# TODO

[linux 内核定时器 timer\_list详解](https://www.cnblogs.com/leaven/archive/2010/08/19/1803382.html)

<https://www.cnblogs.com/leaven/archive/2010/08/19/1803382.html>

中断的学习

https://blog.csdn.net/xiezhi123456/article/details/80414804

# timer

## 数据结构

timer\_list

内核定时器的数据结构

struct timer\_list {

struct list\_head entry;

unsigned long expires;

void (\*function)(unsigned long);

unsigned long data;

struct tvec\_base \*base;

/\* ... \*/

};

其中 expires 字段表示期望定时器执行的 jiffies 值，到达该 jiffies 值时，将调用 function 函数，并传递 data 作为参数。当一个定时器被注册到内核之后，entry 字段用来连接该定时器到一个内核链表中。base 字段是内核内部实现所用的。

需要注意的是 expires 的值是32位的，因为内核定时器并不适用于长的未来时间点。

## 初始化

在使用 struct timer\_list 之前，需要初始化该数据结构，确保所有的字段都被正确地设置。初始化有两种方法。

### 方法一：DEFINE\_TIMER

DEFINE\_TIMER(timer\_name, function\_name, expires\_value, data);

该宏会静态创建一个名叫 timer\_name 内核定时器，并初始化其 function, expires, name 和 base 字段。

### 方法二：setup\_timer(

struct timer\_list mytimer;

setup\_timer(&mytimer, (\*function)(unsigned long), unsigned long data);

mytimer.expires = jiffies + 5\*HZ;

### 方法3：init\_timer

struct timer\_list mytimer;

init\_timer(&mytimer);

mytimer ->timer.expires = jiffies + 5\*HZ;

mytimer ->timer.data = (unsigned long) dev;

mytimer ->timer.function = &corkscrew\_timer; /\* timer handler \*/

通过init\_timer()动态地定义一个定时器，此后，将处理函数的地址和参数绑定给一个timer\_list，

注意，无论用哪种方法初始化，其本质都只是给字段赋值，所以只要在运行 add\_timer() 之前，expires, function 和 data 字段都可以直接再修改。

关于上面这些宏和函数的定义，参见 include/linux/timer.h。

## 注册（注销）add\_timer(

定时器要生效，还必须被连接到内核专门的链表中，这可以通过 add\_timer(struct timer\_list \*timer) 来实现。

重新注册

要修改一个定时器的调度时间，可以通过调用 mod\_timer(struct timer\_list \*timer, unsigned long expires)。mod\_timer() 会重新注册定时器到内核，而不管定时器函数是否被运行过。

注销

注销一个定时器，可以通过 del\_timer(struct timer\_list \*timer) 或 del\_timer\_sync(struct timer\_list \*timer)。其中 del\_timer\_sync 是用在 SMP 系统上的（在非SMP系统上，它等于del\_timer），当要被注销的定时器函数正在另一个 cpu 上运行时，del\_timer\_sync() 会等待其运行完，所以这个函数会休眠。另外还应避免它和被调度的函数争用同一个锁。对于一个已经被运行过且没有重新注册自己的定时器而言，注销函数其实也没什么事可做。

int timer\_pending(const struct timer\_list \*timer)

这个函数用来判断一个定时器是否被添加到了内核链表中以等待被调度运行。注意，当一个定时器函数即将要被运行前，内核会把相应的定时器从内核链表中删除（相当于注销）

## 使用说明

LINUX内核定时器是内核用来控制在未来某个时间点（基于jiffies）调度执行某个函数的一种机制，其实现位于 <linux/timer.h> 和 kernel/timer.c 文件中。被调度的函数肯定是异步执行的，它类似于一种“软件中断”，而且是处于非进程的上下文中，所以调度函数必须遵守以下规则：

1) 没有 current 指针、不允许访问用户空间。因为没有进程上下文，相关代码和被中断的进程没有任何联系。

2) 不能执行休眠（或可能引起休眠的函数）和调度。

3) 任何被访问的数据结构都应该针对并发访问进行保护，以防止竞争条件。

内核定时器的调度函数运行过一次后就不会再被运行了（相当于自动注销），但可以通过在被调度的函数中重新调度自己来周期运行。

在SMP系统中，调度函数总是在注册它的同一CPU上运行，以尽可能获得缓存的局域性

## 列子

### 1

#include <linux/module.h>

#include <linux/timer.h>

#include <linux/jiffies.h>

struct timer\_list mytimer;

static void myfunc(unsigned long data)

{

printk("%s/n", (char \*)data);

mod\_timer(&mytimer, jiffies + 2\*HZ);

}

static int \_\_init mytimer\_init(void)

{

setup\_timer(&mytimer, myfunc, (unsigned long)"Hello, world!");

mytimer.expires = jiffies + HZ;

add\_timer(&mytimer);

return 0;

}

static void \_\_exit mytimer\_exit(void)

{

del\_timer(&mytimer);

}

module\_init(mytimer\_init);

module\_exit(mytimer\_exit);

### 2

例子2

static struct timer\_list power\_button\_poll\_timer;

static void power\_button\_poll(unsigned long dummy)

{

if (gpio\_line\_get(N2100\_POWER\_BUTTON) == 0) {

ctrl\_alt\_del();

return;

}

power\_button\_poll\_timer.expires = jiffies + (HZ / 10);

add\_timer(&power\_button\_poll\_timer);

}

static void \_\_init n2100\_init\_machine(void)

{

；

；

init\_timer(&power\_button\_poll\_timer);

power\_button\_poll\_timer.function = power\_button\_poll;

power\_button\_poll\_timer.expires = jiffies + (HZ / 10);

add\_timer(&power\_button\_poll\_timer);

}

### 例子3

设备open时初始化和注册定时器

static int corkscrew\_open(struct net\_device \*dev)

{

；

；

init\_timer(&vp->timer);

vp->timer.expires = jiffies + media\_tbl[dev->if\_port].wait;

vp->timer.data = (unsigned long) dev;

vp->timer.function = &corkscrew\_timer; /\* timer handler \*/

add\_timer(&vp->timer);

：

；

}

定时器超时处理函数，对定时器的超时时间重新赋值

static void corkscrew\_timer(unsigned long data)

{

；

；

vp->timer.expires = jiffies + media\_tbl[dev->if\_port].wait;

add\_timer(&vp->timer);

；

；

}

设备close时删除定时器

static int corkscrew\_close(struct net\_device \*dev)

{

；

；

del\_timer(&vp->timer);

；

；

}

### 4

本例子用DEFINE\_TIMER静态创建定时器

#include <linux/module.h>

#include <linux/jiffies.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/timer.h>

#include <linux/leds.h>

static void ledtrig\_ide\_timerfunc(unsigned long data);

DEFINE\_LED\_TRIGGER(ledtrig\_ide);

static DEFINE\_TIMER(ledtrig\_ide\_timer, ledtrig\_ide\_timerfunc, 0, 0);

static int ide\_activity;

static int ide\_lastactivity;

void ledtrig\_ide\_activity(void)

{

ide\_activity++;

if (!timer\_pending(&ledtrig\_ide\_timer))

mod\_timer(&ledtrig\_ide\_timer, jiffies + msecs\_to\_jiffies(10));

}

EXPORT\_SYMBOL(ledtrig\_ide\_activity);

static void ledtrig\_ide\_timerfunc(unsigned long data)

{

if (ide\_lastactivity != ide\_activity) {

ide\_lastactivity = ide\_activity;

led\_trigger\_event(ledtrig\_ide, LED\_FULL);

mod\_timer(&ledtrig\_ide\_timer, jiffies + msecs\_to\_jiffies(10));

} else {

led\_trigger\_event(ledtrig\_ide, LED\_OFF);

}

}

static int \_\_init ledtrig\_ide\_init(void)

{

led\_trigger\_register\_simple("ide-disk", &ledtrig\_ide);

return 0;

}

static void \_\_exit ledtrig\_ide\_exit(void)

{

led\_trigger\_unregister\_simple(ledtrig\_ide);

}

module\_init(ledtrig\_ide\_init);

module\_exit(ledtrig\_ide\_exit);

==================================================================================

add\_timer() -- 将定时器添加到定时器等待队列中

2007年08月04日 星期六 15:30

用add\_timer()函数来看timer\_base的作用

static inline void add\_timer(struct timer\_list \*timer)

{

BUG\_ON(timer\_pending(timer));

\_\_mod\_timer(timer, timer->expires);

}

int \_\_mod\_timer(struct timer\_list \*timer, unsigned long expires)

{

tvec\_base\_t \*base, \*new\_base;

unsigned long flags;

int ret = 0;

timer\_stats\_timer\_set\_start\_info(timer);

BUG\_ON(!timer->function);

base = lock\_timer\_base(timer, &flags);

如果timer已经放到定时链表中,则释放开

|--------------------------------|

| if (timer\_pending(timer)) { -|

| detach\_timer(timer, 0); -|

| ret = 1; |

| } |

|--------------------------------|

获取当前CPU的timer base

|-----------------------------------------|

| new\_base = \_\_get\_cpu\_var(tvec\_bases); |

|-----------------------------------------|

如果当前CPU的timer base不是当前timer中的base, 更新timer的base

|----------------------------------------------------|

| if (base != new\_base) { |

| if (likely(base->running\_timer != timer)) { -|

| timer->base = NULL; |

| spin\_unlock(&base->lock); |

| base = new\_base; |

| spin\_lock(&base->lock); |

| timer->base = base; |

| } |

| } |

|----------------------------------------------------|

给定时器timer设置超时时间；并添加该时钟

|-------------------------------------|

| timer->expires = expires; |

| internal\_add\_timer(base, timer); -|

|-------------------------------------|

spin\_unlock\_irqrestore(&base->lock, flags);

return ret;

}

MODULE\_AUTHOR("Richard Purdie <rpurdie@openedhand.com>");

MODULE\_DESCRIPTION("LED IDE Disk Activity Trigger");

MODULE\_LICENSE("GPL");

## 小结

总的来说，timer的用法还是很简单的。主要需要定义一个timer\_list变量timer、先初始化timer

　　init\_timer(&timer);

　　then 对timer的相关参数赋值：

　　timer.function = fun;

　　timer.expires = jiffies + TIMER\_DELAY;

　　add\_timer(&timer);

　　在定时器时间到的时候，会执行fun，如果继续定时，可以通过

　　在fun中执行

　　mod\_timer(&timer, jiffies + TIMER\_DELAY);

　　在不需要的时候通过调用

　　del\_timer(&timer);

　　删除定时器。

　　简单吧。这样一个简单的定时器就完成了。

　　呵呵。

　　附程序：

　　#include <linux/module.h>

　　#include <linux/types.h>

　　#include <linux/fs.h>

　　#include <linux/errno.h>

　　#include <linux/mm.h>

　　#include <linux/sched.h>

　　#include <linux/init.h>

　　#include <linux/cdev.h>

　　#include <asm/io.h>

　　#include <asm/system.h>

　　#include <asm/uaccess.h>

　　#include <linux/timer.h>

　　#include <asm/atomic.h>

　　#define SECOND\_MAJOR 0

　　static int second\_major = SECOND\_MAJOR;

　　struct second\_dev

　　{

　　struct cdev cdev;

　　atomic\_t counter;

　　struct timer\_list s\_timer;

　　};

　　struct second\_dev \*second\_devp;

　　static void second\_timer\_handle(unsigned long arg)

　　{

　　mod\_timer(&second\_devp->s\_timer, jiffies + HZ);

　　atomic\_inc(&second\_devp->counter);

　　printk(KERN\_ERR "current jiffies is %ld/n",jiffies);

　　}

　　int second\_open(struct inode \*inode, struct file \*filp)

　　{

　　init\_timer(&second\_devp->s\_timer);

　　second\_devp->s\_timer.function = &second\_timer\_handle;

　　second\_devp->s\_timer.expires = jiffies + HZ;

　　add\_timer(&second\_devp->s\_timer);

　　atomic\_set(&second\_devp->counter, 0);

　　return 0;

　　}

　　int second\_release(struct inode \*inode, struct file \*filp)

　　{

　　del\_timer(&second\_devp->s\_timer);

　　return 0;

　　}

　　static ssize\_t second\_read(struct file \*filp, char \_\_user \*buf, size\_t count,

　　loff\_t \*ppos)

　　{

　　int counter;

　　counter = atomic\_read(&second\_devp->counter);

　　if (put\_user(counter, (int \*)buf))

　　{

　　return -EFAULT;

　　}else

　　{

　　return sizeof(unsigned int);

　　}

　　}

　　static const struct file\_operations second\_fops =

　　{

　　.owner = THIS\_MODULE,

　　.open = second\_open,

　　.release = second\_release,

　　.read = second\_read,

　　};

　　static void second\_setup\_cdev(struct second\_dev \*dev, int index)

　　{

　　int err, devno = MKDEV(second\_major, index);

　　cdev\_init(&dev->cdev, &second\_fops);

　　dev->cdev.owner = THIS\_MODULE;

　　dev->cdev.ops = &second\_fops;

　　err = cdev\_add(&dev->cdev, devno, 1);

　　if (err)

　　{

　　printk(KERN\_NOTICE "Error %d add second%d", err, index);

　　}

　　}

　　int second\_init(void)

　　{

　　int ret;

　　dev\_t devno = MKDEV(second\_major, 0);

　　if (second\_major)

　　{

　　ret = register\_chrdev\_region(devno, 1, "second");

　　}else

　　{

　　ret = alloc\_chrdev\_region(&devno, 0, 1, "second");

　　second\_major = MAJOR(devno);

　　}

　　if (ret < 0)

　　{

　　return ret;

　　}

　　second\_devp = kmalloc(sizeof(struct second\_dev), GFP\_KERNEL);

　　if (!second\_devp)

　　{

　　ret = -ENOMEM;

　　goto fail\_malloc;

　　}

　　memset(second\_devp, 0, sizeof(struct second\_dev));

　　second\_setup\_cdev(second\_devp, 0);

　　return 0;

　　fail\_malloc:

　　unregister\_chrdev\_region(devno, 1);

　　}

　　void second\_exit(void)

　　{

　　cdev\_del(&second\_devp->cdev);

　　kfree(second\_devp);

　　unregister\_chrdev\_region(MKDEV(second\_major, 0), 1);

　　}

　　MODULE\_AUTHOR("Song Baohua");

　　MODULE\_LICENSE("Dual BSD/GPL");

　　module\_param(second\_major, int, S\_IRUGO);

　　module\_init(second\_init);

　　module\_exit(second\_exit);

　　附上用户端的测试程序：

　　#include <stdio.h>

　　#include <unistd.h>

　　#include <fcntl.h>

　　int main(void)

　　{

　　int fd, i;

　　int data;

　　fd = open("/dev/second",O\_RDONLY);

　　if (fd < 0)

　　{

　　printf("open /dev/second error/n");

　　}

　　for(i = 0; i < 20; i++)

　　{

　　read(fd, &data, sizeof(data));

　　printf("read /dev/second is %d/n",data);

　　sleep(1);

　　}

　　close(fd);

　　}

[复制代码](javascript:void(0);)

## 参考

https://www.cnblogs.com/sky-heaven/p/5157653.html

init\_timer

# irq.drivers

devm\_request\_threaded\_irq

https://blog.csdn.net/xiezhi123456/article/details/80414804