Linux除了内核态外还有用户态。Linux2.6以前，linux kernel只致辞用户态抢占

当从中断处理例程返回到内核态时，kernel会检查是否可以抢占和是否需要重新调度。

Kernel可以在热河时间点上抢占一个任务（因为中断可以发生在任何时间点上）。只要在这个时间点上Kernel的状态是安全的、可重新调度的。

Linux在每个任务的thread\_info结构中增加了preempt\_count变量作为Preemtion的计数器。这个变量初始值为0，当加锁时计数器增一，当解锁时计数器减一

不能被抢占的情况：

1. linux内核进程不能抢占中断（中断只能被其他中断终止、抢占）在中断例程中不允许进行进程调度。
2. 内核代码正持有spinlock、writelock/readlock等锁，处于这种锁保护中
3. 内核正在执行调用程序scheduler。
4. 内核正在对per-cpu数据结构操作时
5. **struct** thread\_info {
6. unsigned **long**       flags;      /\* low level flags \*/
7. mm\_segment\_t        addr\_limit; /\* address limit \*/
8. **struct** task\_struct  \*task;      /\* main task structure \*/
9. **struct** exec\_domain  \*exec\_domain;   /\* execution domain \*/
10. **struct** restart\_block    restart\_block;
11. **int**         preempt\_count;  /\* 0 => preemptable, <0 => bug \*/
12. **int**         cpu;        /\* cpu \*/
13. };

Entry.S中

Preemt\_count被划分为了几个部分，分别用来表示抢占计数、软中断计数、硬件中断计数、NMI计数

\* PREEMPT\_MASK: 0x000000ff

\* SOFTIRQ\_MASK: 0x0000ff00

\* HARDIRQ\_MASK: 0x000f0000

\* NMI\_MASK: 0x00100000

\* PREEMPT\_ACTIVE: 0x00200000

1. asmlinkage **void** \_\_sched notrace preempt\_schedule(**void**)
2. {
3. /\*
4. \* If there is a non-zero preempt\_count or interrupts are disabled,
5. \* we do not want to preempt the current task. Just return..
6. \*/
7. **if** (likely(!preemptible())) /\*preempt计数不为0或者当前禁止硬件中断则不抢占\*/
8. **return**;
10. **do** {
11. \_\_preempt\_count\_add(PREEMPT\_ACTIVE);
12. \_\_schedule();
13. \_\_preempt\_count\_sub(PREEMPT\_ACTIVE);
15. /\*
16. \* Check again in case we missed a preemption opportunity
17. \* between schedule and now.
18. \*/
19. barrier();
20. } **while** (need\_resched());  /\*判断flag为TIF\_NEED\_RESCHED置\*/
21. }