1. **Jiffies与HZ**

自系统启动开始，系统时钟中断周期性产生，每产生一个中断，jiffies加1，HZ保存的值决定系统1秒内产生多少个时间中断，大多数平台为100或1000

在内核中，一般以jiffies来表示时间，如：

#include <linux/jiffies.h>

unsigned long j, stamp\_1, stamp\_half, stamp\_n;

j = jiffies;

stamp\_1 = j + HZ;               // 1 秒后

stamp\_half = j + HZ/2;          // 半秒后

stamp\_n = j + n \* HZ / 1000;    // 1 毫秒后

Jiffies会溢出，所以我们一般不直接比较两个jiffies，使用下面的函数比较两个jiffies

#include <linux/jiffies.h>

int time\_after(unsigned long a, unsigned long b);   // 如果a的时间笔b靠后，则返回真

int time\_before(unsigned long a, unsigned long b);

int time\_after\_eq(unsigned long a, unsigned long b);

int time\_before\_eq(unsigned long a, unsigned long b);

有时需要与用户空间程序交换时间表示, 用户空间使用 struct timeval 和 struct timespec 来表示时间.这些转换有如下函数完成

#include <linux/time.h>

unsigned long timespec\_to\_jiffies(struct timespec \*value);

void jiffies\_to\_timespec(unsigned long jiffies, struct timespec \*value);

unsigned long timeval\_to\_jiffies(struct timeval \*value);

void jiffies\_to\_timeval(unsigned long jiffies, struct timeval \*value);

1. **当前时间**

获取当时间（日常生活的时间）使用如下函数

#include <linux/time.h>

struct timespec current\_kernel\_time(void);

1. **延迟执行**

**简单的方法**

while (time\_before(jiffies, j1))

    cpu\_relax();

我们应避免这种方法，试想如果我们执行这段代码前禁止中断，那么jiffies将不会更新，如果我们还禁止了内核抢占，那么只有强制关机了

**使用睡眠超时**

wait\_queue\_head\_t wait;

init\_waitqueue\_head (&wait);

wait\_event\_interruptible\_timeout(wait, 0, delay);   // 这里第二个参数 唤醒条件 为 0，延迟 delay

或者

#include <linux/sched.h>

signed long schedule\_timeout(signed long timeout);  // 休眠timeout个时钟滴答

**短延迟**

如果我们要休眠的时间很短，则可以使用如下函数

#include <linux/delay.h>

void ndelay(unsigned long nsecs);   // 纳秒

void udelay(unsigned long usecs);   // 微妙

void mdelay(unsigned long msecs);   // 毫秒

1. **内核定时器**

一个内核定时器是一个数据结构, 它在用户定义的时间执行用户定义的函数. 这个实现位于 <linux/timer.h>

内核提供给驱动许多函数来声明, 注册, 以及去除内核定时器. 下列的引用展示了基本的代码块:

#include <linux/timer.h>

struct timer\_list

{

    /\* ... \*/

    unsigned long expires;

    void (\*function)(unsigned long);

    unsigned long data;

};

void init\_timer(struct timer\_list \*timer);

struct timer\_list TIMER\_INITIALIZER(\_function, \_expires, \_data);

void add\_timer(struct timer\_list \* timer);

int del\_timer(struct timer\_list \* timer);

expires 要运行函数的时间（jiffies）

function 要运行的函数

Data 传给函数的参数

1. **Tasklt**

tasklt没有定时功能，使用tasklt可以向内核转递一个任务，内核会在其后的某个时间去执行这个任务

Tasklt使用前必须初始化

#include <linux/interrupt.h>

struct tasklet\_struct

{

    /\* ... \*/

    void (\*func)(unsigned long);

    unsigned long data;

};

void tasklet\_init(struct tasklet\_struct \*t, void (\*func)(unsigned long), unsigned long data);

DECLARE\_TASKLET(name, func, data);

DECLARE\_TASKLET\_DISABLED(name, func, data);

tasklet\_schedule(&short\_tasklet);       // 调用tasklt，系统会在合适的时间调用该tasklt

1. **工作队列**

tasklet 执行的很快, 短时期, 并且在原子态, 而工作队列函数可能有高周期且不需要是原子

**创建工作队列**

struct workqueue\_struct \*create\_workqueue(const char \*name);

struct workqueue\_struct \*create\_singlethread\_workqueue(const char

\*name);

**创建工作**

DECLARE\_WORK(name, void (\*function)(void \*), void \*data);

或者，动态创建使用：

INIT\_WORK(struct work\_struct \*work, void (\*function)(void \*), void

\*data);

PREPARE\_WORK(struct work\_struct \*work, void (\*function)(void \*), void

\*data);

**将工作提交到工作队列**

int queue\_work(struct workqueue\_struct \*queue, struct work\_struct

\*work);

int queue\_delayed\_work(struct workqueue\_struct \*queue, struct

work\_struct \*work, unsigned long delay);

如果使用 queue\_delay\_work，需要经过delay（jiffies）时间后才会执行

**取消某个工作**

int cancel\_delayed\_work(struct work\_struct \*work);

**销毁工作队列**

void destroy\_workqueue(struct workqueue\_struct \*queue);