前言

- 软中断（softirq）导致 CPU 使用率升高也是最常见的一种性能问题
- 所以软中断这个硬骨头必须啃下去！

回忆下什么是中断

- 中断是系统用来响应硬件设备请求的一种机制
- 它会**打断**进程的正常调度和执行
- 然后调用内核中的中断处理程序来**响应**硬件设备的请求

场景类比，加深印象

比如说你订了一份外卖，但是不确定外卖什么时候送到，也没有别的方法了解外卖的进度， 但是，配送员送外卖是不等人的，到了你这儿没人取的话，就直接走人了；所以你只能苦苦等着，时不时去门口看看外卖送到没，而不能干其他事情；不过呢，如果在订外卖的时候，你就跟配送员约定好，让他送到后给你打个电话，那你就不用苦苦等待了，就可以去忙别的事情，直到电话一响，接电话、取外卖就可以了、

- **打电话：**其实就是一个中断，没接到电话的时候，你可以做其他的事情
- 只有接到了电话（也就是发生中断），你才要进行另一个动作：**取外卖**

中断的优势

一种异步的事件处理机制，可以提高系统的并发处理能力

中断运行时间短

- 由于中断处理程序**会打断其他进程的运行**，为了减少对正常进程运行调度的影响，中断处理程序就需要尽可能快地运行
- 如果中断要处理的事情很多，中断服务程序就有可能要运行很长时间

中断处理程序在**响应中断**

会临时关闭中断。这就会导致上一次中断处理完成之前，其他中断都不能响应，也就是说中断有可能会丢失

响应中断场景类比

假如你订了 2 份外卖，一份主食和一份饮料，并且是由 2 个不同的配送员来配送。这次你不用时时等待着，两份外卖都约定了电话取外卖的方式。但是，问题又来了，当第一份外卖送到时，配送员给你打了个长长的电话，商量发票的处理方式。与此同时，第 二个配送员也到了，也想给你打电话。 但是很明显，因为电话占线（也就是**关闭了中断响应**），第二个配送员的电话是打不通的。 所以，第二个配送员很可能试几次后就走掉了（也就是**丢失了一次中断**）

软中断

中断处理过程分割

- **为了解决中断处理程序执行过长和中断丢失的问题**，Linux 会将中断处理过程**分成两个阶段**，也就是上半部和下半部
- **上半部：**快速处理中断，它在中断禁止模式下运行，主要处理跟硬件紧密相关的或时间敏感的工作
- **下半部：**延迟处理上半部**未完成**的工作，通常以**内核线程**的方式运行

承上启下

- 上面说到的响应中断场景
- **上半部**就是你接听电话，告诉配送员你已经知道了，其他事儿见面再说，然后电话就可以挂断了
- **下半部**才是取外卖的动作，以及见面后商量发票处理的动 作。

网卡接收数据包的栗子

网卡接收到数据包后，会通过**硬件中断**的方式，通知内核有新的数据到了。这时，内核就应该调用中断处理程序来响应它

上半部

1. 快速处理
2. 首先，要把网卡的数据读到内存中
3. 然后，**更新**一下硬件寄存器的状态（表示数据已经读好了）
4. 最后，再发送一个**软中断信号**，**通知下半部**做进一步的处理

下半部

1. 被软中断信号唤醒
2. 需要从内存中找到网络数据，再按照网络协议栈，对数据进行逐层解析和处理，直到把它送给应用程序

总结

上半部

- 直接处理**硬件**请求，也就是**硬中断**
- **特点：**快速执行
- 会打断 CPU 正在执行的任务，然后立即执行中断处理程序

下半部

- 由**内核**触发，也就是**软中断**
- **特点：**延迟执行
- 以内核线程的方式执行，并且**每个 CPU 都对应一个软中断内核线程**，名字为 “ksoftirqd/CPU 编号”，比如说， 0 号 CPU 对应的软中断内核线程的名字就是 ksoftirqd/0
- 不只包括了硬件设备中断处理程序的下半部，一些内核自定义的事件也属于软中断，网络收发、定时、调度、RCU 锁等各种类型
- 内核调度和 RCU 锁（Read-Copy Update）， RCU 是 Linux 内核中最常用的锁之一

查看软中断和内核线程

proc 文件系统

它是一种内核空间和用户空间进行通信的机制，可以用来查看内核的数据结构，或者用来动态修改内核的配置

- `/proc/softirqs` ： 提供了软中断的运行情况
- `/proc/interrupts` ： 提供了硬中断的运行情况

查看软中断文件内容

```
$ cat /proc/softirqs
CPU0 CPU1
HI: 0 0
TIMER: 811613 1972736
NET_TX: 49 7
NET_RX: 1136736 1506885
BLOCK: 0 0
IRQ_POLL: 0 0
TASKLET: 304787 3691
SCHED: 689718 1897539
HRTIMER: 0 0
RCU: 1330771 1354737
```

注意软中断的类型

- 从第一列可以看出，软中断包括了 10 个类别
- **比如：** NET_RX 表示网络接收中断，而 NET_TX 表示网络发送中断

注意同一种软中断在不同 CPU 上的分布情况

- 也就是同一行的内容
- 正常情况 下，同一种中断在不同 CPU 上的累积次数应该差不多
- **比如：**上面的，NET_RX 在 CPU0 和 CPU1 上的中断次数基本是同一个数量级，相差不大

TASKLET

- TASKLET 在不同 CPU 上的分布并不均匀
- **TASKLET 是最常用的软中断实现机制**，每个 TASKLET **只运行一次**就会结束 ，并且只在调用它的函数所在的 CPU 上运行
- 存在的问题： 由于只在一个 CPU 上运行导致的调度不均衡，再比如因为不能在多个 CPU 上并行运行带来了性能限制

查看软中断线程

```
1   ps aux | grep softirq
```

```
root@ubuntu:~# ps aux | grep softirq
root         10  0.0  0.0      0     0 ?        S    21:40   0:00 [ksoftirqd/0]
root         18  0.0  0.0      0     0 ?        S    21:40   0:00 [ksoftirqd/1]
root      48253  0.0  0.0  14424  1000 pts/0    S+   23:33   0:00 grep --color=auto softirq
```

- 注意，这些线程的名字外面都有中括号，这说明 ps **无法获取它们的命令行参数** （cmline）
- 一般来说，ps 的输出中，名字括在中括号里的，一般都是内核线程