Linux系统内核定时器机制详解（下）（3） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144979.htm 7．6．3．7 定时器迁移操作 由于一个定时器的interval值会随着时间的不断流逝（即jiffies值的不断增大）而不断变小，因此那些原本到期紧迫程度较低的定时器会随着jiffies值的不断增大而成为既将马上到期的定时器。比如定时器向量tv2.vec[0]中的定时器在经过256个时钟滴答后会成为未来256个时钟滴答内会到期的定时器。因此，定时器在内核动态定时器链表中的位置也应相应地随着改变。改变的规则是：当tv1.index重新变为0时（意味着tv1中的256个定时器向量都已被内核扫描一遍了，从而使tv1中的256个定时器向量变为空），则用tv2.vec［index］定时器向量中的定时器去填充tv1，同时使tv2.index加1（它以64为模）。当tv2.index重新变为0（意味着tv2中的64个定时器向量都已经被全部填充到tv1中去了，从而使得tv2变为空），则用tv3.vec［index］定时器向量中的定时器去填充tv2。如此一直类推下去，直到tv5。 函数cascade_timers()完成这种定时器迁移操作，该函数只有一个timer_vec结构类型指针的参数tv。这个函数将把定时器向量tv->vec［tv->index］中的所有定时器重新填充到上一层定时器向量中去。如下所示（kernel/timer.c）： static inline void cascade_timers(struct timer_vec *tv) { /* cascade all the timers from tv up one level */ struct list_head *head, *curr, *next. head = tv->vec tv->index. curr = head->next. /* * We are removing _all_ timers from the list, so we dont have to * detach them individually, just clear the list afterwards. */ while (curr != head) {

struct timer_list *tmp. tmp = list_entry(curr, struct timer_list, list). next = curr->next. list_del(curr). // not needed internal_add_timer(tmp). curr = next. } INIT_LIST_HEAD(head). tv->index = (tv->index 1) &amp. TVN_MASK. }对该函数的注释如下：（1）首先，用指针head指向定时器头部向量头部的list_head结构。指针curr指向定时器向量中的第一个定时器。（2）然后，用一个while{}循环来遍历定时器向量tv->vec［tv->index］。由于定时器向量是一个双向循环队列，因此循环的终止条件是curr=head。对于每一个被扫描的定时器，循环体都先调用list_del()函数将当前定时器从链表中摘除，然后调用internal_add_timer()函数重新确定该定时器应该被放到哪个定时器向量中去。（3）当从while{}循环退出后，定时器向量tv->vec［tv->index］中所有的定时器都已被迁移到其它地方（到它们该呆的地方：－），因此它本身就成为一个空队列。这里我们显示地调用INIT_LIST_HEAD()宏来将定时器向量的表头结构初始化为空。（4）最后，将tv->index值加1，当然它是以64为模。