openEuler内核编程

课程讲稿

第6章 第1讲

《内核时钟基础》

软件所制

第6章 第1讲 内核时钟基础

**学时：**1学时

**教学目的：对内核的时钟产生基本的概念，了解HZ（节拍率）与jiffies在内核中的作用。深入理解jiffies回绕问题，通过Linux解决方案来学习Linux的设计理念。对定时器、时钟中断产生基本的认识，为后面深入的学习做准备。**

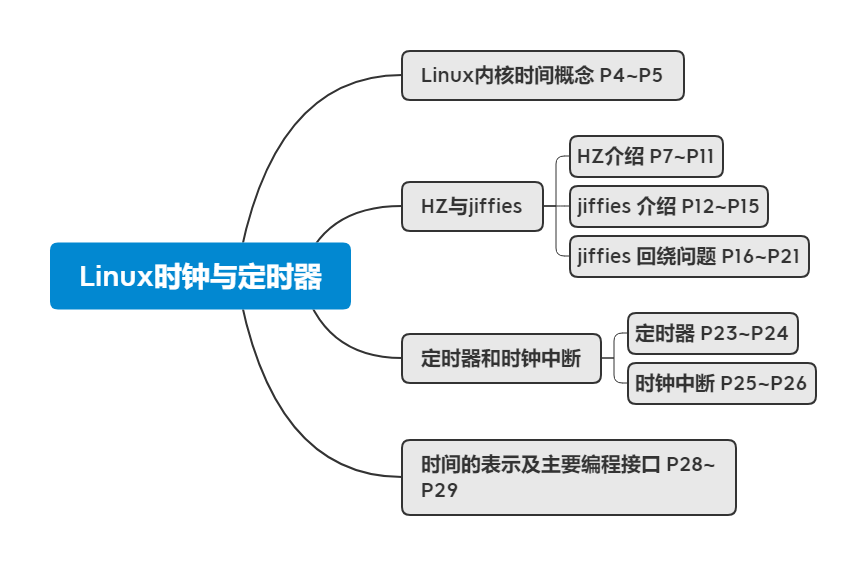
**课程时间线：**



**课外参考读物：**

https://www.ibm.com/developerworks/cn/linux/l-cn-linuxkernelint/index.html

**知识框图：**

****

**PPT讲稿：**

这一章我们来讲时钟。

1. **章节概略**

本章会从三个方面来进行讲解，第一是内核时钟基础，然后是定时器，最后是内核实时性的分析。



本节也从三方面来展开。先讲节拍率和jiffies，这是Linux内核的计时基础，然后是定时器以及时钟中断。最后简单说一下和时间相关的编程接口。

绝大多数计算机都是通过时间来驱动的，所谓时间驱动，就是我计算及内部的运算最底层都是以时间片为基本单位。比如显示器刷新或者计划任务等等。Linux内核除了管理系统运行的时间外，还要管理当前的实际日期和时间，当然这一切都需要借助硬件的帮助才能实现。

RTC,即实时时钟芯片，用来在系统断电时，利用备用的锂电池继续记录时间。这里记录的是真实时间，即年月日，时分秒。RTC是SoC中一个内部外设，RTC有自己独立的晶振提供RTC时钟源，内部有一些寄存器用来记录时间（年月日时分秒星期）。一般情况下为了在系统关机时时间仍然在走，还会给RTC提供一个电池供电。这一个时间存放在xtime变量中。

Linux系统还提供了了一种周期性触发中断的机制。x86中采用可编程中断时钟（PIT）实现，PIT的作用类似于烤箱的闹钟，PIT永远以内核确定的固定频率发出中断，但频率不算高。



下面我们再来了解一下内核时钟的基本概念，先看HZ和jiffies



HZ就是每秒钟产生中断的次数，也叫节拍率，在系统初始化的时候，会按照设定来对硬件进行设置。那么这个参数设定在什么地方呢，在内核文件的asm/param.h中。在x86、Arm的体系结构中，这个数值默认都是100，也就是说每秒钟要产生100次中断，每一次中断持续10ms。节拍率并不是一成不变的，都是可调的，同时不同的架构下甚至不同的硬件下，默认的节拍率也不一样。



下面这张表我们可以更清楚的看到不同体系结构下默认的节拍率。基本上都是100，个别的架构会有差别。



那么节拍率的大小究竟设置为多大合适呢？默认的就是最好的么？这里我们要提到一个概念，粒度。也就是时钟中断解析度。节拍率越大，单位时间中断的次数越多，我们就说时钟中断的解析度越高，粒度越细。提高节拍率意味着我们系统的中断精度更大了，提高了事件驱动时间的准确率。比如默认的100HZ，时间会在正负5ms内发生，而1000HZ则会在正负0.5ms内发生。这样看起来似乎HZ越高越好，毕竟谁不希望自己的程序运行的更精确呢？但高HZ真的没有什么劣势么？我们往下看。



首先，高HZ的优势很明显，首先就是程序运行的准确度提高了，同时带来的好处就是进程抢占会有更高的准确度。比如HZ等于100的情况下，A进程的时间片只剩下2ms，程度调度抢占，最差要10ms后才能抢占成功。而HZ等于1000的情况下，最坏是1ms。

另一方面，频繁的中断意味着系统的负担加重，这是高HZ的劣势。



实验表明，对于一般系统程序来说，1000HZ和100HZ的效果都差不多，即使将HZ提高到1000，在目前的硬件水平上来看，也不会对系统造成多大的影响。



接下来再看下jiffies。Linux系统中用一个交jiffies的全局变量来记录从启动以来所产生的节拍数。这个变量初始化为0。时钟每中断一次这个值便加一，假如HZ等于100，那么每秒钟jiffies的值便加100。通过jiffies这个变量和节拍率HZ，我们就可以计算得出系统自启动以来的运行时间，jiffies除以HZ。同样三者也可以互相转化。jiffies的定义在内核的linux/jiffies.h中。



利用jiffies和HZ变量，我们可以做很多事情。这里列举了四个简单地实例。第一个直接复制，可以看做当前的时间戳。要注意这里的时间戳以节拍为单位，同时以系统启动为起始点。一般内核本身很少用到绝对时间。第二个就是在当前的下一个节拍。第三个是从现在节拍开始后的5秒。第四个是现在开始的十分之一秒。结合这两个变量，发挥你们的想象，可以有很多的用法。



我们看一下jiffies在计算机中是如何表示的。他是一个unsigned long。在32位体系下，节拍率为100，那么大约497天后溢出。节拍率增加10倍，49.7天会溢出。那么为什么是497呢？我们知道32位最长可以表示的数字是4294967296，这个数除以节拍率就是时间，也就是42949673秒，换算成日就是497.1天。所以这个数字就是这么来的。那么到了64位体系下，最长可表示的数字就太长了，具体能表示多少时间大家可以自己换算一下，算出来一定把你吓一跳。



虽然一般我们个人电脑很少会开机497天，就算是节拍率1000，最多49.7天也很少会超过。但是在工业以及商业领域。别说497天，一个服务器开着几年不关都是有可能的。那么我们的操作系统，这里说的32位操作系统，就必须要对这个问题作出回应，不能因为这个问题把系统搞崩溃掉。在Linux 2.6版本中，引入了jiffies\_64这样一个无符号整型，长度为64位。系统为了兼容性考虑，仍然保留了32位的jiffies变量，直接就是指向jiffies\_64的低32位。jiffies一般应用也足够用了。同时如果想获取jiffies\_64，系统也准备了get\_jiffies\_64()这样一个函数来取数。



下面我们来说回绕问题。在32位的机器上，jiffies作为一个无符号的长整型，有着最大值，就是2的32次方减1，我们假设系统运行到这个边界值，再往下运行会怎么样呢。显然jiffies不会增加到2的32次方，这是自然的想法，在计算机中，所有的位都会归零，所以jiffies的值就变为了0。所以从操作系统的或者用户软件的角度来看，操作系统才刚刚启动，一切才刚刚开始，这显然是不合理的。那么在运行软件的过程中就可能会造成错误，我们看下面这个例子。

程序先定义了一个超时变量timeout，这里的意思是给了程序0.5s的运行时间。不要觉得半秒的时间很短，对于软件来说，已经能做很多事情了。那么软件运行完之后，在程序的末尾加了一个判别语句，用timeout和当前的jiffies进行比较。如果当前的jiffies比预设的timeout要小，就是没超时，否则就是超时了。这个判断语句在平常肯定是没问题对的，现在我们看一下如果遇到jiffies回绕会发生什么。假设一开始的设定timeout的时候jiffies就已经接近2^32-1，假设为2^32-10，timeout增加50以后，timeout这个变量就会发生回绕，变为39。继续假设程序运行5个节拍。这个时候jiffies的值就变为了2^32-5。此时对两者进行对比。会发现进入了else语句，但程序确实没有超时，只运行了5个节拍，那么这里程序就会运行出错。



为了规避这个问题，Linux定义了四个宏。其实非常简单，就是把unsigned long转换为了long，就解决这个问题了。这里的unknown一般是当前的jiffies，而known是要对比的值。



我们举例说明，32位的jiffies太长，我们用8位的char来代替，原理都是一样的。先看无符号的char，取值范围是0~255，J1时刻的值是250，设定两个节拍后超时，那么timeout就是252。程序运行了7个节拍后停止在J2时刻，正常来讲应该是250+7=257。但是char的最大值就是255，于是产生了回绕，回绕为1。这个时候拿1和252来比，会得出程序没有超时的结论，但实际上程序运行了7个节拍，远远超过了两个节拍的超时时间。错误。

再来看以8位有符号char的情况。还是原始的二进制值，直接转换为有符号char，J1就是-6。J2还是1。timeout变为了-4。这时发现J2>timeout，得出结论程序超时，符合预期，结果正确。



在实际使用的时候直接用上一页的几个函数来替换之前的判别语句就可以了。



我们再来思考另一个问题，刚才的例子是无符号转换为有符号发生在边界处。还有一种情况是发生在中部，刚才的例子中就是127附近，这种情况也成立吗？



这两张图很清晰的说明了这个问题。j1，timeout都在前半部分，j2存在于后半部分，当强制转换后，j2被颠倒到了开头的地方。现在看到(long)timeout - (long)j2的值其实就是图中x2的值，只要第一张图片中的x1小于2^31-1，x2就大于2^31-1。大于2^31-1的数，最高的比特位（bit31）必定为1，由于计算是按long类型计算的，所以bit31上的1被当做负号处理，(long)timeout - (long)j2的结果其实是一个负数，当然小于0。因此正数减去负数也会小于0。好了，现在说说使用范围，为了使(long)timeout - (long)j2小于零，必须使x1小于2^31-1。也就是说只要timeout和j2之差绝对值不超过2^31-1，time\_after这组宏就不会出问题。这组宏其实是牺牲了范围换取正确性。



下面我们来看一下定时器。



定时器是Linux内核中非常重要的一个概念。定时器是管理内核流逝时间的基础。在x86体系结构中，系统定时器主要采用可编程中断时钟（PIT）实现。



定时器定义在linux/timer.h文件中。这个结构体大家不用记，有个印象就可以了，内核部分提供了相应的API来简化定时器的操作。



时钟中断是一种硬中断，由时间硬件产生，CPU处理后交由时间中断处理程序来完成更新系统时间、执行周期性任务等。Linux时间中断处理程序分种两部分：体系结构相关部分与体系结构无关部分。体系结构相关部分被注册到内核中，确保中断产生时能执行，这部分不能有耗时操作，主要是更新时间与调用结构无关部分列程。已到期的定时器由体系结构无关部分来处理，其它的一些耗时操作，如显示时间的更新也在这一部分。

Linux系统在时钟中断周期会进行如下操作。更新系统运行时间，更新实际时间，检查当前时间片是否用尽，用尽的话则重新调度。运行超时的动态定时器，更新资源消耗以及处理器时间的统计值。



时钟中断处理程序作为系统定时器而注册到内核中，体系结构的不同，可能时钟中断处理程序中处理的内容不同。一般来说会进行以下的一些具体操作。大家了解一下就可以了。



这节课最后我们再来了解一下和时间相关的主要API。



实际时间，也就是常说的墙上时间，以秒为单位。记录着从1970年1月1日凌晨以来所经历的时间。也就是我们常说的时间戳的概念。这个数据存放在xtime变量中。这个变量是一个系统定义的结构体，在linux/time.h文件中。这个结构体分为两个部分，一部分是秒为单位的，一部分是纳秒为单位的，代表着两种不同的精度。这里要注意，读取xtime变量需要使用的xtime\_lock锁，它是一个seq锁。



这里是一些常用的编程接口，大家要有所了解，知道系统有这些功能，要有印象。具体编程的时候需要的时候再去查文档就可以了。



略