Struct thread\_info {

Unsigned long flags;

Int preempt\_count;

Mm\_segment\_t addr\_limit;

Struct task\_struct \*task;

\_\_u32 cpu;

\_\_u32 cpu\_domain;

};

1. Irq->kernel (entry-armv.S) \_\_irq\_svc
2. Load struct thread\_info preempt\_count 到r8
3. Load struct thread\_info flags 到r0
4. 判断r8是否是0
5. 如果r8不是0，则r0赋值为0
6. 检测r0的\_TIF\_NEED\_RESCHED是否置位
7. 如果置位，则进入svc\_preempt
8. 将lr赋值给r8
9. 进入preempt\_schedule\_irq
10. 进入\_\_schedule(true)逻辑
11. 将struct thread\_info flags load到r0
12. 判断r0的\_TIF\_NEED\_RESCHED是否置位
13. 如果没置位，则返回到r8
14. 如果置位，则调到1b
15. Irq->user(entry-armv.S) \_\_irq\_usr
16. 在irq\_handler之后，进入ret\_to\_user\_from\_irq
17. 将struct thread\_info的flags load到r1
18. 判断r1的\_TIF\_WORK\_MASK是否置位
19. 如果置位，则进入slow\_work\_pending
20. 进入do\_work\_pending
21. 在do\_work\_pending中，首先判断struct thread\_info的\_TIF\_NEED\_RESCHED是否置位，如果置位，则进入schedule()
22. Exception->user(entry-armv.S) \_\_dabt\_usr

在dabt\_helper之后，进入ret\_from\_exception

1. Ret\_from\_exception调用ret\_to\_user
2. Ret\_to\_user会进入ret\_to\_user\_from\_irq
3. Syscalls (entry-common.S) vector\_swi
4. 会调用ret\_fast\_syscall
5. 同时也会进入do\_work\_pending
6. Preempt\_enable的定义由CONFIG\_PREEMPT\_COUNT宏和CONFIG\_PREEMPT控制，一般CONFIG\_PREEEMPT使能，CONFIG\_PREEMPT\_COUNT即使能。

那我们看一下CONFIG\_PREEMPT\_COUNT和CONFIG\_PREEMPT都使能的例子:

1. Preempt\_count\_dec\_and\_test
2. Struct thread\_info 的preempt\_count减一
3. 如果为零，则判断struct thread\_info的TIF\_NEED\_RESCHED是否置位，如果置位，则返回1；否则，则返回0；
4. 如果 preempt\_count\_dec\_and\_test返回1，则进入\_\_preempt\_schedule函数。
5. 首先判读preemptible
6. 首先判读struct thread\_info preempt\_count是否为0，如果为0，则判断irq是否禁止，如果不禁止，则返回1；否则返回0；
7. 如果preemptibel返回1，则进入preempt\_schedule\_common中，即进入schedule中
8. Cond\_resched

Cond\_resched首先调用should\_resched判读struct thread\_info preempt\_count为0和struct thread\_info的flags的TIF\_NEED\_RESCHED置位，如果是，则进入preempt\_schedule\_common函数中。