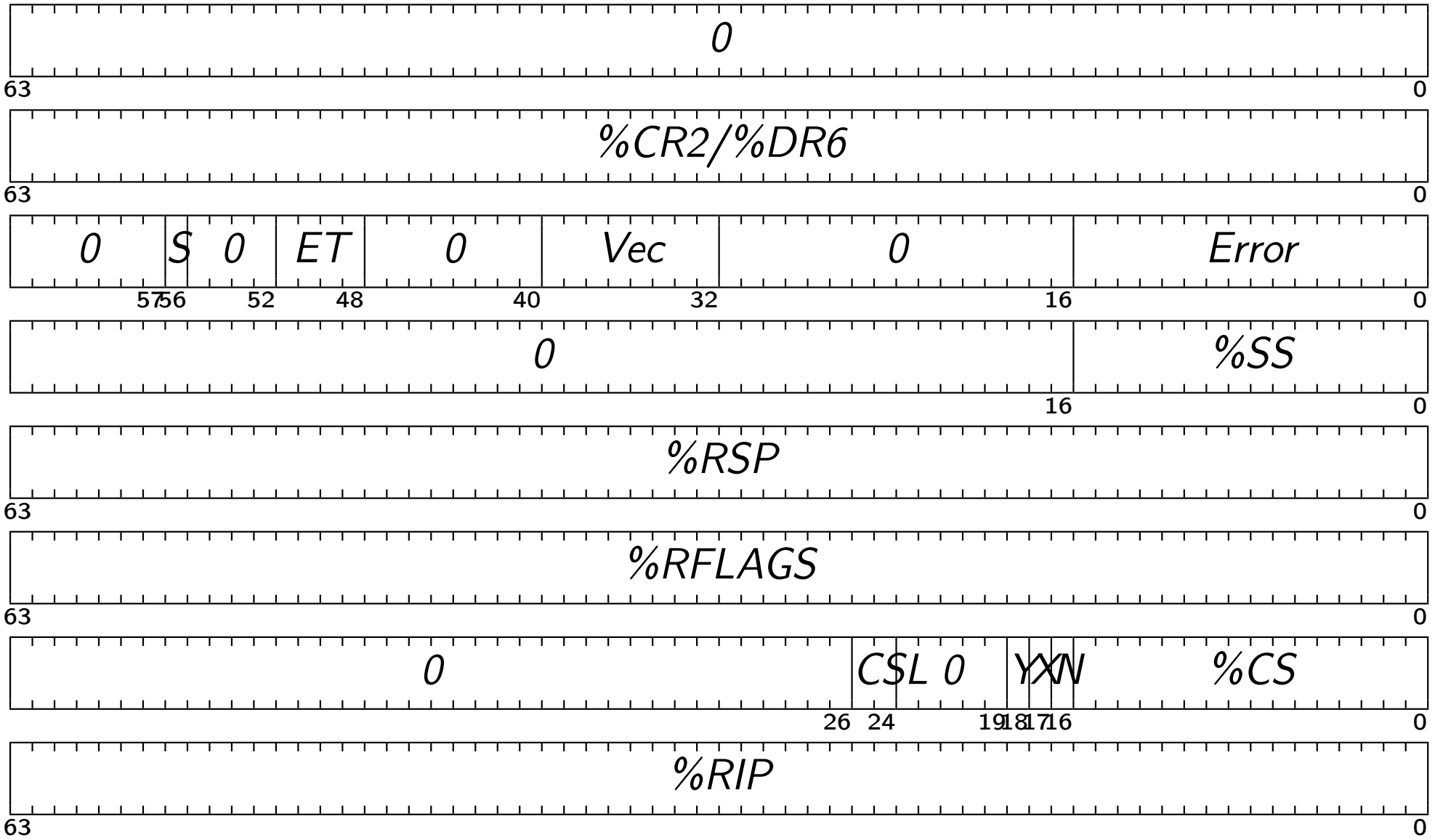
RSP

- Four stack levels, Current Stack Level CSL
- Stack level is configurable for event (maskable interrupts and 32 exceptions), otherwise it is 0
- Event from usermode: CSL = -1 effectively
- New stack level = Max(CSL, event stack level)
- %RSP = CSL changed ? Top of new stack : %RSP - GAP
- (GAP is configurable)

Stack Level

- Select by allowed nesting and requirement of having good stack
- Not a mistake to handle lower event on higher level stack

FRED saved Context



Legend

- S Enclave/SGX
- ET Event Type (same as VMX)
- N NMI
- X SYSCALL SYSENTER INT
- Y STI blocking AKA Interrupt shadow

Fred: return from event

New Instructions

- ERETS return to supervisor
- ERETU return to user, can fault in kernel ctx like IRETQ

NMI

%CS[16]

AMD: Supervisor Entry Extension

(this is my interpretation of an incomplete spec)

Kernel Entry Changes

- GS.base is managed by SYSENTER
- No interrupt shadowing

IDT Re-EntrantProtection

- Busy bit in the IDT entry
- Second exception of the same kind translated to double-fault
- How it would work? Perhaps trap handler must clear the busy bit before enabling preemption

Glossary

- DB Debugger Exception
- GPR General Purpose Registers
- IDT Interrupts Descriptors Table
- IRETQ Return from Interrupt instruction
- MCE Machine Check Exception
- MSR Model Specific Register, often not model-specific
- NMI Non-Maskable Interrupt
- PTI Page Table Isolation, a technique to mitigate Meltdown
- SGX Software Guard Extensions AKA Enclaves
- SMI System Management Interrupt
- SMM System Management Mode

Glossary: continued

- STI Enable Interrupts instruction
- TLS Thread Local Storage
- VMX Virtual Machine Extensions
- %gs segment register, base for TLS segment
- %cs code segment register
- %CR0 control register, machine control, in combination with %CR4 and %EFER registers
- %CR2 control register, address of the last page fault
- %CR3 control register, physical address of the page tables root
- %rflags CPU state flags registers
- %rip instructions pointer
- %rsp stack pointer
- %ss stack segment register

References

- Intel 64 and IA-32 Architectures Software Developer Manuals, Volume 3
- Intel, 346446, Flexible Return and Event Delivery (FRED), rev. 1.0, March 2021
- AMD, AMD64 Architecture Programmer's Manual Volume 2: System Programming
- AMD, 57115, AMD Supervisor Entry Extensions, rev. 0.50, February 2021