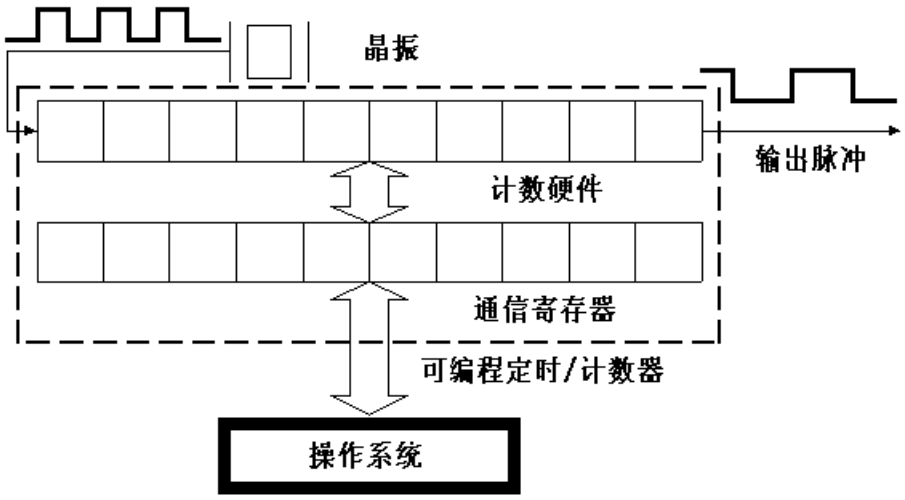
5.4 Linux时间系统

西安邮电大学

引言

“时钟中断”是特别重要的一个中断，因为整个操作系统的活动都受到它的激励。系统利用时钟中断维持系统时间、促使进程的切换，以保证所有进程共享CPU；利用时钟中断进行记帐、监督系统工作以及确定未来的调度优先级等工作。可以说，“时钟中断”是整个操作系统的脉搏。

基本时钟硬件



Linux的OS时钟的物理产生原因是可编程定时/计数器产生的输出脉冲，这个脉冲送入CPU，就可以引发一个中断请求信号，我们就把它叫做时钟中断。时钟中断的周期，也就是脉冲信号的周期，我们叫做“滴答”或“节拍”（tick）。从本质上说，时钟中断只是一个周期性的信号，完全是硬件行为，该信号触发CPU去执行一个中断服务程序。这是最简单的时钟硬件，在目前的系统中，还有更多相关的时钟硬件。

与时钟有关的硬件

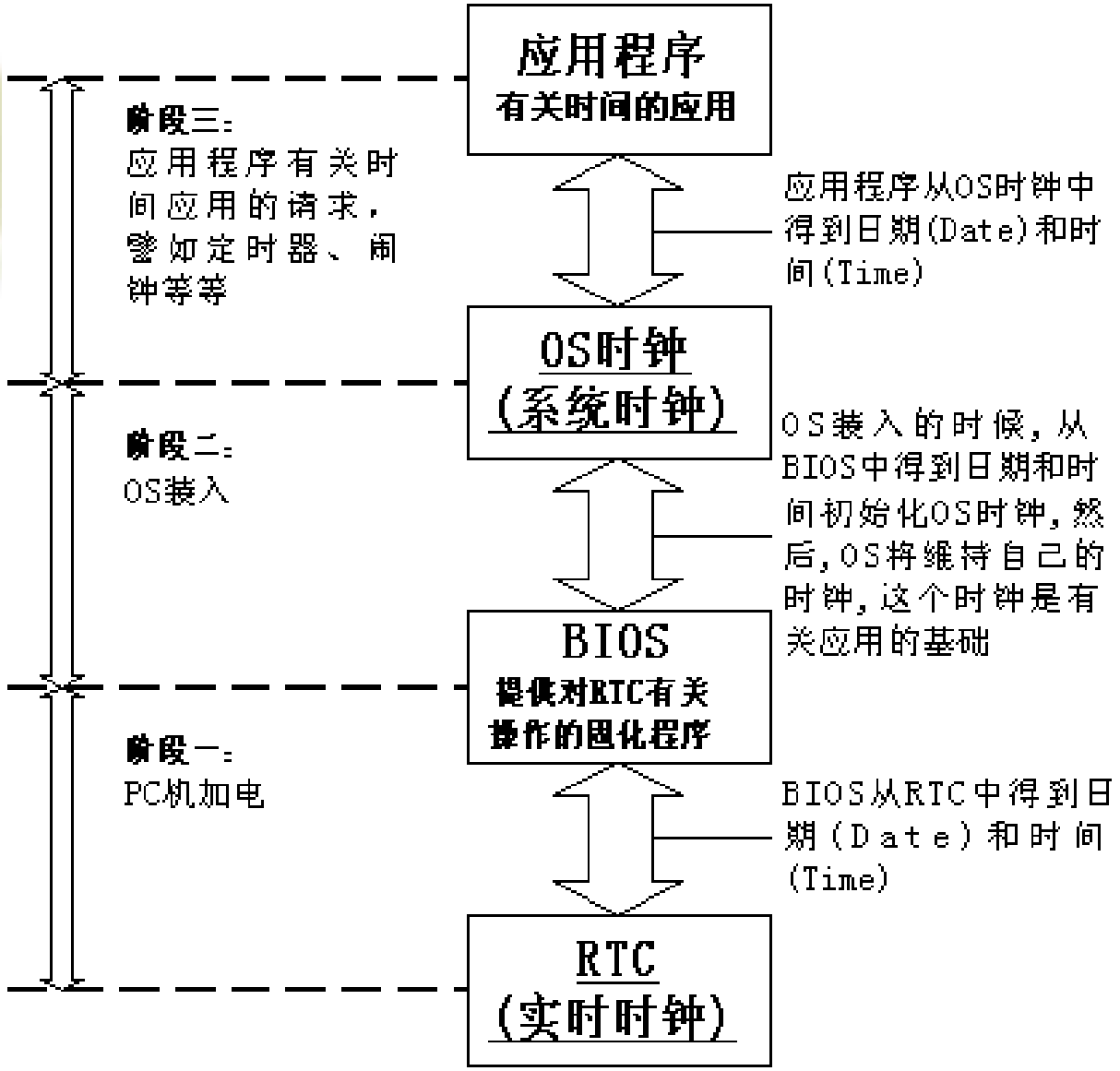
实时时钟RTC (Real Time Clock)	• 用于长时间存放系统时间的设备，即使关机后也可依靠主板CMOS电池继续保持系统的计时.
可编程间隔器 PIT(Programmable Interval Timer)	• 该设备可以周期性的发送一个时间中断信号。在Linux系统中，该中断时间间隔由HZ表示，这个时间间隔也被称为一个节拍(tick).
时间戳计数器 TSC(Time Stamp Clock)	• CPU附带了一个64位的时间戳寄存器，当时钟信号到来的时候，该寄存器内容自动加1

与时钟有关的硬件

高精度计时器(HPET)	• 这是一种由intel开发的新型定时芯片。该设备有一组寄时器,每个寄时器对应有自己的时钟信号,时钟信号到来的时候就会自动加1。
CPU本地定时器	• 在处理器的本地APIC提供的一个定时设备。 • 可以单次或者周期性的产生中断信号。
高精度定时器（hrtimer）	• 提供纳秒级的定时精度，以满足对精确时间有迫切需求的应用程序或内核驱动，例如多媒体应用，音频设备的驱动程序等等

在这6个相关硬件中，其中有2个计时器（counter），有2个定时器（timer），每个都有其特定的功能。

基本时钟运作机制


应用程序
有关时间的应用
阶段三：
应用程序有关时间应用的请求，譬如定时器、闹钟等等
应用程序从OS时钟中得到日期(Date)和时间(Time)
OS时钟
(系统时钟)
阶段二：
OS装入
OS装入的时候，从BIOS中得到日期和时间初始化OS时钟，然后，OS将维持自己的时钟，这个时钟是有关应用的基础
BIOS
提供对RTC有关操作的固化程序
阶段一：
PC机加电
BIOS从RTC中得到日期(Date)和时间(Time)
RTC
(实时时钟)

基本时钟运作机制

- 一般来说，RTC是OS时钟的时间基准，操作系统通过读取RTC来初始化OS时钟，此后二者保持同步运行，共同维持着系统时间。所谓同步，是指操作系统在运行过程中，每隔一个固定时间会刷新或校正RTC中的信息。
- Linux中的时钟运作机制如图所示。OS时钟和RTC之间要通过BIOS的连接，是因为传统PC机的BIOS中固化有对RTC进行有关操作的函数，通常操作系统也直接利用这些函数对RTC进行操作，例如从RTC中读出有关数据对OS时钟初始化、对RTC进行更新等等。Linux在内核初始化完成后就完全抛弃了BIOS中的程序。
- RTC处于最底层，提供最原始的时钟数据。OS时钟建立在RTC之上，初始化完成后将完全由操作系统控制，和RTC脱离关系。操作系统通过OS时钟提供给应用程序所有和时间有关的服务。因为OS时钟完全是一个软件问题，其所能表达的时间由操作系统的设计者决定，将OS时钟定义为整型还是长整型或者大的超乎想象都是由设计者决定。

Linux时间系统

•tick（节拍）	•节拍率(HZ)是时钟中断的频率，
•jiffies	•jiffies用来记录自系统启动以来产生的总节拍数
•xtime	• xtime和RTC时间一样，都是人们日常所使用的墙上时间。

- 操作系统的“时间基准” 由设计者决定，Linux的时间基准是1970年1月1日凌晨0点
- OS时钟记录的时间就是系统时间。系统时间以“时钟节拍”为单位，时钟节拍（tick）表示发生一次中断的时间，比如1ms。
- Linux中用全局变量jiffies表示系统自启动以来的时钟节拍数目，比如系统启动了 N 秒，那么 jiffies就为 N×HZ
- 实际时间存放在内核的xtime中，系统启动时内核通过读取RTC来初始化实际时间

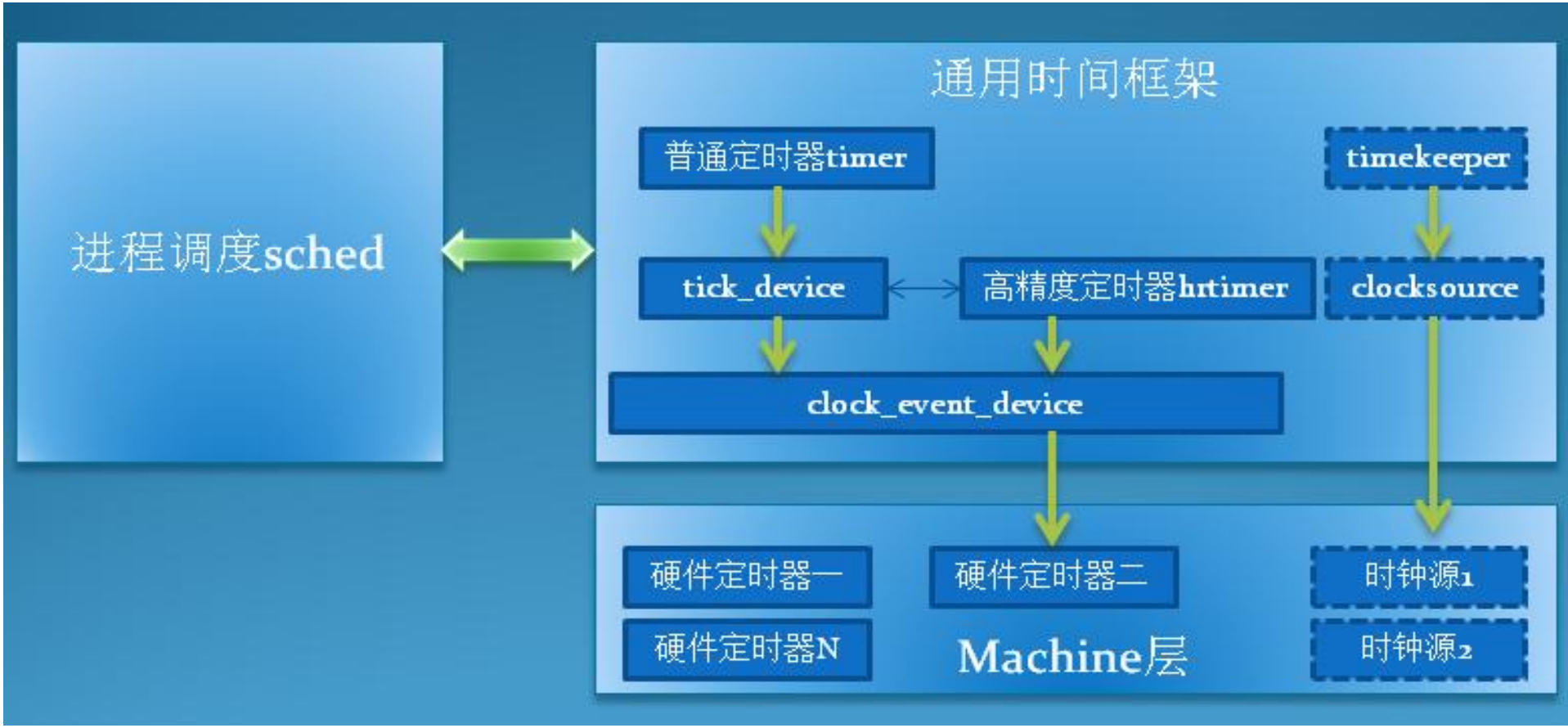
Linux时钟框架

Linux时钟框架相当复杂，在此给予简述

如果你输入data命令，内核从哪个部件获取时间？从时钟源clock source！从硬件层来说，它其实就是固定时钟频率驱动的计数器，计数器只能单调地增加，直到溢出为止。

对于真实的用户来说，我们感知的是真实世界的真实时间，也就是所谓的墙上时间，clocksource只能提供一个按给定频率不停递增的周期计数，如何把它和真实的墙上时间相关联？这就是timekeeper。

Linux时钟框架


进程调度sched
通用时间框架
普通定时器timer
timekeeper
tick_device
高精度定时器hrtimer
clocksource
clock_event_device
硬件定时器一
硬件定时器二
时钟源1
硬件定时器N
Machine层
时钟源2

Linux时钟框架

那么时钟事件设备(clock_event_device)起什么作用呢？它主要用于实现普通定时器和高精度定时器 ，同时也用于产生节拍tick事件，供给进程调度子系统使用。

在软件架构上看，时钟事件设备（clock_event_device）被分为了两层，与硬件相关的被放在了machine层，而与硬件无关的通用代码则被集中到了通用时间框架层，这符合内核对软件的设计需求，平台的开发者只需实现平台相关的接口即可，无需关注复杂的上层时间框架。

tick_device起什么作用？它是基于时钟事件设备的进一步封装，用于代替原有的时钟滴答中断，给内核提供节拍事件，以完成进程的调度和进程信息统计，负载平衡和时间更新等操作。

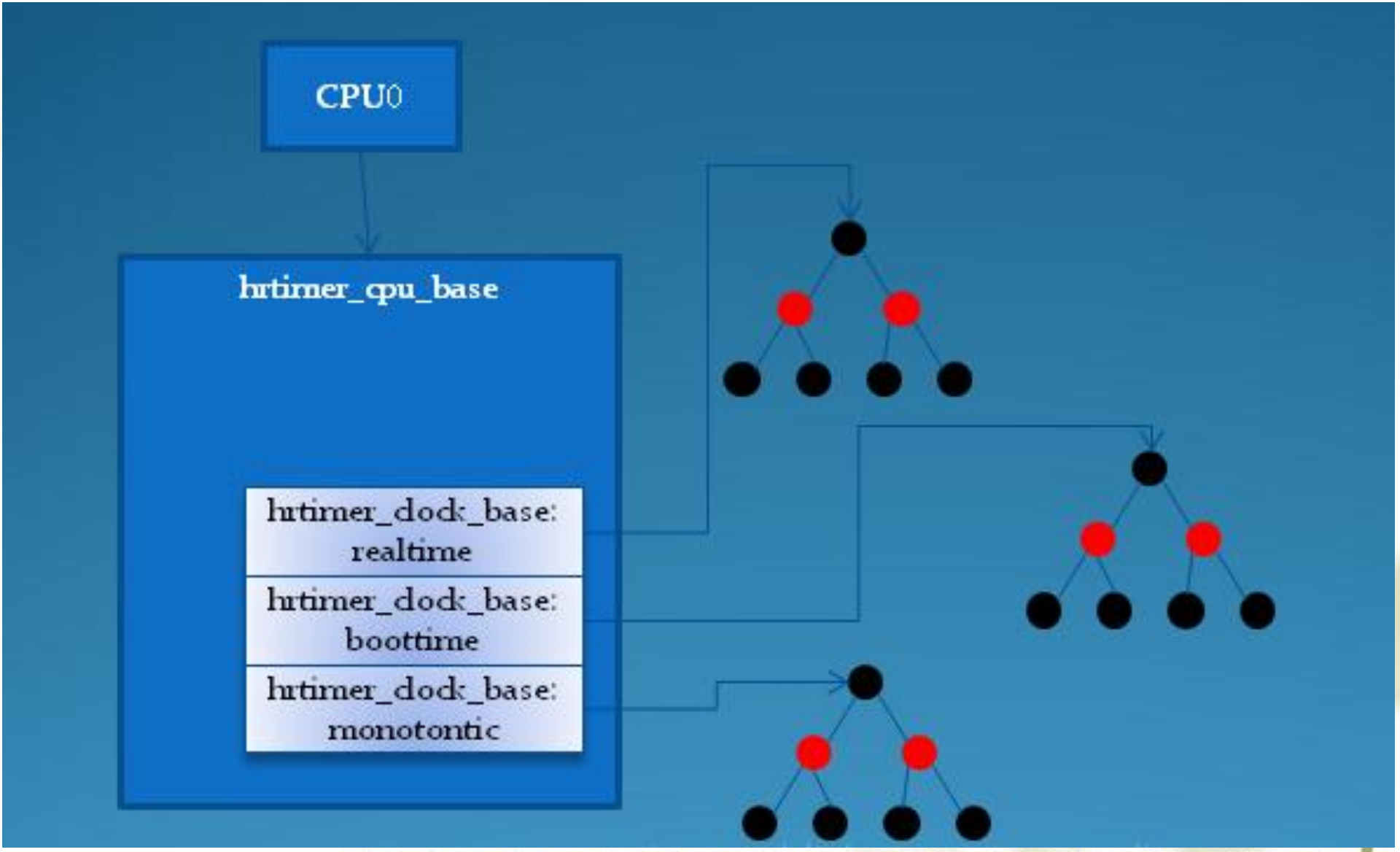
高精度定时器实现机制

高分辨率定时器的代码实现时，内核的开发者考察了多种数据结构，例如基数树、哈希表等等，最终选择了红黑树（rbtree）来组织hrtimer。hrtimer不停地被创建和销毁，新的hrtimer按顺序被插入到红黑树中，树的最左边的节点就是最快到期的定时器。

每个cpu有一个hrtimer_cpu_base结构；这个结构管理着3种不同的时间基准系统的hrtimer，分别是：实时时间，启动时间和单调时间；

每种时间基准系统通过它的一个字段，指向它们各自的红黑树；

高精度定时器实现机制


CPU0
hrtimer_cpu_base
hrtimer_clock_base:
realtime
hrtimer_clock_base:
boottime
hrtimer_clock_base:
monotontic

定时器及应用

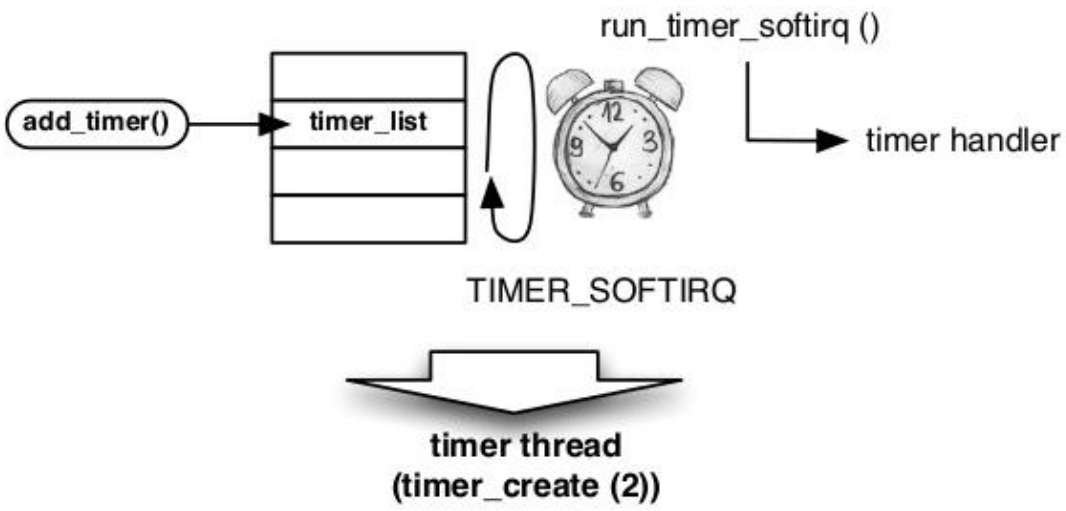
```
struct timer_list {
    struct list_head  entry;
    unsigned long expires;
    unsigned long data;
    void
(*function)(unsigned   long);
    unsigned long data;
};
```

- 定时器是管理内核所花时间的基础，也被称为动态定时器或内核定时器
- 定时器的使用步骤：执行一些初始化工作，设置一个到期时间，指定到时后执行的函数，然后激活定时器就可以了
- 定时器由timer_list结构表示
  （这句话讲完后出现左边的代码）

定时器的使用

- 定义定时器：
- struct timer_list my_timer;
- 初始化定时器，填充定时器的结构
- init_timer(&my_timer);
- 激活定时器：add_timer(&my_timer);
- 如果需要修改修改定时器，则调用 mod_timer(&my_timer)
- 如果需要在定时器到期前停止定时器，可以使用 del_timer(&my_timer)函数：

定时器的执行与应用



- 内核在时钟中断发生后执行定时器，定时器作为软中断在下半部中被run_timer_softirq( )执行
- 定时器的使用：
- 例：创建和使用进程延时

定时器的应用举例

```
timeout = 2 * HZ; /*1HZ等于100，因此为2000ms*/
set_current_state(TASK_INTERRUPTIBLE);
remaining = schedule_timeout(timeout);
```

内核用定时器实现进程的延时，调用schedule_timeout( )函数，该函数的主要代码如下：

```
unsigned schedule_timeout(unsigned long timeout)
   {
         struct timer_list timer;
        unsigned long expire;

      expire = timeout + jiffies;
      init_timer(&timer);
      timer.expires = expire;
      timer.data = (unsigned long) current;
      timer.function = process_timeout;

      add_timer(&timer);
        schedule( );      /* 进程被挂起直到定时器到期 */
         del_timer(&timer);

      timeout = expire - jiffies;
          return (timeout < 0 ? 0 : timeout);
  };
```

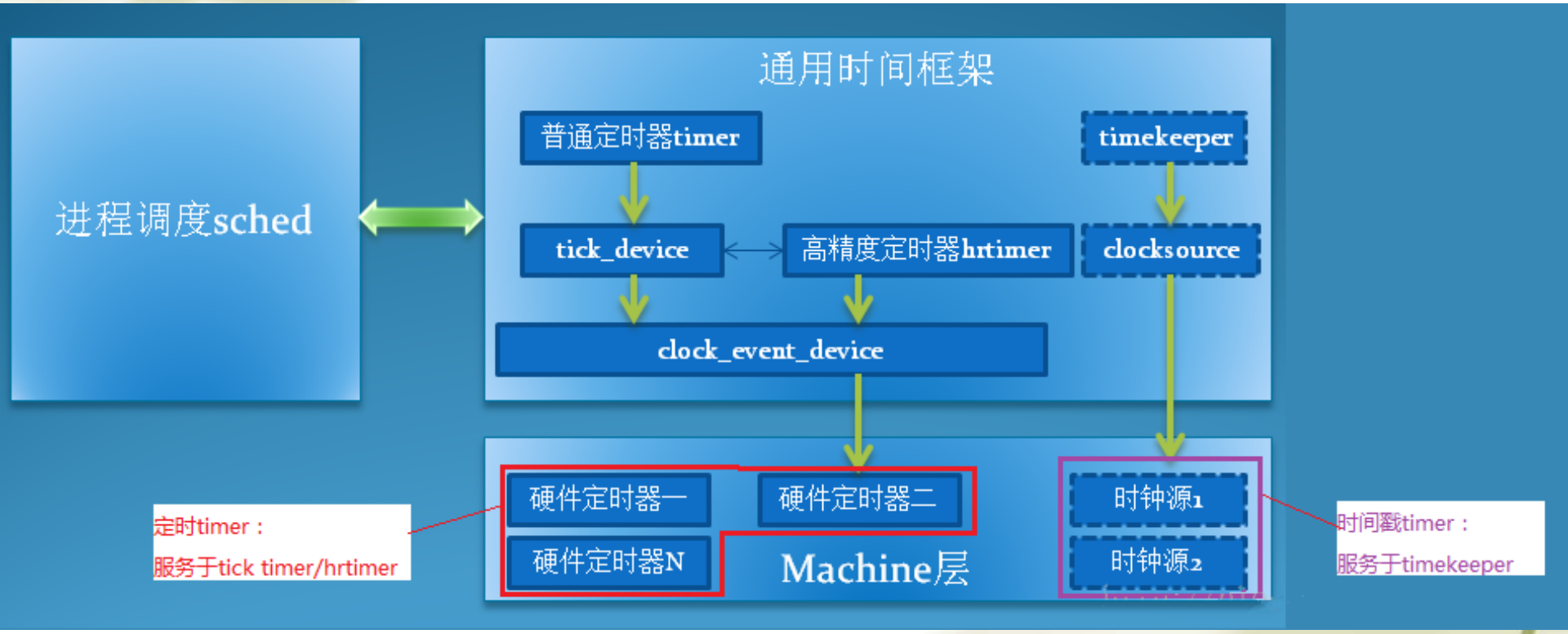
定时器的应用举例

- 首先设置进程的状态为不可中断的睡眠。
- 然后编写 内核用定时器实现进程的延时。其中，首先对定时器的各个字段赋初值，然后通过激活函数add_timer（）激活定时器，再调用调度函数schedule（ ）让该进程挂起，最后调度删除函数del_timer（）删除调度器。具体实现的代码参加内核之旅http://www.kerneltravel.net/?p=1047

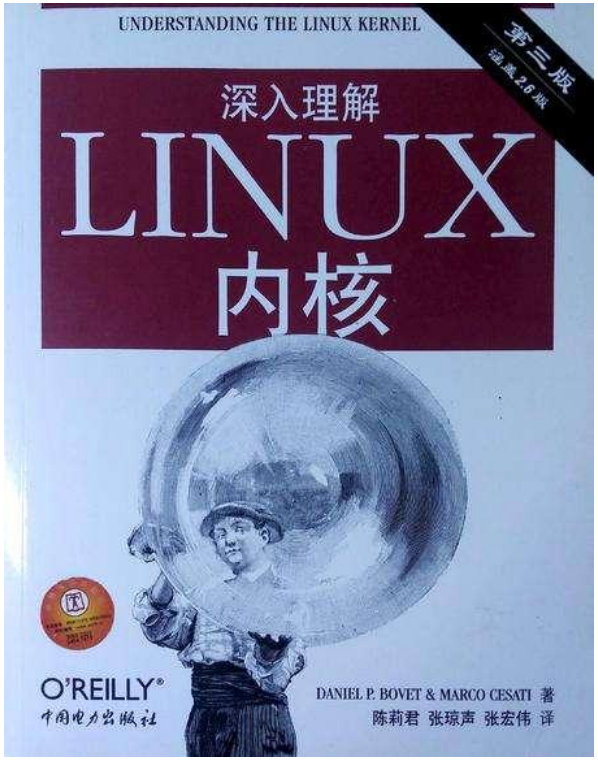
Linux时钟子系统总结

随着应用环境的改变，使用需求的多样化，Linux 的时钟子系统也在不断的衍变。为了更好的支持音视频等对时间精度高的应用，Linux 提出了 hrtimer 这一高精度的时钟子系统，为了节约能源，Linux 改变了长久以来一直使用的基于 HZ 的 tick 机制，采用了 tickless 系统。即使是在对硬件平台的支持上，也是在不断改进。Linux 定时器机制比较复杂，本讲只是做一个概要介绍。

Linux时钟子系统小结



参考资料



深入理解Linux内核 第三版第六章

带着思考离开

系统有了低精度定时器，为什么还要有高精度定时器？二者之间兼容么，在实现上有关系么？

“内核之旅”网站

- Linux 内核之旅网站http://www.kerneltravel.net/
- 电子杂志栏目是关于内核研究和学习的资料
- 第八期“中断”，将向读者依次解释中断概念，解析Linux中的中断实现机理以及Linux下中断如何被使用。

谢谢大家！

THANK YOU