int request\_threaded\_irq(unsigned int irq, irq\_handler\_t handler, irq\_handler\_t thread\_fn,

                                                      unsigned long irqflags,const char \*devname, void \*dev\_id);

分析request\_threaded\_irq()函数中的各个形参

1>：irq:表示申请的中断号。

2>：handler:表示中断服务例程

3.> thread\_fn：中断线程化，此处传递的是NULL。NULL表示没有中断线程化。

    在 Linux 中，中断具有最高的优先级。不论在任何时刻，只要产生中断事件，内核将立即执行相应的中断处理程序，等到所有挂起的中断和软中断处理完毕后才能执行正常的任务，因此有可能造成实时任务得不到及时的处理。中断线程化之后，中断将作为内核线程运行而且被赋予不同的实时优先级，实时任务可以有比中断线程更高的优先级。这样，具有最高优先级的实时任务就能得到优先处理，即使在严重负载下仍有实时性保证。but,并不是所有的中断都可以被线程化，比如时钟中断，主要用来维护系统时间以及定时器等，其中定时器是操作系统的脉搏，一旦被线程化，就有可能被挂起，这样后果将不堪设想，所以不应当被线程化。

4>.irqflags:表示中断标志位。

5>.devname:表示请求中断的设备的名称。

6>.dev\_id: 对应于request\_irq()函数中所传递的第五个参数，可取任意值，但必须唯一能够代表发出中断请求的设备，通常取描述该设备的结构体。 共享中断时所用。

int devm\_request\_threaded\_irq(struct device \*dev, unsigned int irq,

         irq\_handler\_t handler, irq\_handler\_t thread\_fn,

         unsigned long irqflags, const char \*devname, void \*dev\_id)

{

     struct irq\_devres \*dr;

     int rc;

    dr = devres\_alloc(devm\_irq\_release, sizeof(struct irq\_devres),GFP\_KERNEL);

    if (!dr)

       return -ENOMEM;

   rc = request\_threaded\_irq(irq, handler, thread\_fn, irqflags,devname, dev\_id);

   if (rc) {

      devres\_free(dr);

      return rc;

   }

   dr->irq = irq;

   dr->dev\_id = dev\_id;

   devres\_add(dev, dr);

   return 0;

}

————————————————

版权声明：本文为CSDN博主「qwaszx523」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qwaszx523/article/details/52635431