# 多线程应用中的定时器管理设计与实现

一、基础知识

**2.2 通知链技术（ notification chain ）**

在时钟处理这部分中，内核用到了所谓的“通知链（ notification chain ）”技术。所以在介绍时钟处理过程之前先来了解下“通知链”技术。

在 Linux 内核中，各个子系统之间有很强的相互关系，一些被一个子系统生成或者被探测到的事件，很可能是另一个或者多个子系统感兴趣的，也就是说这个事件的获取者必须能够通知所有对该事件感兴趣的子系统，并且还需要这种通知机制具有一定的通用性。基于这些， Linux 内核引入了“通知链”技术。

**2.2.1 数据结构：**

通知链有四种类型，

1.原子定时器：定时器的回调函数（当事件发生时要执行的函数）只能在中断上下文中运行，不允许阻塞。

2.可阻塞定时器：定时器的回调函数在进程上下文中运行，允许阻塞。

3.原始定时器（ Raw notifier chains ）：对原始定时器的回调函数没有任何限制，所有锁和保护机制都由调用者维护。

4.SRCU 定时器（ SRCU notifier chains ）：可阻塞定时器的一种变体

5. 循环使用的定时器：定时器需要多次被循环调用。

二、定时器管理算法

1. 数据结构：

(1) CTimeWheel：时间轮，实际上是一个循环数组，长度可定为应用所需的定时器的最大数量(maxTimers)，每个数组元素是一个链表。

(2) timer\_node：定时器节点，该节点结构应包括的属性有：时间片号(slotNum)，时间轮圈数(round)。

(3) addTimer：根据定时器的超时时间(interval)和时间片的时长(timeslot)相除，得到该定时器超时所需的时间片总数 (ticks)。根据该定时器所需的时间片总数(ticks)和当前的时间片号，得出该定时器在时间轮中应处的时间片号，把该定时器插入该时间片上的定时器链表中，同时计算时间轮圈数。

ticks = interval / timeslot;

slotNum = (currentSlot + (ticks % maxTimers)) % maxTimers;

round = ticks / maxTimers;

(4)timerTick：只要判断当前 slotNum上的节点或者链表 中的 round 是否为 0，就可以知道是否超时。时间复杂度为 O(1)。

2. 算法思想：



**图1 currentSlot=0 时，定时器 A,B的添加**

把循环数组当成一个表盘，时间槽指针currentSlot当成表盘上的时钟，每一个时间槽为指向一个带头结点的链表的指针。图1中，时间槽指针currentSlot指向0时，添加定时器 A,B。时钟每隔100ms跳一格，当定时器管理器时针将要指向下一个 slot 时，如果下一个slot的链表结点的圈数round等于0，则表示该节点上的计时器超时，触发超时事件，并删除该计时器。之后，我们还需要将链表上所有的节点的圈数round减一。图2中，时间槽指针currentSlot指向1时，表盘的状态：触发定时器A，之后删除它。定时器B的round=1，当时间槽指针currentSlot第一次指向6时，定时器B的round值将减1，但此时并不触发定时器B的事件，只有当定时器再次指向时间槽6才触发。

**图2 currentSlot=1 时，定时器A的触发，以及定时器 C的添加**

对于循环使用的定时器，需作特殊处理。在删除该定时器之后，必须重新找到相应的位置，再插入该循环定时器。

三、定时器管理实现