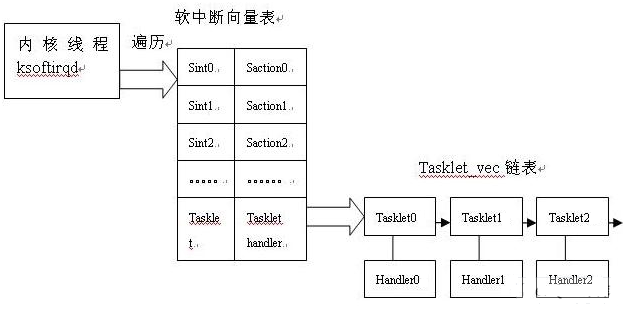
linux设备驱动编写tasklet机制

　　在编写设备驱动时， tasklet 机制是一种比较常见的机制，通常用于减少中断处理的时间，将本应该是在中断服务程序中完成的任务转化成软中断完成。

　　为了最大程度的避免中断处理时间过长而导致中断丢失，有时候我们需要把一些在中断处理中不是非常紧急的任务放在后面执行，而让中断处理程序尽快返回。在老版本的 linux 中通常将中断处理分为 top half handler 、 bottom half handler 。利用 top half handler 处理中断必须处理的任务，而 bottom half handler 处理不是太紧急的任务。

　　但是 linux2.6 以后的 linux 采取了另外一种机制，就是软中断来代替 bottom half handler 的处理。而 tasklet 机制正是利用软中断来完成对驱动 bottom half 的处理。 Linux2.6 中软中断通常只有固定的几种： HI\_SOFTIRQ( 高优先级的 tasklet ，一种特殊的 tasklet) 、 TIMER\_SOFTIRQ （[定时器](http://www.dzsc.com/product/searchfile/5431.html)）、 NET\_TX\_SOFTIRQ （网口发送）、 NET\_RX\_SOFTIRQ （网口接收） 、 BLOCK\_SOFTIRQ （块设备）、 TASKLET\_SOFTIRQ （普通 tasklet ）。当然也可以通过直接修改内核自己加入自己的软中断，但是一般来说这是不合理的，软中断的优先级比较高，如果不是在内核处理频繁的任务不建议使用。通常驱动用户使用 tasklet 足够了。

　　软中断和 tasklet 的关系如下图：



　　上图可以看出， ksoftirqd 是一个后台运行的内核线程，它会周期的遍历软中断的向量列表，如果发现哪个软中断向量被挂起了（ pend ），就执行对应的处理函数，对于 tasklet 来说，此处理函数就是 tasklet\_action ，这个处理函数在系统启动时初始化软中断的就挂接了。

　　Tasklet\_action 函数，遍历一个全局的 tasklet\_vec 链表（此链表对于 SMP 系统是每个 [CPU](http://www.dzsc.com/product/searchfile/5590.html) 都有一个），此链表中的元素为 tasklet\_struct 。此结构如下 :

　　struct tasklet\_struct

　　{

　　struct tasklet\_struct \*next;

　　unsigned long state;

　　atomic\_t count;

　　void (\*func)(unsigned long);

　　unsigned long data;

　　};

　　每个结构一个函数指针，指向你自己定义的函数。当我们要使用 tasklet ，首先新定义一个 tasklet\_struct 结构，并初始化好要执行函数指针，然后将它挂接到 task\_vec 链表中，并发一个软中断就可以等着被执行了。

　　原理大概如此，对于 linux 驱动的作者其实不需要关心这些，关键是我们如何去使用 tasklet 这种机制。

　　Linux 中提供了如下[接口](http://www.dzsc.com/product/searchfile/5540.html)：

　　DECLARE\_TASKLET([name](http://www.dzsc.com/stock-ic/NAME.html),function,data) ：此接口初始化一个 tasklet ；其中 name 是 tasklet 的名字， function 是执行 tasklet 的函数； data 是 unsigned long 类型的 function 参数。

　　static inline void tasklet\_schedule(struct tasklet\_struct \*t) ：此接口将定义后的 tasklet 挂接到 cpu 的 tasklet\_vec 链表，具体是哪个 cpu 的 tasklet\_vec 链表，是根据当前线程是运行在哪个 cpu 来决定的。此函数不仅会挂接 tasklet ，而且会起一个软 tasklet 的软中断 , 既把 tasklet 对应的中断向量挂起 (pend) 。

　　两个工作完成后，基本上可以了， tasklet 机制并不复杂，很容易的使程序尽快的响应中断，避免造成中断丢失。