CS5250 Advanced Operating Systems

Pop Quiz 6

(Due: 1 Mar 2022, 11:59pm)

Name: _____Ding Fan_____

Student Number: _____A0248373X_____

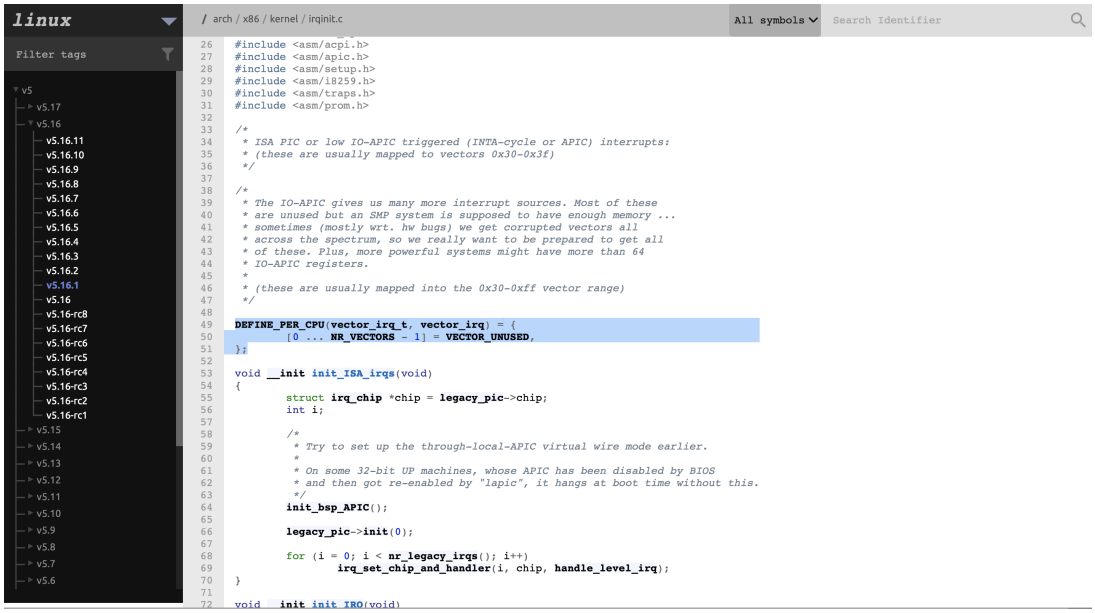
Please do a code walkthrough of the Linux 5.16.1 kernel and try to explain how the table vector_irq[] is initialized and used in interrupt processing. (Basically expand on my description in class.)

How the table vector_irq[] is initialized

1. In the file `init/main.c` we can see the call of the `init_IRQ()` function, which is called after the `early_irq_init()`

2. This function `init_IRQ()` is defined in the `arch/x86/kernel/irqinit.c`

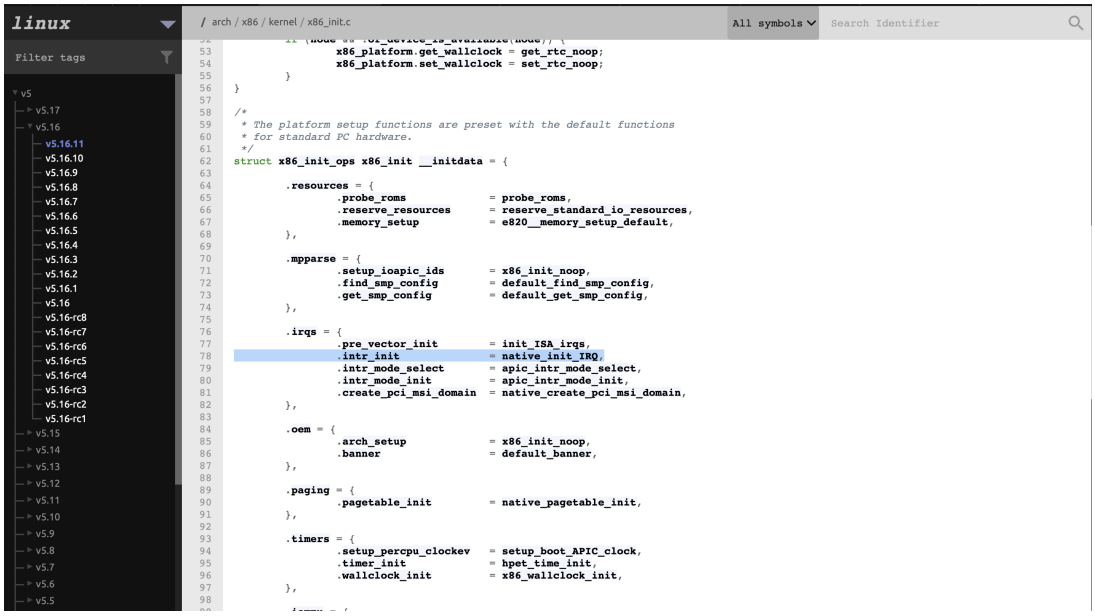
- The `init_IRQ` function makes initialization of the `vector_irq` percpu variable
- `vector_irq` is defined in the same `arch/x86/kernel/irqinit.c` source code file:
  - it represents the represents percpu array of the interrupt vector numbers.



- In conclusion in the start of the `init_IRQ()` function it will fill the `vector_irq` percpu array with the vector number of the `legacy` interrupts:

3. In the end of the `init_IRQ` function we can see the call of the following function: `x86_init.irqs.intr_init();`

- from `/arch/x86/kernel/x86_init.c` source code file. we find `x86_init.irqs.intr_init()` is point to the function `native_init_IRQ()` function in the `arch/x86/kernel/irqinit.c` source code file.



4. `native_init_IRQ()` function in the same `arch/x86/kernel/irqinit.c` source code file:

- in this function `native_init_IRQ()`, it will make initialization of the intel reserved interrupt vectors

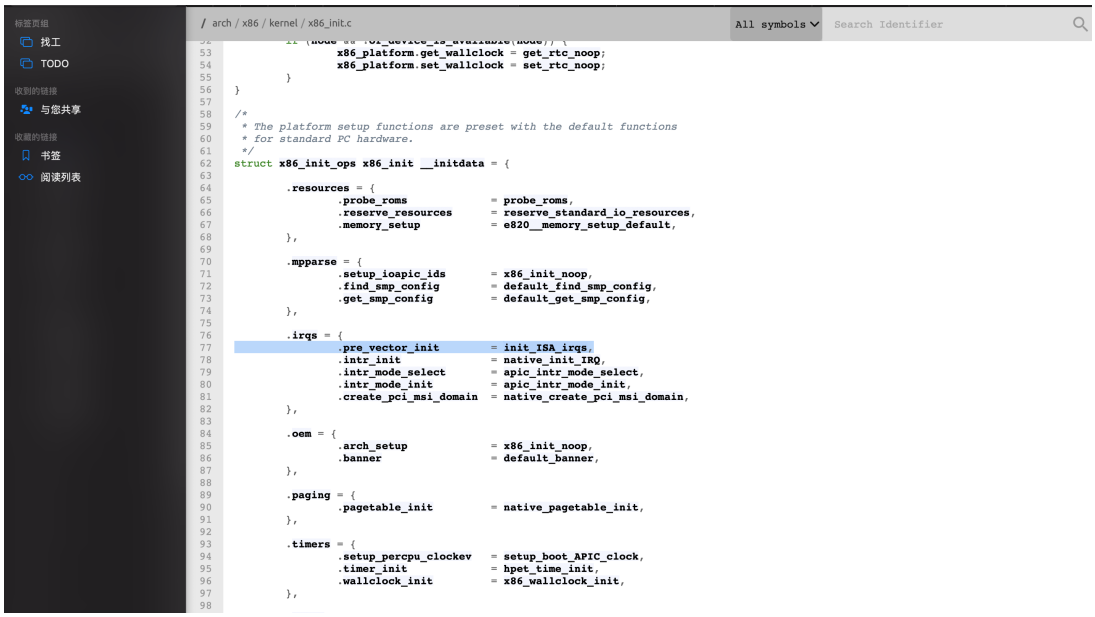
Intel Reserved Interrupts

Vector	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2		NMI Interrupt	Non-maskable external interrupt.
3	#BP	Breakpoint	INT 3 instruction.
4	#OF	Overflow	INTO instruction.
5	#BR	BOUND Range Exceeded	BOUND instruction.
6	#UD	Invalid Opcode (UnDefined Opcode)	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Floating-point instruction.[2]
10	#TS	Invalid TSS	Task switch or TSS access.
11	#NP	Segment Not Present	Loading segment registers or accessing system segments.
12	#SS	Stack Segment Fault	Stack operations and SS register loads.
13	#GP	General Protection	Any memory reference and other protection checks.
14	#PF	Page Fault	Any memory reference.
15		Reserved	
16	#MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Any data reference in memory.[3]
18	#MC	Machine Check	Error codes (if any) and source are model dependent.[4]
19	#XM	SIMD Floating-Point Exception	SIMD Floating-Point Instruction[5]
20	#VE	Virtualization Exception	EPT violations[6]
21-31		Reserved	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT *n* instruction.

Intel Reserved

CS5250 - 2021/2022 Sem 2 25

- More specifically, the first fucntion `x86_init.irqs.pre_vector_init()` in `native_init_IRQ()` , is points to the `init_ISA_irqs()` function in the `arch/x86/kernel/irqinit.c` source code file: according to the `/arch/x86/kernel/x86_init.c` source code file




```
        x86_platform.get_wallclock = get_rtc_noop;
        x86_platform.set_wallclock = set_rtc_noop;
    }
}

/*
 * The platform setup functions are preset with the default functions
 * for standard PC hardware.
 */
struct x86_init_ops x86_init __initdata = {

    .resources = {
        .probe_roms             = probe_roms,
        .reserve_resources      = reserve_standard_io_resources,
        .memory_setup           = e820__memory_setup_default,
    },

    .mpparse = {
        .setup_ioapic_ids       = x86_init_noop,
        .find_smp_config        = default_find_smp_config,
        .get_smp_config         = default_get_smp_config,
    },

    .irqs = {
        .pre_vector_init        = init_ISA_irqs,
        .intr_init              = native_init_IRQ,
        .intr_mode_select       = apic_intr_mode_select,
        .intr_mode_init         = apic_intr_mode_init,
        .create_pci_msi_domain  = native_create_pci_msi_domain,
    },

    .oem = {
        .arch_setup             = x86_init_noop,
        .banner                 = default_banner,
    },

    .paging = {
        .pagetable_init         = native_pagetable_init,
    },

    .timers = {
        .setup_percpu_clockev   = setup_boot_APIC_clock,
        .timer_init             = hpet_time_init,
        .wallclock_init         = x86_wallclock_init,
    },
```

  - `init_ISA_irqs()` function in the `arch/x86/kernel/irqinit.c` source code file: it makes initialization of the `ISA` related interrupts

How the table vector_irq[] is used in interrupt processing

1. `vector_irq[]` will be used during the first steps of an external hardware interrupt handling.
2. Every time an interrupt come in the kernel will use the `vector_irq[]` to look for the corresponding IDT entry.
    - Special exceptions have special handlers
    - General interrupts eventually make their way to `common_interrupt()` (used to be called do_IRQ)
    - `common_interrupt()` is defined in `/arch/x86/kernel/irq.c`