

# Linux内核编程04期：中断

作者：王利涛 视频配套地址：https://wanglitaobao.com

系统启动过程中，每个控制器（irq\_domain）初始化，会建立硬件中断号（HW interrupt ID）和软件中断号（IRQ number）之间的映射

```
static int gic_irq_domain_map(struct irq_domain *d, unsigned int irq, irq_hw_number_t hw)
{
    struct gic_chip_data *gic = d->host_data;
    struct irq_data *irqd = irq_desc_get_irq_data(irq_to_desc(irq));

    switch (hw) {
        case 0 ... 15:
            irq_set_percpu_devid(irq);
            irq_domain_set_info(d, irq, hw, &gic->chip, d->host_data, handle_percpu_devid_fasteoi_irq, NULL, NULL);
            break;
        case 16 ... 31:
            irq_set_percpu_devid(irq);
            irq_domain_set_info(d, irq, hw, &gic->chip, d->host_data, handle_percpu_devid_irq, NULL, NULL);
            break;
        default:
            irq_domain_set_info(d, irq, hw, &gic->chip, d->host_data, handle_fasteoi_irq, NULL, NULL);
            irq_set_probe(irq);
            irqd_set_single_target(irqd);
            break;
    }

    /* Prevents SW retriggerers which mess up the ACK/EOI ordering */
    irqd_set_handle_enforce_irqctx(irqd);
    return 0;
}
```

```
static void
irq_desc_set_handler(struct irq_desc *desc, irq_flow_handler_t
handle, int is_chained, const char *name)
{
    desc->handler_irq = handle;
    desc->name = name;
    ...
}
```

```
kernel/irq/handle.c
irqreturn_t _handle_irq_event_percpu(struct irq_desc *desc,
unsigned int *flags)
{
    irqreturn_t retval = IRQ_NONE;
    unsigned int irq = desc->irq_data.irq;
    struct irqaction *action;

    for_each_action_of_desc(desc, action) {
        irqreturn_t res;
        trace_irq_handler_entry(irq, action);
        res = action->handler(irq, action->dev_id);
        trace_irq_handler_exit(irq, action, res);
        retval = res;
    }
    return retval;
}
```

```
kernel/irq/chip.c
void handle_fasteoi_irq(struct irq_desc *desc)
{
    struct irq_chip *chip = desc->irq_data.chip;
    raw_spin_lock(&desc->lock);
    desc->istate &= (IRQS_PENDING | IRQS_WAITING);
    statust_inn_irq_this_cpu(desc);
    if (desc->istate & IRQS_ONESHOT)
        handle_irq_event(desc);
    cond_unmask_eoi_irq(desc, chip);
    raw_spin_unlock(&desc->lock);
    return;
}

out:
if (!!(chip->flags & IRQCHIP_EOI_IF_HANDLED))
    chip->irq_eoi(&desc->irq_data);
raw_spin_unlock(&desc->lock);
}
```

```
kernel/irq/handle.c
irqreturn_t _handle_irq_event(struct irq_desc *desc)
{
    irqreturn_t ret;

    desc->istate &= "IRQS_PENDING;
    irqd_set(&desc->irq_data, IRQD_IRQ_INPROGRESS);
    raw_spin_unlock(&desc->lock);

    ret = handle_irq_event_percpu(desc);

    raw_spin_lock(&desc->lock);
    irqd_clear(&desc->irq_data, IRQD_IRQ_INPROGRESS);
    raw_spin_unlock(&desc->lock);
    return ret;
}
```

```
irq_desc[NR_IRQS]
struct irq_desc {
    struct irq_common_data irq_common_data;
    struct irq_data *irq_data;
    unsigned int _parent;
    struct irq_flow_handler_t handler_irq;
    struct irqaction_list; /* IRQ action list */
    atomic_t threads_handled;
}

struct irqaction {
    irq_handler_t handler;
    void *dev_id;
    void (*percpu)(struct irq_desc *desc, irq_hw_number_t);
    struct irqaction *next;
    irq_handler_t thread_fn;
    struct task_struct *thread;
    struct irqaction *secondary;
    unsigned int flags;
    unsigned long thread_flags;
    unsigned long thread_mask;
    const char *name;
    struct proc_entry *dir;
}

struct irq_data {
    struct irq_data *common;
    struct irq_data *local;
    struct irq_chip *chip;
    struct irq_domain *domain;
    void *chip_data;
};

struct irq_chip {
    struct device *parent_device;
    const char *name;
    unsigned int (*irq_startup)(struct irq_data *data);
    void (*irq_shutdown)(struct irq_data *data);
    void (*irq_enable)(struct irq_data *data);
    void (*irq_disable)(struct irq_data *data);
    void (*irq_ack)(struct irq_data *data);
    void (*irq_mask)(struct irq_data *data);
    void (*irq_unmask)(struct irq_data *data);
    void (*irq_eoi)(struct irq_data *data);
    unsigned long flags;
};
```

普通进程

指令1  
指令2  
指令3  
指令4  
指令5

Linux中断处理流程（基于Linux-5.10.4）

中断处理，CPU硬件自动完成的部分

ARM中断向量表

\_\_irq\_usr

\_\_irq\_usr

根据HW interrupt ID找到IRQ number，调用asm\_do\_IRQ

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

vector\_stub宏定义

ARM中断向量表

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

\_\_irq\_usr

tasklet的执行过程

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

tasklet的执行过程

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

kernel/softirq.c

workqueue工作队列工作流程

schedule\_work

queue\_work

queue\_work\_on

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

queue\_work

## 文档说明

本文档是Linux内核编程04期：中断，的配套文档，

结合视频教程、代码、PPT文档学习，效果更好。

视频地址：https://wanglitaobao.com