Linux操作系统分析
Chapter 7 Linux的时钟和定时测量

陈香兰（xlanchen@ustc.edu.cn）

计算机应用教研室@计算机学院
嵌入式系统实验室@苏州研究院
中国科学技术大学
Fall 2014

January 14, 2015

Outline

定时测量

- Linux内核提供两种主要的定时测量
  1. 获得当前的时间和日期
     - 系统调用：time()，ftime()以及gettimeofday()
  2. 维持定时器
     - settimer()，alarm()
- 定时测量是由基于固定频率振荡器和计数器的几个硬件电路完成的

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

Linux的计时体系结构

- Linux的计时体系结构
  - 更新自系统启动以来所经过的时间
  - 更新时间和日期
  - 确定当前进程的执行时间，考虑是否要抢占
  - 更新资源使用统计计数
  - 检查到期的软定时器

计时体系结构中的关键数据结构和变量

1. ARM中的系统时钟system_timer和sys_timer结构
   - 时钟中断发生源
   - 参见sys_timer数据结构
2. arm的滴答产生机制：时钟中断→tick
   - timer_tick()→do_timer()
3. Jiffies变量
4. 计时时钟源
5. Xtime变量

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

ARM中的系统时钟system_timer

- ARM的系统时钟：全局量system_timer
  - 是一个struct sys_timer*指针
  - 在文件arch/arm/kernel/time.c中定义
  - 在Linux初始化过程中得到初始化

start_kernel()→ setup_arch()@arch/arm/kernel/setup.c:

```
...
system_timer = mdesc->timer;
...
```

数据结构struct sys_timer

- 提供了与具体时钟中断源的接口
- 注册方法：machine_desc.timer

include/asm-arm/mach/time.h

```
/*
 * This is our kernel timer structure.
 * ...  */
struct sys_timer {
      struct sys_device dev;
      void (*init)(void);
      void (*suspend)(void);
      void (*resume)(void);
 #ifndef CONFIG_GENERIC_TIME
      unsigned long (*offset)(void);
 #endif

#ifdef CONFIG_NO_IDLE_HZ
      struct dyn_tick_timer *dyn_tick;
 #endif
};
```

以s3c2410为例

```
struct sys_timer s3c24xx_timer = {
        .init = s3c2410_timer_init,
        .offset = s3c2410_gettimeoffset,
        .resume = s3c2410_timer_setup
};
```

- 请搜索一下s3c24xx_timer的使用情况（即注册情况）

sys_timer的init接口的调用

- 在arch/arm/kernel/time.c的time_init中

start_kernel()→time_init():

```
    ...
    system_timer->init();
    ...
```

- 在sys_timer的init接口函数中，以s3c2410_timer_init为例：

```
static void __init s3c2410_timer_init (void) {
        s3c2410_timer_setup();
        setup_irq(IRQ_TIMER4, &s3c2410_timer_irq);
}
```

初始化时钟，并将中断处理函数s3c2410_timer_irq关联到中断号IRQ_TIMER4上

系统时钟小结

init/main.c :: start_kernel()

arch/arm/kernel/setup.c :: setup_arch()

arch/arm/kernel/time.c :: time_init()

初始化system_timer

调用system_timer的init函数指针

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

arm的滴答产生机制

- arm中，系统时钟的周期性时钟中断用来产生滴答，其方法是在时钟中断处理函数中调用timer_tick函数
- 以s3c24xx_timer为例，时钟中断处理函数如下：

arch/arm/plat-s3c24xx/time.c

```
/*
 * IRQ handler for the timer
 */
static irqreturn_t s3c2410_timer_interrupt(int irq, void *dev_id) {
        timer_tick();
        return IRQ_HANDLED;
}
```

- timer_tick调用Linux体系结构无关的do_timer()

arm的滴答产生机制

```
#ifndef CONFIG_GENERIC_CLOCKEVENTS
/*
 * Kernel system timer support.
 */
void timer_tick(void) {
        profile_tick(CPU_PROFILING);
        do_leds();
        do_set_rtc();
        write_seqlock(&xtime_lock);
        do_timer(1);
        write_sequnlock(&xtime_lock);
#ifndef CONFIG_SMP
        update_process_times(user_mode(get_irq_regs()));
#endif
}
#endif
```

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

Jiffies变量

- Jiffies变量用来记录系统自启动以来系统产生的tick数，每次时钟中断+1。其定义方式如下：

jiffies_64在kernel/timer.c中定义：

```
u64 jiffies_64 __cacheline_aligned_in_smp = INITIAL_JIFFIES;
EXPORT_SYMBOL(jiffies_64);
```

在include/linux/jiffies.h中

```
...
#define __jiffy_data __attribute__((section(".data")))
...
extern u64 __jiffy_data jiffies_64;
extern unsigned long volatile __jiffy_data jiffies;
...
/*
 * Have the 32 bit jiffies value wrap 5 minutes after boot
 * so jiffies wrap bugs show up earlier.
 */
#define INITIAL_JIFFIES ((unsigned long)(unsigned int) (-300*HZ))
```

Jiffies变量

- jiffies_64和32位的jiffies的关系：

jiffies在arch/arm/kernel/vmlinux.lds.S中定义

```
...
OUTPUT_ARCH(arm)
ENTRY(stext)

#ifndef __ARMEB__
jiffies = jiffies_64;
#else
jiffies = jiffies_64 + 4;
#endif
...
```

在vmlinux的符号表中，可以看到这两个变量在同一个地址上

```
c0314554 D jiffies
c0314554 D jiffies_64
```

Jiffies变量

- jiffies变量的更新函数

kernel/timer.c

```
/*
 * The 64-bit jiffies value is not atomic - you MUST NOT read it
 * without sampling the sequence number in xtime_lock.
 * jiffies is defined in the linker script...
 */
void do_timer(unsigned long ticks) {
     jiffies_64 += ticks;
     update_times(ticks);
}
```

jiffies小结

1. jiffies和jiffies_64的维护与体系结构无关，在kernel/timer.c中
2. jiffies_64的定义在kernel/timer.c中
3. jiffies的定义与体系结构相关，在arch/arm/kernel/vmlinux.lds.S中
4. jiffies的产生源（即滴答的产生源）是arm的system_timer

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

时钟源机制

- 时钟源抽象
  - 是系统时钟源，定义了系统时钟源的接口
  - 数据结构include/linux/clocksource.h::clocksource
- 时钟源列表clocksource_list：
  - 按照各自的rating值由高到低排序
  - 时钟源注册/注销函数：clocksource_register()/clocksource_unregister()：将指定的时钟源插入到时钟源列表中，或者从中移除。
- 缺省时钟源：具有最低rating值（=1）的Jiffies时钟源（clocksource_jiffies）
- 当前时钟源指针curr_clocksource指向当前所用的时钟源。最开始使用缺省时钟源。
- 时钟源的更新时机：kernel/time/timekeeping.c::update_wall_time结尾处。

时钟源机制

在kernel/time/clocksource.c中：

```
/* XXX - Would like a better way for initializing curr_clocksource */
extern struct clocksource clocksource_jiffies;
/*[Clocksource internal variables]———
 * curr_clocksource: ...
 * next_clocksource:
 * pending next selected clocksource.
 * clocksource_list:
 * linked list with the registered clocksources
 * clocksource_lock: ...
 * override_name:
 * Name of the user-specified clocksource.
 */
static struct clocksource *curr_clocksource = &clocksource_jiffies;
static struct clocksource *next_clocksource;
static struct clocksource *clocksource_override;
static LIST_HEAD(clocksource_list);
static DEFINE_SPINLOCK(clocksource_lock);
static char override_name[32];
static int finished_booting;
```

缺省时钟源：jiffies时钟源

- 参见kernel/time/jiffies.c

```
static cycle_t jiffies_read(void) {
    return (cycle_t) jiffies;
}

struct clocksource clocksource_jiffies = {
    .name = "jiffies",
    .rating = 1, /* lowest valid rating*/
    .read = jiffies_read,
    .mask = 0xffffffff, /*32bits*/
    .mult = NSEC_PER_JIFFY << JIFFIES_SHIFT, /* details above */
    .shift = JIFFIES_SHIFT,
};

static int __init init_jiffies_clocksource(void) {
    return clocksource_register(&clocksource_jiffies);
}
core_initcall(init_jiffies_clocksource);
```

以at91的时钟源clk32k为例

arch/arm/mach-at91/at91rm9200_time.c

```
static struct clocksource clk32k = {
     .name = " 32k_counter" ,
     .rating = 150,
     .read = read_clk32k,
     .mask = CLOCKSOURCE_MASK(20),
     .shift = 10,
     .flags = CLOCK_SOURCE_IS_CONTINUOUS,
};
```

Outline

xtime变量

- xtime：存放当前时间和日期

kernel/time/timekeeping.c

```
/*
 * The current time
 * wall_to_monotonic is what we need to add to xtime (or xtime corrected
 * for sub jiffie times) to get to monotonic time.  Monotonic is pegged
 * at zero at system boot time, so wall_to_monotonic will be negative,
 * however, we will ALWAYS keep the tv_nsec part positive so we can use
 * the usual normalization.
 *
 * wall_to_monotonic is moved after resume from suspend for the monotonic
 * time not to jump.  We need to add total_sleep_time to wall_to_monotonic
 * to get the real boot based time offset.
 *
 * - wall_to_monotonic is no longer the boot time, getboottime must be
 * used instead.
 */
struct timespec xtime __attribute__ ((aligned (16)));
struct timespec wall_to_monotonic __attribute__ ((aligned (16)));
static unsigned long total_sleep_time; /* seconds */
static struct timespec xtime_cache __attribute__ ((aligned (16)));
```

xtime变量

- xtime使用数据结构timespec

timespec@include/linux/time.h

```
#ifndef _STRUCT_TIMESPEC
#define _STRUCT_TIMESPEC
struct timespec {
    time_t tv_sec; /* seconds */
    long tv_nsec; /* nanoseconds */
};
#endif
```

- 时间纪元（Epoch）：即时间的起点 1970-01-01 00:00:00 +0000 午夜 (UTC).
- 时间的单位

xtime变量

- Xtime的更新
  - 基本上每个tick更新一次
  - 参见：update_wall_time@kernel/time/timekeeping.c
    - 根据时钟源来更新xtime的秒数和纳秒数
    - 时钟源

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

时钟中断处理

1. 仍以s3c24xx_timer的s3c2410_timer_interrupt为例：
   - 该函数调用timer_tick()
   - timer_tick()调用do_timer(1)

时钟中断处理

2. do_timer() @ kernel/timer.c
   1. 更新jiffies
   2. 更新xtime

```
/*
 * Called by the timer interrupt. xtime_lock must already be taken
 * by the timer IRQ!
 */
static inline void update_times(unsigned long ticks) {
    update_wall_time();
    calc_load(ticks);
}

/*
 * The 64-bit jiffies value is not atomic - you MUST NOT read it
 * without sampling the sequence number in xtime_lock.
 * jiffies is defined in the linker script...
 */
void do_timer(unsigned long ticks) {
     jiffies_64 += ticks;
     update_times(ticks);
}
```

时钟中断处理

3. update_process_times() @ kernel/timer.c
   1. 更新软定时器
   2. 调用调度器的tick函数：scheduler_tick()

```
/*
 * Called from the timer interrupt handler to charge one tick to the current
 * process. user_tick is 1 if the tick is user time, 0 for system.
 */
void update_process_times(int user_tick) {
    struct task_struct *p = current;
    int cpu = smp_processor_id();
    /* Note: this timer irq context must be accounted for as well. */
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
```

时钟中断处理

4. scheduler_tick() @ kernel/sched.c
   - 调用当前进程所属调度类的task_tick函数

```
/*
 * This function gets called by the timer code, with HZ frequency.
 * We call it with interrupts disabled.
 *
 * It also gets called by the fork code, when changing the parent’ s
 * timeslices.
 */
void scheduler_tick(void) {
    int cpu = smp_processor_id();
    struct rq *rq = cpu_rq(cpu);
    struct task_struct *curr = rq->curr;

    sched_clock_tick();

    spin_lock(&rq->lock);
    update_rq_clock(rq);
    update_cpu_load(rq);
    curr->sched_class->task_tick(rq, curr, 0);
    spin_unlock(&rq->lock);
    ...
```

Outline

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

软定时器

- 定时器是一种软件功能，它允许在将来的某个时刻调用某个函数
- 大多数设备驱动程序利用定时器完成一些特殊工作
  - 软盘驱动程序在软盘暂时不被访问时就关闭设备的发动机
  - 并行打印机利用定时器检测错误的打印机情况
- Linux中存在两类定时器：
  - 动态定时器：内核使用
  - 间隔定时器：由进程在用户态创建
  - 注意：
    由于软定时器在下半部分处理，内核不能保证定时器正好在时钟到期的时候被执行，会存在延迟，不适用于实时应用

动态定时器

- 动态定时器被动态的创建和撤销，当前活动的动态定时器个数没有限制
- 一个定时器由一个timer_list数据结构来定义，参见include/linux/timer.h

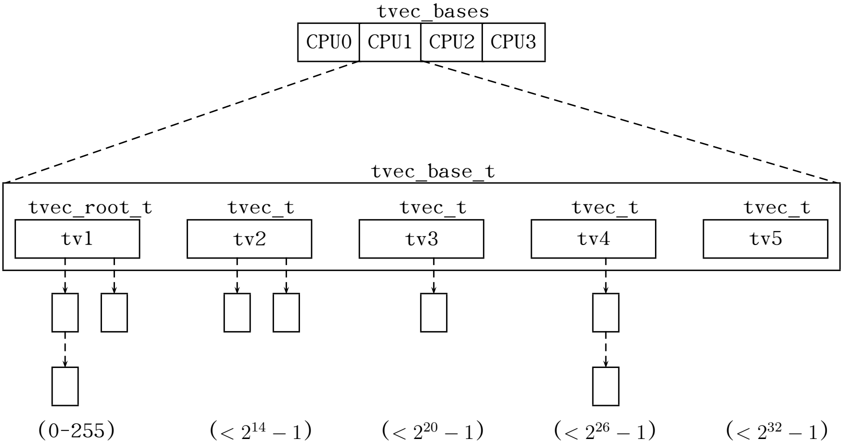
```
struct timer_list {
    struct list_head entry;
    unsigned long expires;

    void (*function)(unsigned long);
    unsigned long data;
    struct tvec_base *base;
    ...
};
```

创建并激活一个动态定时器

1. 创建一个新的timer_list对象
2. 调用init_timer初始化，并设置定时器要处理的函数和参数
3. 设置定时时间
4. 使用add_timer加入到合适的链表中

- 通常定时器只能执行一次，如果要周期性的执行必须再次将其加入链表

动态定时器的维护



Dynamic Timer Lists

维护用的数据结构

kernel/timer.c

```
/* * per-CPU timer vector definitions: */
#define TVN_BITS (CONFIG_BASE_SMALL ? 4 : 6)
#define TVR_BITS (CONFIG_BASE_SMALL ? 6 : 8)
#define TVN_SIZE (1 << TVN_BITS)
#define TVR_SIZE (1 << TVR_BITS)
#define TVN_MASK (TVN_SIZE - 1)
#define TVR_MASK (TVR_SIZE - 1)
```

```
Sstruct tvec {
    struct list_head vec[TVN_SIZE];
};

struct tvec_root {
    struct list_head vec[TVR_SIZE];
};

struct tvec_base {
    spinlock_t lock;
    struct timer_list *running_timer;
    unsigned long timer_jiffies;
    struct tvec_root tv1;
    struct tvec tv2;
    struct tvec tv3;
    struct tvec tv4;
    struct tvec tv5;
} ____cacheline_aligned;
```

```
struct tvec_base boot_tvec_bases;
EXPORT_SYMBOL(boot_tvec_bases); static
DEFINE_PER_CPU(struct tvec_base *, tvec_bases) = &boot_tvec_bases;
```

动态定时器的处理

- run_local_timers() @ kernel/timer.c在时钟中断处理过程中被update_process_times() @ kernel/timer.c调用

```
/*
 * Called by the local, per-CPU timer interrupt on SMP.
 */
void run_local_timers(void) {
    hrtimer_run_queues();
    raise_softirq(TIMER_SOFTIRQ);
    softlockup_tick();
}
```

- 软中断TIMER_SOFTIRQ对应的处理函数？？

init_timers()@kernel/timer.c

```
void __init init_timers(void) {
    ...
    open_softirq(TIMER_SOFTIRQ, run_timer_softirq, NULL);
}
```

动态定时器应用之delayed work

kernel/workqueue.c

```
int queue_delayed_work_on(int cpu, struct workqueue_struct *wq,
                struct delayed_work *dwork, unsigned long delay)
{
    int ret = 0;
    struct timer_list *timer = &dwork->timer;
    struct work_struct *work = &dwork->work;

    if (!test_and_set_bit(WORK_STRUCT_PENDING, work_data_bits(work))) {
        BUG_ON(timer_pending(timer));
        BUG_ON(!list_empty(&work->entry));
        timer_stats_timer_set_start_info(&dwork->timer);
        /* This stores cwq for the moment, for the timer_fn */
        set_wq_data(work, wq_per_cpu(wq, raw_smp_processor_id()));
        timer->expires = jiffies + delay;
        timer->data = (unsigned long)dwork;
        timer->function = delayed_work_timer_fn;
        if (unlikely(cpu >= 0))
            add_timer_on(timer, cpu);
        else
            add_timer(timer);
        ret = 1;
    }
    return ret;
}
```

动态定时器应用之schedule_timeout

kernel/timer.c

```
signed long __sched schedule_timeout(signed long timeout) {
    ...
    expire = timeout + jiffies;

    setup_timer_on_stack(&timer, process_timeout, (unsigned long)current);
    __mod_timer(&timer, expire);
    schedule();
    del_singleshot_timer_sync(&timer);

    /* Remove the timer from the object tracker */
    destroy_timer_on_stack(&timer);

    timeout = expire - jiffies;

out:
    return timeout < 0 ? 0 : timeout;
}
```

Outline

1. Linux的计时体系结构
2. 延迟函数
3. 相关API和命令
4. 小结和作业

延迟函数

- 常见手段：
  - 执行一些特殊指令来消耗一些时间
  - 执行一些循环来消耗时间
  - ...
- udelay(n), ndelay(n) @ include/asm-arm/delay.h
- __udelay等 @ arch/arm/lib/delay.S

Outline

与时钟和定时测量相关的API

- time() - get time in seconds
  - 返回从1970年1月1日凌晨0点开始的秒数

```
time_t time(time_t *t);
```

- ftime() - return date and time
  - 返回从1970年1月1日凌晨0点开始的秒数以及最后一秒的毫秒数

```
int ftime(struct timeb *tp);
```

```
struct timeb {
    time_t time;
    unsigned short millitm;
    short timezone;
    short dstflag;
};
```

与时钟和定时测量相关的API

- gettimeofday() , settimeofday() - get and set the time
  - 前者返回从1970年1月1日凌晨0点开始的秒数
  - 对应于sys_gettimeofday()

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
int settimeofday(const struct timeval *tv, const struct timezone *tz);
```

```
struct timeval {
    time_t tv_sec; /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
struct timezone {
    int tz_minuteswest; /* minutes west of Greenwich */
    int tz_dsttime; /* type of DST correction */
};
```

与时钟和定时测量相关的API

- getitimer(), setitimer() - get or set value of an interval timer
  - 每个进程有三个间隔定时器：
    1. ITIMER_REAL：real time
    2. ITIMER_VIRTUAL：user space time
    3. ITIMER_PROF：user + kernel space time
  - 频率：周期性的触发定时器（若为0，只触发一次）

```
int getitimer(int which, struct itimerval *curr_value);

int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value);
```

```
struct itimerval {
    struct timeval it_interval; /* next value */
    struct timeval it_value; /* current value */
};
struct timeval {
    time_t tv_sec; /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
```

与时钟和定时测量相关的API

- alarm() - set an alarm clock for delivery of a signal
  - 若干秒后引起SIGALARM信号

unsigned int alarm(unsigned int seconds);

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
static int flag=0;
void sig_alarm(int signo){
    flag=1;
}
int main(void){
    if (signal(SIGALRM, sig_alarm)==SIG_ERR){
        perror("Can't set new signal action");
        exit(1);
    }
    alarm(10);
    pause();

    if(flag) printf("SIGALRM received and flag changed!\n");
    return 0;
```

与时钟和定时测量相关的API

- asctime, ctime, gmtime, localtime, mktime, asctime_r, ctime_r, gmtime_r, localtime_r - transform date and time to broken-down time or ASCII
  - 改变时钟格式

与时钟和定时测量相关的命令

- date - print or set the system date and time
- time - - run programs and summarize system resource usage

Outline

小结

1. Linux的计时体系结构
   - ARM中的系统时钟system_timer
   - arm的滴答机制
   - Jiffies变量
   - Linux的时钟源
   - xtime变量
   - 时钟中断处理
   - 软定时器
2. 延迟函数
3. 相关API和命令
4. 小结和作业

作业

1. 名词解释
   1. 系统时钟
   2. 滴答和jiffies变量
   3. 动态定时器

Thanks !

The end.