粤嵌教育

Linux时间管理

www.gec-edu.org

内核时钟

粤嵌教育

- 硬件为内核提供了一个时钟定时器，用来计算流逝的时间，该时钟称为内核时钟、系统定时器、软件时钟、滴答时钟。
- 内核时钟以某种频率自行触发时钟中断，该频率可以通过编程预定，称节拍率。
- 内核时钟连续两次时钟中断的间隔时间称为节拍（tick）。
- 内核时钟的工作频率（节拍率）是通过静态预处理定义的，也就是HZ，为1秒内时钟中断的次数，在系统启动时按照HZ对硬件进行设置。体系结构不同，HZ的值也不同。
- 内核在文件<asm/param.h>中定义了HZ的实际值，节拍率就是HZ，周期为1/HZ。
  - # define HZ　CONFIG_HZ　/* Internal kernel timer frequency */
  - 在内核源码的根目录.config文件中：

    CONFIG_HZ = 200

jiffies

- 全局变量jiffies用来记录自系统启动以来产生的节拍的总数，即记录了系统自开机以来，已经过了多少tick。
- 启动时，内核将jiffies初始化为0 。
- 每发生一次内核时钟中断，Jiffies变数会被加1。
- jiffes一秒内增加的值也就为HZ，如果系统运行时间以秒为单位计算，就等于jiffies/HZ。
- Jiffies定义在文件linux/jiffies.h中
  - #define __jiffy_data __attribute__((section(".data")))
  - extern u64 __jiffy_data jiffies_64;
  - extern unsigned long volatile __jiffy_data jiffies;
  - 在linux链接脚本文件vmlinux.lds.S中：
    - OUTPUT_ARCH(arm)
    - ENTRY(stext)
    - #ifndef __ARMEB__
    - jiffies = jiffies_64;
    - #else
    - jiffies = jiffies_64 + 4;
    - #endif

Jiffies应用举例

- #include<linux/jiffies.h>
-
- unsigned long j,stamp_1,stamp_half,stamp_n;
-
- j=jiffies; //read the current value
-
- stamp_1 = j+HZ; //1second in the future
-
- stamp_half = j+HZ/2; //0.5second in the future
-
- stamp_n = j+n*HZ/1000; // n milliseconds

jiffies wrap around

- 在32位体系结构上，此jiffies取整个jiffies_64变量的低32位;在64位体系结构上，jiffies_64和jiffies指的是同一个变量。
- 当jiffies的值超过它的最大存放范围后就会发生溢出。对于32位无符号长整型，最大取值为(2^32)-1,即429496795。如果节拍计数达到了最大值后还要继续增加，它的值就会回绕到0。
- 内核提供了四个宏来帮助比较节拍计数，它们能正确的处理节拍计数回绕的问题：

```
#define time_after(a,b)     \
        (typecheck(unsigned long, a) && \
        typecheck(unsigned long, b) && \
        ((long)(b) - (long)(a) < 0))
```

returns true if the time a is after time b.

用户空间和HZ

粤嵌教育

- 在2.6以前的内核中，如果改变内核中的HZ值会给用户空间中某些程序造成异常结果。因为内核是以节拍数/秒的形式给用户空间导出HZ，应用程序便依赖这个特定的HZ值。如果在内核中的HZ改变了，就打破了用户空间的常量关系---用户空间并不知道新的HZ值。
- 内核更改所有导出的jiffies值。内核定义了USER_HZ来代表用户空间看到的HZ值。在x86体系结构上，由于HZ值原来一直是100，所以USER_HZ值就定义为100。
- 内核可以使用宏jiffies_to_clock_t()将一个有HZ表示的节拍计数转换为一个由USER_HZ表示的节拍计数。

内核定时器

粤嵌教育

- **内核定时器的主要作用：**
  - 更新系统运行时间(uptime)
  - 更新当前墙上时间(wall time)
  - 在对称多处理器系统(SMP)上，均衡调度各处理器上的运行队列
  - 检查当前进程是否用完了时间片(time slice)，如果用尽，则进行重新调度
  - 运行超时的动态定时器
  - 更新资源耗尽和处理器时间的统计值

- **内核定时器依赖于系统时钟中断，因为只有在系统时钟中断发生后内核才会去检查当前是否有超时的动态定时器。**

时间比较

- 通过两个时间点的比较，以确定时间点之间的先后次序。
- #include<linux/jiffies.h>
- int time_after(unsigned long a, unsigned long b);
- 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);
- 举例：
  - int demo_function()

```
{
        unsigned long timeout=jiffies+10*HZ/1000;
        do_time_task();
        if(time_after(jiffies, timeout))
        return task_timeout();
}
```

时间转换

用户空间：**timeval,timespec**
内核空间：**jiffies**

```
#include<linux/time.h>

struct timespec {
    time_t    tv_sec;        /* seconds */
    long    tv_nsec;    /* nanoseconds */
};

struct timeval {
    time_t        tv_sec;        /* seconds */
    SUSEconds_t    tv_usec;    /* microseconds */
};
```

接口函数：

- <include/linux/jiffies.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);

延时操作

粤嵌教育

- 忙等待
  - while(time_before(jiffies, j1))
    { cpu_relax(); }
  - #include<linux/delay.h>
    - void ndelay(unsignedlong nsecs);
    - void udelay(unsignedlong usecs);
    - void mdelay(unsignedlong msecs);

- 睡眠
  - while(time_before(jiffies, j1)){
    schedule();
    }
  - #include<linux/delay.h>
    - void msleep(unsignedint millisecs);
    - unsigned long msleep_interruptible(unsignedint millisecs);
    - void ssleep(unsignedint seconds)

内核定时器

- 内核定时器由数据结构timer_list表示，我们称该数据结构为内核定时器节点。
- <inlcude/linux/timer.h>

```
struct timer_list {
        struct list_head entry;
        unsigned long expires;
        void (*function)(unsigned long);
        unsigned long data;
        struct tvec_base *base;
        int slack;
};
```

- expires：该无符号长整型变量，保存了该定时器的超时时间，用于和内核变量jiffies进行比较。
- function：该函数指针变量保存了内核定时器超时后要执行的函数，即定时器超时处理函数。定时器超时处理函数将在中断上下文中执行。
- 成员变量data：该无符号长整型变量，用作定时器超时处理函数的参数。

内核定时器

粤嵌教育

- **1、声明一个内核定时器数据结构：**
- struct timer_list my_timer;

- **2、对内核定时器结构进行初始化：**
- init_timer(&my_timer);

- **3、设置内核定时器的超时时间expires、超时处理函数function、超时处理函数所使用的参数data：**
- my_timer.expires = jiffies + delay;
- my_timer.data = 0;
- my_timer.function = my_function;

- **4、激活内核定时器：**
- add_timer(&my_timer);

- 通过上面4步，我们就创建了一个内核定时器节点my_timer。该内核定时器在当前时刻以后delay个时钟中断后超时，执行超时处理函数my_function，传给超时处理函数的参数为0。

内核定时器

- void init_timer(struct timer_list *timer);
- void add_timer(struct timer_list *timer);
- int mod_timer(struct timer_list *timer，unsigned long expires)：
- 该函数负责修改内核定时器timer的超时字段expires。该函数可以修改激活和没有激活的内核定时器的超时时间，并把它们都设置为激活状态；

  返回值为0表示修改的内核定时器在修改之前处于未激活状态，返回值为1表示修改的内核定时器在修改之前处于已激活状态。
- int del_timer(structtimer_list* timer)
- int del_timer_sync(structtimer_list* timer)
- 这两个函数负责从链表中删除内核定时器timer。它们的区别在于，后者在多处理器系统中会确保其他处理器上没有处理或者处理完毕当前内核定时器timer时才退出。


谢谢!
THANKS