Kernel Entries

This file documents some of the kernel entries in arch/x86/entry/entry_64.S. A lot of this explanation is adapted from an email from Ingo Molnar:

https://lore.kernel.org/r/20110529191055.GC9835%40elte.hu

The x86 architecture has quite a few different ways to jump into kernel code. Most of these entry points are registered in arch/x86/kernel/traps.c and implemented in arch/x86/entry/entry_64.S for 64-bit, arch/x86/entry/entry_32.S for 32-bit and finally arch/x86/entry/entry_64_compat.S which implements the 32-bit compatibility syscall entry points and thus provides for 32-bit processes the ability to execute syscalls when running on 64-bit kernels.

The IDT vector assignments are listed in arch/x86/include/asm/irq_vectors.h.

Some of these entries are:

- system_call: syscall instruction from 64-bit code.
- entry_INT80_compat: int 0x80 from 32-bit or 64-bit code; compat syscall either way.
- entry_INT80_compat, ia32_sysenter: syscall and sysenter from 32-bit code
- interrupt: An array of entries. Every IDT vector that doesn't explicitly point somewhere else gets set to the corresponding value in interrupts. These point to a whole array of magically-generated functions that make their way to do_IRQ with the interrupt number as a parameter.
- APIC interrupts: Various special-purpose interrupts for things like TLB shootdown.
- Architecturally-defined exceptions like divide_error.

There are a few complexities here. The different x86-64 entries have different calling conventions. The syscall and sysenter instructions have their own peculiar calling conventions. Some of the IDT entries push an error code onto the stack; others don't. IDT entries using the IST alternative stack mechanism need their own magic to get the stack frames right. (You can find some documentation in the AMD APM, Volume 2, Chapter 8 and the Intel SDM, Volume 3, Chapter 6.)

Dealing with the swapgs instruction is especially tricky. Swapgs toggles whether gs is the kernel gs or the user gs. The swapgs instruction is rather fragile: it must nest perfectly and only in single depth, it should only be used if entering from user mode to kernel mode and then when returning to user-space, and precisely so. If we mess that up even slightly, we crash.

So when we have a secondary entry, already in kernel mode, we *must not* use SWAPGS blindly - nor must we forget doing a SWAPGS when it's not switched/swapped yet.

Now, there's a secondary complication: there's a cheap way to test which mode the CPU is in and an expensive way.

The cheap way is to pick this info off the entry frame on the kernel stack, from the CS of the ptregs area of the kernel stack:

```
xorl %ebx,%ebx
testl $3,CS+8(%rsp)
je error_kernelspace
SWAPGS
```

The expensive (paranoid) way is to read back the MSR_GS_BASE value (which is what SWAPGS modifies):

```
      movl $1,%ebx
      movl $MSR_GS_BASE,%ecx
      rdmsr
      testl %edx,%edx
      js 1f   /* negative -> in kernel */
      SWAPGS
      xorl %ebx,%ebx
1:    ret
```

If we are at an interrupt or user-trap/gate-alike boundary then we can use the faster check: the stack will be a reliable indicator of whether SWAPGS was already done: if we see that we are a secondary entry interrupting kernel mode execution, then we know that the GS base has already been switched. If it says that we interrupted user-space execution then we must do the SWAPGS.

But if we are in an NMI/MCE/DEBUG/whatever super-atomic entry context, which might have triggered right after a normal entry wrote CS to the stack but before we executed SWAPGS, then the only safe way to check for GS is the slower method: the RDMSR.

Therefore, super-atomic entries (except NMI, which is handled separately) must use idtentry with paranoid=1 to handle gsbase correctly. This triggers three main behavior changes:

- Interrupt entry will use the slower gsbase check.
- Interrupt entry from user mode will switch off the IST stack.
- Interrupt exit to kernel mode will not attempt to reschedule.

We try to only use IST entries and the paranoid entry code for vectors that absolutely need the more expensive check for the GS base - and we generate all 'normal' entry points with the regular (faster) paranoid=0 variant.