针对非实时操作系统的实时性优化

作者：张雷 院系：计算学部

摘要：随着信息科技的飞速进步和嵌入式技术快速普及的今天，实时操作系统在嵌入式方向具有广阔的应用前景。但是对于很多现有的嵌入式操作系统，其实时性不能满足使用的要求。所以对于提高操作系统的实时性是一个重要的话题。

本文主题从针对非实时操作系统的实时性优化出发，参考三篇相关论文，提出操作系统实时性优化的策略。首先可以降低系统噪声来提高某些控制进程的实时性，系统噪声通常是直接来自于调度和中断；然后从调度的角度入手，分析现有的调度算法，并进行实时性改进；最后针对Linux2.6内核中大量的自旋锁改为互斥锁，而自然地提高实时性。并调整中断任务为内核线程，这样一来，实时任务就可以作为较高优先级的任务执行了。

关键词：系统噪声；实时性；锁；内核线程

# 一、绪论

如今，许多操作系统的实时性要求越来越高。操作系统经历了几个阶段的发展，比如说嵌入式操作系统从最初以单芯片为核心的可编程控制器形式的系统，到现在以嵌入式操作系统和微处理器为标志的嵌入式系统。在很多的应用中，嵌入式系统必须是一个实时系统，即系统的正确性不仅仅依赖于计算的逻辑结果，而且依赖于结果产生的时间。实时系统的核心就是操作系统必须能够在一个实现定义好的时间限制内对内部和外部的事件进行响应和处理。

不仅仅是嵌入式操作系统，我们平时使用的个人PC，以及服务器操作系统也有实时性要求。因为平时与用户交互的桌面程序，以及后台服务器都有着严格的实时性要求。如果不能满足的话，将严重影响用户体验。

本文将参考针对嵌入式操作系统的实时性优化，来提出几种对于个人电脑和服务器操作系统进行实时性优化的方法。

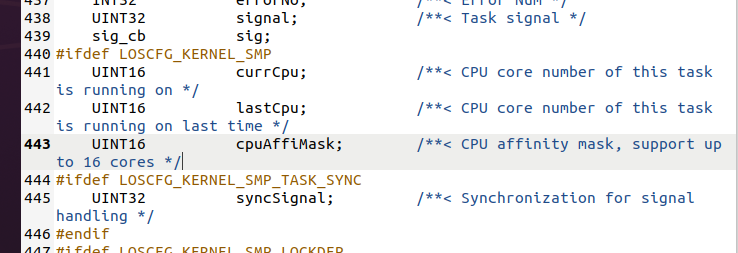
# 二、降低系统噪声

## （一）背景

无论是在个人PC电脑中还是服务器操作系统中，都有一些任务需要良好的实时性。以EAST PCS系统为例，控制进程的实时性与其工作结果有着较大的关联，控制进程的实时性与进程抢占与中断相应的实时性关系很小，关键在于控制进程能否不被打扰的运行。系统完全不对进程进行干扰是不可能的。这里我们可以将EAST PCB系统的控制进程类比于我们PC机中比较重要且特殊的系统进程（实时性要求较高），它们在运行过程中经常被干扰，每一次干扰都会对进程带来延迟。可以通过减少这些延时来进行实时性优化。

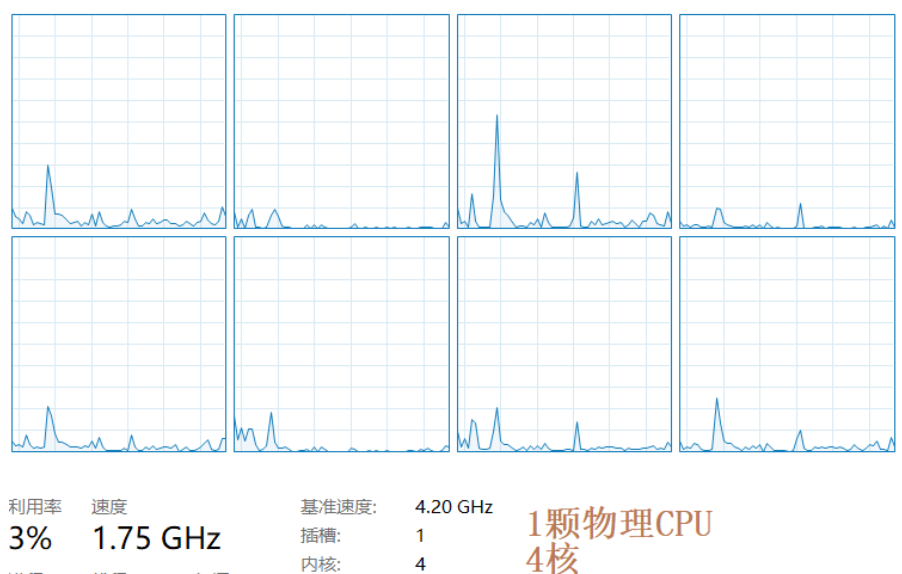
## （二）系统噪声改善方法

首先想到的是可以手动设置相应进程的PCB，使其与一个内核强关联。也就是说，我们可能需要实现对CPU内部各个核心进行统计，找到一段时间内中断频率较低的内核，然后将该进程切换到我们选取的内核中工作。这样就可以一定程度上减轻实时性进程受系统噪声影响的延时了。



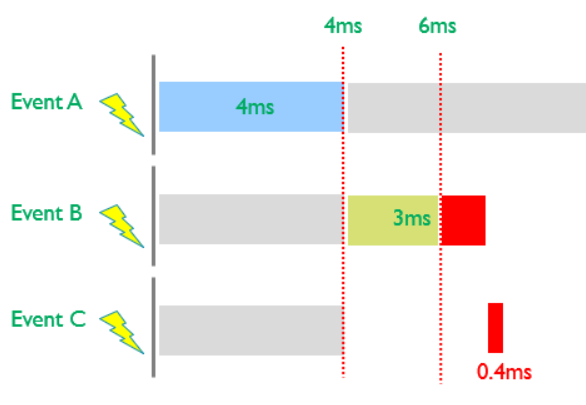
但是在EAST PCS系统介绍的论文中，中断引起的噪声（中断噪声）相比无关进程的意外运行给实时性进程造成的影响是微弱的，也就是说我们需要将精力放在其他无关任务的插入执行上。因此我们需要从其他任务的插入运行的角度来减少系统噪声。我们称这一噪声为进程噪声。

进程噪声可以分为两种类型：  
 1）内核为了保证各个核心负载均衡，会将进程分配到各个核上。（Linux有着负载均衡模块，一个任务进入系统，都要经过负载均衡模块添加到一个核心的调度队列上去）



2）内核可能在各个核上维持一些守护进程，在一定条件下会唤醒守护进程执行日常事务活动。

这两种任务都可能使实时性控制进程发生上下文切换，从而存在延迟。如果我们在内核中使用单一的调度算法，那么控制进程始终可能被抢占（或者就是给控制进程设定足够高的优先级，但是在一种调度策略下，控制进程就会一直占用处理器内核，而用户进程得不到cpu资源而造成饥饿）。



所以我们需要保证，控制进程（实时性）所在处理器没有用户进程，并且控制进程所在的处理器上的守护进程不会被唤醒。为了实现这两点我们可以在用户进程调度到控制进程所在的处理器之后，调整到其他核心上去。最好是将控制进程固定在特定的几个处理器核上，守护进程全部分配到其他处理器上。这样可以解决进程噪声的问题。由此，实时性要求较高的一些任务就可以在特定的几个核上运行而不被其他任务干扰。

# 三、调度算法的实时性改进

良好的调度算法可以使任务的响应更好这是毋庸置疑的，调度算法的优化包括硬实时调度优化和软实时调度优化。

## （一）硬实时调度优化

硬实时调度优化就是我们事先就设定好调度算法，之后在系统调度过程中就遵循既定的系统调度算法进行任务调度。

如果上层任务有明确的调度时机，那么在算法设计的时候就可以明确指出各个进程的调度时机，而且需要合理的安排各个阶段所需要的时间使得实时性高的任务响应的快。例如RED-Linux算法。



针对不明确调度时机但是有时限的上层任务，可以使用基于优先级的调度算法。例如我们在实验二中实现的EDF算法。按照正常的情况，内核代码中的调度类（一个调度策略）的任务就是为系统提供可执行的任务。在优先级高的调度任务完成之后，优先级低的才能操作。EDF算法本身就保证了实时性要求较高的任务（也就是执行截止时间往往较小的任务）率先执行，于是这些任务相比之下更多的获得CPU资源。

针对调度时间不明确，且不可预测操作时间的上层任务，可以根据资源预留的思想设计此问题的答案。但是这样做的弊端就是这可能是现有CPU资源的浪费，不过如果系统中实时性任务较少的情况下，这样的操作是可行的。

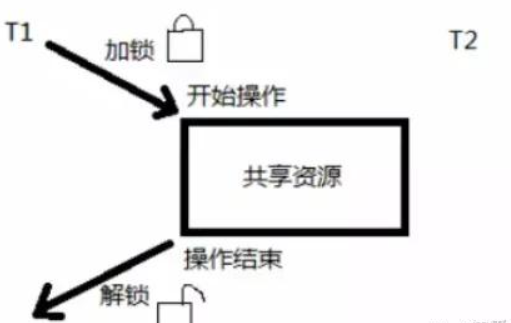
## （二）软实时调度优化

上述所提到的硬实时调度算法不可能在所有场景都能够适用，有时候一些软实时的场景就不能使用硬实时调度算法，可以采用基于比例共享的资源分配思想。也就是说依据任务所分配的权重来分享CPU（实时性任务获取的CPU资源单元较大），当系统过载时，所有的任务也会依据相应的权重缓慢操作。但是为了使系统中比较重要的任务及时执行，需要采取动态调节进程权重的方式解决这样的问题。总之，软实时调度算法应该动态调整各个任务的CPU资源量，以保证实时性任务分得的CPU资源较多。

# 四、修改自旋锁为互斥锁

## （一）自旋锁和互斥锁

锁是用于解决进程同步问题的系统资源。当多个进程都要访问互斥资源的时候，需要某一个进程获取资源的锁，然后进入临界区，对资源进行操作。其他进程需要等待持有锁的进程将锁释放。内核中，锁有自旋锁，有互斥锁。自旋锁是进程将访问的资源的锁被其他进程占用时，这个进程就会在核心上循环等待，直到这个锁被释放。互斥锁是当一个进程访问某个互斥资源，但是互斥锁被其他进程占用时，进程会先放弃CPU而进入阻塞队列。可以发现自旋锁情况下，进程循环等待自旋锁的释放，是对CPU资源的浪费。



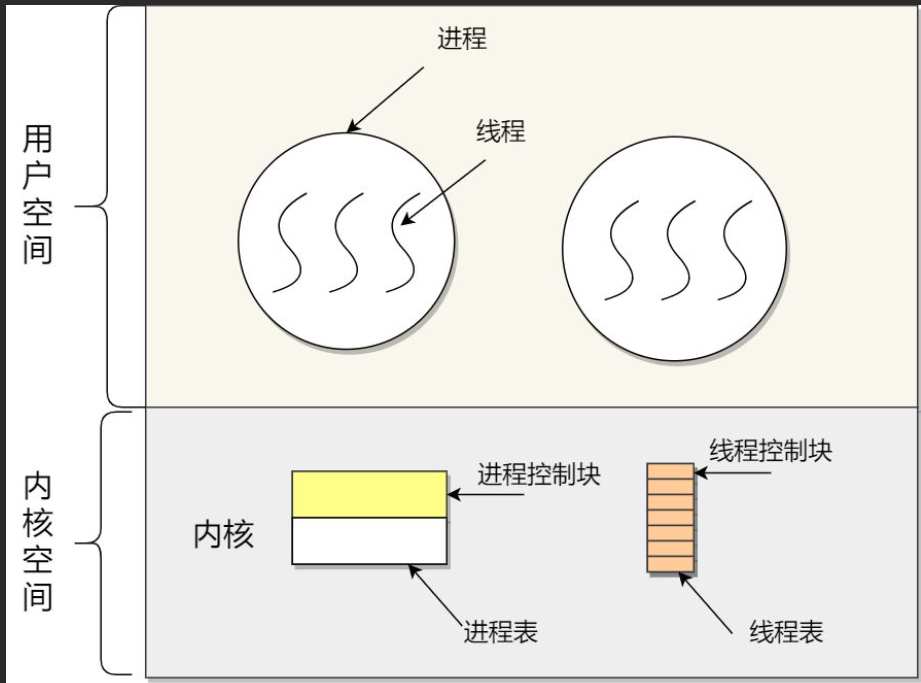
## （二）修改锁以提高实时性

在Linux中，spinlock是自旋锁，其目的是为了用来对临界资源进行保护，但是如上一节所说，自旋锁存在一些缺陷，就是在没有获得该临界资源锁时不会放弃CPU，而是不断地循环等待。为了解决这样的问题，可以用mutex互斥体来取代spinlock来完成spinlock的可抢占性。虽然使用这种方法能够解决上述问题，但是仍然存在抢夺失败的情况发生，所以在每个spinlock内增加等待队列来应对这样的问题。

# 五、将中断任务线程化

## （一）用户线程和内核线程的区别

在操作系统第四章中介绍了用户线程和内核线程的主要区别。用户线程和内核线程都是由用户态创建的，但是用户线程由用户态进程管理调度；内核级线程由内核进行管理和调度。所以内核感知不到用户级线程的存在，它们只能运行在其所在的进程所运行的那个核上。但是内核级线程由内核直接操纵，而且同一任务的不同线程可以在不同的核心上工作。（这样就可以实现之后的将中断任务线程化的策略了）。所以我们可以看出，内核级线程由于一个进程下的多个线程可以并行的运行在多个核心上，并发性较好。用户级线程只能运行在同一个核心上，并发性较差。将中断任务线程化实际上是将任务转化为内核级线程。



## （二）中断线程化

在实时性较高的任务执行过程中，当中断服务程序将要执行时，可能是系统中存在一些重要的事情需要进行处理。但是中断服务程序中也可能包含一些不是很重要的操作。那么我们可以把中断服务程序拆分为多个线程，其中一些承载重要任务的线程首先执行，然后那么承载着不重要任务的线程延后执行。就可以整体减少系统中任务等待中断的延时，提高系统的实时性。

在Linux中，中断是优先级最高的执行单元，一旦发生中断事件，系统将立即响应该事件并执行相关的中断处理代码，除非系统已经关中断。因此，如果我们的计算机有着严重的外部IO负载或者网络阻塞时，中断会非常频繁，我们电脑中的实时任务几乎得不到执行（网络不好，卡），实时性无法保障。但是我们中断线程化以后，中断就会作为内核线程，并且被赋予不同的优先级。此时，实时任务（具有高优先级）就可以作为较高优先级的执行单元来运行，这样在严重负载的情况下依然有实时性。

其实，中断线程化是必须的，因为Linux中中断处理代码中大量的使用了自旋锁（根据参考论文），当自旋锁被mutex取代后，中断处理代码会因为得不到锁而被挂在等待队列上去，而这需要可调度的进程来实现，因此中断必须线程化。

另外，并不是所有的中断都可以被线程化，比如时钟中断，主要用来维护系统时间以及定时器等，其中定时器一旦被线程化，操作系统将不能正常工作。

# 四、结论

在操作系统中有很多不同的角度都可以进行系统任务的实时性优化，再进行实时性优化的思考过程中，我们不光要考虑操作系统，硬件各个模块，还要考虑系统中的事件。比如说，任务在运行过程中一定会被其他任务以及内核中断等事件所打断，这样是极其影响任务响应时间的，可以通过减少这样的事件来进行实时性优化。还有从全局考虑，调度算法也影响着任务平均响应时间，那么可以通过优化调度算法。在进程同步过程中用到的自旋锁本身有着浪费CPU资源的特点，那么修改锁可能会提高CPU利用率从而减少等待时间。还可以将系统任务细分为线程，先执行一部分，另一部分延后执行，这样可以隐式的提升实时任务的优先级。

对操作系统实时性优化的思路就在于操作系统的各个模块，和各个处理过程。

# 参考文献

[1]唐剑寅. EAST PCS操作系统实时性定制与优化研究. 中国科学技术大学，2018.

[2]吴华侨.基于Zynq平台的嵌入式操作系统实时性优化设计.哈尔滨理工大学,2018-3.

[3]肖振华. 嵌入式Linux操作系统的实时性研究. 太原科技大学，2008-7.