

# 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()`

```

linux / init / main.c
All symbols ▾ Search Identifier
Filter tags
1000      /*
1001      * Set up housekeeping before setting up workqueues to allow the unbound
1002      * workqueue to take non-housekeeping into account.
1003      */
1004      housekeeping_init();
1005
1006      /*
1007      * Allow workqueue creation and work item queueing/cancelling
1008      * early. Work item execution depends on kthreads and starts after
1009      * workqueue_init().
1010      */
1011      workqueue_init_early();
1012
1013      rcu_init();
1014
1015      /* Trace events are available after this */
1016      trace_init();
1017
1018      if (initcall_debug)
1019          initcall_debug_enable();
1020
1021      context_tracking_init();
1022      /* init some links before init_ISA_irqs() */
1023      early_irq_init();
1024      init_IRQ();
1025      tick();
1026      rcu_init_nohz();
1027      init_timers();
1028      srmu_init();
1029      hrtimers_init();
1030      softirq_init();
1031      timekeeping_init();
1032      kfence_init();
1033
1034      /*
1035      * For best initial stack canary entropy, prepare it after:
1036      * - setup_arch() for any UEFI RNG entropy and boot cmdline access
1037      * - timekeeping_init() for ktime entropy used in rand_initialize()
1038      * - rand_initialize() to get any arch-specific entropy like RDRAND
1039      * - add_latent_entropy() to get any latent entropy
1040      * - adding command line entropy
1041      */
1042      rand_initialize();
1043      add_latent_entropy();
1044      add_device_randomness(command_line, strlen(command_line));
1045

```

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

```

linux / arch / x86 / kernel / irqinit.c
All symbols ▾ Search Identifier
Filter tags
57      /*
58      * Try to set up the through-local-APIC virtual wire mode earlier.
59      *
60      * On some 32-bit UP machines, whose APIC has been disabled by BIOS
61      * and then got re-enabled by "lapic", it hangs at boot time without this.
62      */
63      init_bsp_APIC();
64
65      legacy_pic->init(0);
66
67      for (i = 0; i < nr_legacy_irqs(); i++)
68          irq_set_chip_and_handler(i, chip, handle_level_irq);
69
70 }
71
72 void __init init_IRQ(void)
73 {
74     int i;
75
76     /*
77     * On cpu 0, Assign ISA_IRQ_VECTOR(irq) to IRQ 0..15.
78     * If these IRQ's are handled by legacy interrupt-controllers like PIC,
79     * then this configuration will likely be static after the boot. If
80     * these IRQs are handled by more modern controllers like IO-APIC,
81     * then this vector space can be freed and re-used dynamically as the
82     * IRQ's migrate etc.
83     */
84     for (i = 0; i < nr_legacy_irqs(); i++)
85         per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);
86
87     BUG_ON(irq_init_percpu_irqstack(smp_processor_id()));
88
89     x86_init.irqs.intr_init();
90
91 }
92
93 void __init native_init_IRQ(void)
94 {
95     /* Execute any quirks before the call gates are initialised: */
96     x86_init.irqs.pre_vector_init();
97
98     id_setup_apic_and_irq_gates();
99     lapic_assign_system_vectors();
100
101     if (!acpi_iopanic && !of_iopanic && nr_legacy_irqs()) {
102         /* IRQ2 is cascade interrupt to second interrupt controller */
103         if (request_irq(2, no_action, IRQF_NO_THREAD, "cascade", NULL))
104             pr_err("%s: request_irq() failed\n", "cascade");

```

- 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.

```

linux / arch / x86 / kernel / irqinit.c
Filter tags
v5
v5.17
v5.16
v5.16.11
v5.16.10
v5.16.9
v5.16.8
v5.16.7
v5.16.6
v5.16.5
v5.16.4
v5.16.3
v5.16.2
v5.16.1
v5.16
v5.16.rc8
v5.16.rc7
v5.16.rc6
v5.16.rc5
v5.16.rc4
v5.16.rc3
v5.16.rc2
v5.16.rc1
v5.15
v5.14
v5.13
v5.12
v5.11
v5.10
v5.9
v5.8
v5.7
v5.6
void __init init_ISA_irq(void)
{
    struct irq_chip *chip = legacy_pic->chip;
    int i;

    /* Try to set up the through-local-APIC virtual wire mode earlier.
     * On some 32-bit UP machines, whose APIC has been disabled by BIOS
     * and then got re-enabled by "lalic", it hangs at boot time without this.
     */
    init_bsp_APIC();
    legacy_pic->init(0);
    for (i = 0; i < nr_legacy_irqs(); i++)
        irq_set_chip_and_handler(i, chip, handle_level_irq);
}
void __init init_IRO(void)
{
    /*
     * and then got re-enabled by "lalic", it hangs at boot time without this.
     */
    init_bsp_APIC();
    legacy_pic->init(0);
    for (i = 0; i < nr_legacy_irqs(); i++)
        irq_set_chip_and_handler(i, chip, handle_level_irq);
}
void __init init_IRQ(void)
{
    int i;

    /*
     * On cpu 0, Assign ISA IRQ_VECTOR(i) to IRQ 0..15.
     * If these IRO's are handled by legacy interrupt-controllers like PIC,
     * then this configuration will likely be static after the boot. If
     * these IRQs are handled by more modern controllers like IO-APIC,
     * then this vector space can be freed and re-used dynamically as the
     * irq's migrate etc.
     */
    for (i = 0; i < nr_legacy_irqs(); i++)
        per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);
    BUG_ON(irq_init_percpu_irqstack(smp_processor_id()));
    x86_init.irqs.intr_init();
}
void __init native_init_IRQ(void)
{
    /* Execute any quirks before the call gates are initialised: */
    x86_init.irqs.pre_vector_init();

    idt_setup_apic_and_irq_gates();
    lapic_assign_system_vectors();
    if (!acpi_ipic && !iosapic && nr_legacy_irqs() == 1)
        /* IRO2 is cascade interrupt to second interrupt controller */
        if (request_irq(2, no_action, IRQF_NO_THREAD, "cascade", NULL))
            pr_err("request_irq() failed\n");
}

```

- 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();`

```

linux / arch / x86 / kernel / irqinit.c
Filter tags
v5
v5.17
v5.16
v5.16.11
v5.16.10
v5.16.9
v5.16.8
v5.16.7
v5.16.6
v5.16.5
v5.16.4
v5.16.3
v5.16.2
v5.16.1
v5.16
v5.16.rc8
v5.16.rc7
v5.16.rc6
v5.16.rc5
v5.16.rc4
v5.16.rc3
v5.16.rc2
v5.16.rc1
v5.15
v5.14
v5.13
v5.12
v5.11
v5.10
v5.9
v5.8
v5.7
v5.6
void __init native_init_IRQ(void)
{
    /* Execute any quirks before the call gates are initialised: */
    x86_init.irqs.pre_vector_init();

    idt_setup_apic_and_irq_gates();
    lapic_assign_system_vectors();
    if (!acpi_ipic && !iosapic && nr_legacy_irqs() == 1)
        /* IRO2 is cascade interrupt to second interrupt controller */
        if (request_irq(2, no_action, IRQF_NO_THREAD, "cascade", NULL))
            pr_err("request_irq() failed\n");
}

```

- 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.



The screenshot shows the QEMU debugger interface with the assembly code for `x86_init.c`. The assembly code is highlighted in blue, and the instruction at address `0x100000000` is selected. The assembly code includes comments indicating the setup of wallclock and RTC noop, and the selection of standard PC hardware. It also defines the `x86_init_ops` structure with various resource and configuration pointers.

```
    .resources = {
        .probe_roms = probe_roms,
        .reserve_resources = reserve_standard_io_resources,
        .memory_setup = e820_memory_setup_default,
    },
    .mparse = {
        .setup_iopanic_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,
    },
},
```

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

linux

Filter tags

v5.17

v5.16

v5.16.11

v5.16.10

v5.16.9

v5.16.8

v5.16.7

v5.16.6

v5.16.5

v5.16.4

v5.16.3

v5.16.2

v5.16.1

v5.16

v5.16.c8

v5.16.c7

v5.16.c6

v5.16.c5

v5.16.c4

v5.16.c3

v5.16.c2

v5.16.c1

v5.15

v5.14

v5.13

v5.12

v5.11

v5.10

v5.9

v5.8

v5.7

v5.6

v5.5

/ arch / x86 / kernel / irqinit.c

All symbols ▾

Search Identifier

```
1 /*  
2  * init_bsp_APIC();  
3  
4  legacy_pic->init(0);  
5  
6  for (i = 0; i < nr_legacy_irqs(); i++)  
7      irq_set_chip_and_handler(i, chip, handle_level_irq);  
8 }  
9  
10 void __init init_IRQ(void)  
11 {  
12     int i;  
13  
14     /*  
15      * On cpu 0, Assign ISA IRQ_VECTOR(irq) to IRQ 0..15.  
16      * If these IRQ's are handled by legacy interrupt-controllers like PIC,  
17      * then this configuration will likely be static after the boot. If  
18      * these IRQs are handled by more modern controllers like IO-APIC,  
19      * then this vector space can be freed and re-used dynamically as the  
20      * irq's migrate etc.  
21     */  
22     for (i = 0; i < nr_legacy_irqs(); i++)  
23         per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);  
24  
25     BUG_ON(irq_init_percpu_irqstack(smp_processor_id()));  
26  
27     x86_init.irqs.intr_init();  
28 }  
29  
30 void __init native_init_IRQ(void)  
31 {  
32     /* Execute any quirks before the call gates are initialised:< */  
33     x86_init.irqs.pre_vector_init();  
34  
35     idt_setup_apic_and_irq_gates();  
36     lapic_assign_system_vectors();  
37  
38     if (!acpi_iopanic && !of_iopanic && nr_legacy_irqs()) {  
39         /* IRQ2 is cascade Interrupt to second interrupt controller */  
40         if (request_irq(2, no_action, IRQF_NO_THREAD, "cascade", NULL))  
41             pr_err("%s: request_irq() failed\n", "cascade");  
42     }  
43 }
```

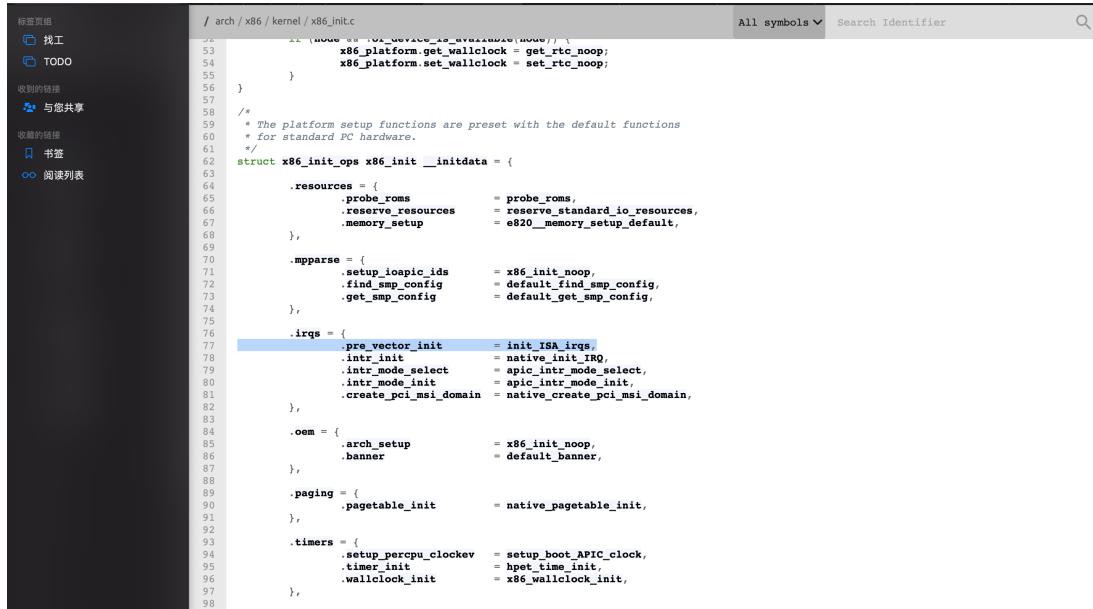
- in this function `native_init_IRQ()`, it will make initialization of the intel reserved interrupt vectors

# Intel Reserved Interrupts

Vector	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2	#NM	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. <sup>1</sup>
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. <sup>2</sup>
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. <sup>3</sup>
18	#MC	Machine Check	Error codes (if any) and source are model dependent. <sup>4</sup>
19	#XM	SMD Floating-Point Exception	SMD Floating-Point instruction <sup>5</sup>
20	#VE	Virtualization Exception	EPT violations <sup>6</sup>
21-31		Reserved	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT n instruction.

Intel Reserved

- More specifically, the first function `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



```

// arch/x86/kernel/x86_init.c
52     /* (CODE) (C) _x86_init_x86_gva_table(NODE) */
53     x86_platform.get_wallclock = get_rtc_noop;
54     x86_platform.set_wallclock = set_rtc_noop;
55 }
56 }
57 /*
58 * The platform setup functions are preset with the default functions
59 * for standard PC hardware.
60 */
61 */
62 struct x86_init_ops x86_init __initdata = {
63     .resources = {
64         .probe_roms = probe_roms,
65         .reserve_resources = reserve_standard_io_resources,
66         .memory_setup = e820__memory_setup_default,
67     },
68     .mpparse = {
69         .setup_apic_ids = x86_init_noop,
70         .find_smp_config = default_find_smp_config,
71         .get_smp_config = default_get_smp_config,
72     },
73     .irqs = {
74         .pre_vector_init = init_ISA_irqs,
75         .int_init = native_init_IRQ,
76         .int_level_select = apic_int_level_select,
77         .int_mode_init = apic_int_mode_init,
78         .create_pci_msi_domain = native_create_pci_msi_domain,
79     },
80     .oem = {
81         .arch_setup = x86_init_noop,
82         .banner = default_banner,
83     },
84     .paging = {
85         .pageable_init = native_pageable_init,
86     },
87     .timers = {
88         .setup_percpu_clockev = setup_boot_APIC_clock,
89         .timer_init = hpet_time_init,
90         .wallclock_init = x86_wallclock_init,
91     },
92 };
93
94
95
96
97
98

```

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

```

linux
/ arch / x86 / kernel / irqinit.c
All symbols ▾ Search Identifier
Filter tags

v5
  v5.17
  v5.16
    v5.16.11
    v5.16.10
    v5.16.9
    v5.16.8
    v5.16.7
    v5.16.6
    v5.16.5
    v5.16.4
    v5.16.3
    v5.16.2
    v5.16.1
    v5.16
    v5.16rc8
    v5.16rc7
    v5.16rc6
    v5.16rc5
    v5.16rc4
    v5.16rc3
    v5.16rc2
    v5.16rc1
  v5.15
  v5.14
  v5.13
  v5.12
  v5.11
  v5.10
  v5.9
  v5.8
  v5.7
  v5.6
  v5.5

30      * are unused but an SMP system is supposed to have enough memory ...
31      * sometimes (mostly wrt. hw bugs) we get corrupted vectors all
32      * across the spectrum, so we really want to be prepared to get all
33      * of these. Plus, more powerful systems might have more than 64
34      * IO-APIC registers.
35      *
36      *(these are usually mapped into the 0x30-0xff vector range)
37
38 DEFINE_PER_CPU(vector_irq_t, vector_irq) = {
39     [0 ... NR_VECTORS - 1] = VECTOR_UNUSED,
40 };
41
42 void __init __init_ISA_irqs(void)
43 {
44     struct irq_chip *chip = legacy_pic->chip;
45     int i;
46
47     /*
48     * Try to set up the through-local-APIC virtual wire mode earlier.
49     *
50     * On some 32-bit UP machines, whose APIC has been disabled by BIOS
51     * and then got re-enabled by "lspic", it hangs at boot time without this.
52     */
53     init_bsp_APIC();
54     legacy_pic->init(0);
55
56     for (i = 0; i < nr_legacy_irqs(); i++)
57         irq_set_chip_and_handler(i, chip, handle_level_irq);
58 }
59
60 void __init __init_IRQ(void)
61 {
62     int i;
63
64     /*
65     * On cpu 0, Assign ISA_IRQ_VECTOR(irq) to IRQ 0..15.
66     *
67     * These IRQs are handled by legacy interrupt controllers like PIC,
68     * their configuration will likely be static after the boot. If
69     * these IRQs are handled by more modern controllers like IO-APIC,
70     * then this vector space can be freed and re-used dynamically as the
71     * irq's migrate etc.
72     */
73     for (i = 0; i < nr_legacy_irqs(); i++)
74         per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);
75 }
76

```

## How the table `vector_irq[]` is used in interrupt processing

- `vector_irq[]` will be used during the first steps of an external hardware interrupt handling.
- 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>

```

linux
/ arch / x86 / kernel / irq.c
All symbols ▾ Search Identifier
Filter tags

v5
  v5.17
  v5.16
    v5.16.11
    v5.16.10
    v5.16.9
    v5.16.8
    v5.16.7
    v5.16.6
    v5.16.5
    v5.16.4
    v5.16.3
    v5.16.2
    v5.16.1
    v5.16
    v5.16rc8
    v5.16rc7
    v5.16rc6
    v5.16rc5
    v5.16rc4
    v5.16rc3
    v5.16rc2
    v5.16rc1
  v5.15
  v5.14
  v5.13
  v5.12
  v5.11
  v5.10
  v5.9
  v5.8
  v5.7
  v5.6
  v5.5

232     else
233         __handle_irq(desc, regs);
234     }
235
236     /*
237     * common_interrupt() handles all normal device IRQ's (the special SMP
238     * cross-CPU interrupts have their own entry points).
239     */
240     DEFINE_IDTENTRY_IRQ(common_interrupt)
241     {
242         struct pt_regs *old_regs = set_irq_regs(regs);
243         struct irq_desc *desc;
244
245         /* entry code tells RCU that we're not quiescent. Check it. */
246         RCU_LOCKDEP_WARN(!rcu_is_watching(), "IRQ failed to wake up RCU");
247
248         desc = __this_cpu_read(vector_irq[vector]);
249         if (likely(!IS_ERR_OR_NULL(desc))) {
250             handle_irq(desc, regs);
251         } else {
252             ack_APIC_irq();
253
254             if (desc == VECTOR_UNUSED) {
255                 pr_err_ratelimited("%s: %d.%u No irq handler for vector\n",
256                                   func, smp_processor_id(),
257                                   vector);
258             } else {
259                 __this_cpu_write(vector_irq[vector], VECTOR_UNUSED);
260             }
261         }
262         set_irq_regs(old_regs);
263     }
264
265 #ifdef CONFIG_X86_LOCAL_APIC
266     /* Function pointer for generic interrupt vector handling */
267     void (*x86_platform_ipi_callback)(void) = NULL;
268
269     /*
270     * Handler for X86_PLATFORM_IPI_VECTOR.
271     */
272     DEFINE_IDTENTRY_SYSVEC(sysvec_x86_platform_ipi)
273     {
274         struct pt_regs *old_regs = set_irq_regs(regs);
275
276         ack_APIC_irq();
277     }
278
279

```