

一、时钟中断

硬件有一个时钟装置，该装置每隔一定时间发出一个时钟中断（称为一次时钟嘀嗒-tick），对应的中断处理程序就将全局变量jiffies_64加1

jiffies_64 是一个全局64位整型，jiffies全局变量为其低32位的全局变量，程序中一般用jiffies

HZ: 可配置的宏，表示1秒钟产生的时钟中断次数，一般设为100或200

二、延时机制

1. 短延迟：忙等待

```
1. void ndelay(unsigned long nsecs)
2. void udelay(unsigned long usecs)
3. void mdelay(unsigned long msecs)
```

2. 长延迟：忙等待

使用jiffies比较宏来实现

```
time_after(a,b)    //a > b
time_before(a,b)   //a < b

//延迟100个jiffies
unsigned long delay = jiffies + 100;
while(time_before(jiffies,delay))
{
    ;
}

//延迟2s
unsigned long delay = jiffies + 2*HZ;
while(time_before(jiffies,delay))
{
    ;
}
```

3. 睡眠延迟----阻塞类

```
void msleep(unsigned int msecs);

unsigned long msleep_interruptible(unsigned int msecs);
```

Q what? 哦，可初信号中断的。

延时机制的选择原则:

1. 异常上下文中只能采用忙等待类
2. 任务上下文短延迟采用忙等待类，长延迟采用阻塞类

----- 长 - 短

sleep

三、定时器

(1) 定义定时器结构体

```
struct timer_list
{
    struct list_head entry;
    unsigned long expires; // 期望的时间值 jiffies + x * HZ
    void (*function)(unsigned long); // 时间到达后，执行的回调函数，软中断异常上下文
    unsigned long data;
};
```

Q what? 去查表头? 包含双向链表头, 用于插入.

Q what? 回调函数

(2) 初始化定时器

```
init_timer(struct timer_list *)
```

XF. 它是统一放一地方统一加时间.

(3) 增加定时器 ----- (定时器开始计时)

```
void add_timer(struct timer_list *timer);
```

(4) 删除定时器 ----- (定时器停止工作)

```
int del_timer(struct timer_list * timer);
```

(5) 修改定时器

```
int mod_timer(struct timer_list *timer, unsigned long expires);
```

(定时处理的)

定义struct timer_list t1类型的变量

function 中用)

```
init_timer(...); // 模块入口函数
```

```
// 模块入口函数或open或希望定时器开始工作的地方
t1.expires = jiffies + n * HZ // n秒
```

```

t1.function = xxx_func;
t1.data = ...;

add_timer(...);

//不想让定时器继续工作时
del_timer(...);

void xxx_func(unsigned long arg)
{
    .....
    mod_timer(...); //如需要定时器继续隔指定时间再次调用本函数
}

```

✓ open: 初始化
开始

四、课堂练习一秒设备

Q what? close: 结束

先"板"下来, 之后处理问题吧.

read: copy-to-user (size小
返回, 大致int大小用)

(初始化) (更之前)
function (计时 续命)

