操作系统论文

120L022109 李世轩

2023 年 6 月 25 日

摘要

随着计算机技术的快速发展, 非实时操作系统在个人电脑和服务器中的应用越来越广泛。相比于实时操作系统, 非实时操作系统更注重系统吞吐量和资源利用率。但是, 对于非实时操作系统，如果也能够进行实时性的优化，则可以提高系统的性能和稳定性，提高用户的使用体验。为了满足用户的实时需求, 也需要对非实时操作系统进行实时性优化。本文将从调度算法、中断处理和架构优化等几个方面探讨非实时操作系统的实时性优化方法。

目录

1 区别分析

我们在日常工作学习环境中接触最多的是通用操作系统，通用操作系统是由分时操作系统发展而来，大部分都支持多用户和多进程，负责管理众多的进程并为它们分配系统资源。分时操作系统的基本设计原则是：尽量缩短系统的平均响应时间并提高系统的吞吐率，在单位时间内为尽可能多的用户请求提供服务。由此可以看出，分时操作系统注重平均表现性能，不注重个体表现性能。如对于整个系统来说，注重所有任务的平均响应时间而不关心单个任务的响应时间，对于某个单个任务来说，注重每次执行的平均响应时间而不关心某次特定执行的响应时间。

通用操作系统中采用的很多策略和技巧都体现出了这种设计原则，如虚存管理机制中由于采用了 LRU 等页替换算法，使得大部分的访存需求能够快速地通过物理内存完成，只有很小一部分的访存需求需要通过调页完成，但从总体上来看，平均访存时间与不采用虚存技术相比没有很大的提高，同时又获得了虚空间可以远大于物理内存容量等好处，因此虚存技术在通用操作系统中得到了十分广泛的应用。类似的例子还有很多，如 Unix 文件系统中文件存放位置的间接索引查询机制等，甚至硬件设计中的 Cache 技术以及 CPU 的动态分支预测技术等也都体现出了这种设计原则。由此可见，这种注重平均表现，即统计型表现特性的设计原则的影响是十分深远的。

而对于实时操作系统，前面我们已经提到，它除了要满足应用的功能需求以外，更重要的是还要满足应用提出的实时性要求，而组成一个应用的众多实时任务对于实时性的要求是各不相同的，此外实时任务之间可能还会有一些复杂的关联和同步关系，如执行顺序限制、共享资源的互斥访问要求等，这就为系统实时性的保证带来了很大的困难。

因此，实时操作系统所遵循的最重要的设计原则是：采用各种算法和策略，始终保证系统行为的可预测性 (predictability)。可预测性是指在系统运行的任何时刻，在任何情况下，实时操作系统的资源调配策略都能为争夺资源 (包括 CPU、内存、网络带宽等) 的多个实时任务合理地分配资源，使每个实时任务的实时性要求都能得到满足。

与通用操作系统不同，实时操作系统注重的不是系统的平均表现，而是要求每个实时任务在最坏情况下都要满足其实时性要求，也就是说，实

时操作系统注重的是个体表现，更准确地讲是个体最坏情况表现。举例来说，如果实时操作系统采用标准的虚存技术，则一个实时任务执行的最坏情况是每次访存都需要调页，如此累计起来的该任务在最坏情况下的运行时间是不可预测的，因此该任务的实时性无法得到保证。从而可以看出在通用操作系统中广泛采用的虚存技术在实时操作系统中不宜直接采用。

2 调度优化

非实时操作系统通常采用时间片轮转或优先级调度算法，但这些算法不足以满足实时系统的需求，因此需要研究更加适合实时系统的调度算法。

最常用的实时调度算法包括最高优先级优先调度算法(Highest Priority First, HPF)、最早截止时间优先算法 (Earliest Deadline First, EDF)、Rate-Monotonic Scheduling (RMS)、Deadline-Monotonic Scheduling (DMS) 等。这些算法都有各自的优缺点，需要根据具体的应用场景选择合适的算法。

最早截止时间优先算法 (Early Deadline First，EDF)。这是一种将任务的截止时间作为调度优先级衡量标准的调度策略，一个任务的截止时间越近则越有可能得到优先调度，避免任务超时。这种调度策略是根据任务的截止时间来动态分配任务的调度优先级，当有新的任务就绪，或者一个任务运行结束时，EDF 算法需要对系统中就绪的任务优先级进行调整，在运行中开销较大。采用 EDF 算法的调度器只需要知道任务的截止时间即可，因此通用性较强，可扩展性也较高。对于单核场景下的一组互不相干的实时任务来说，Dertouzos 证明了 EDF 调度算法是理论上最优的动态优先级实时调度策略。

先来先服务调度算法 (First Come First Serve，FCFS)。这种算法按照任务就绪的时间来调度执行任务，一个任务一旦运行，其他任务必须要等待该任务运行结束才会得到调度，这会导致后续的任务产生饥饿问题，实时性不高。

最短优先调度算法 (Shortest Job First，SJF)。这种算法根据任务的执行时间长短来划分任务的优先级。执行时间越短的任务越先得到执行，如果不断有较短执行时间的任务就绪，那么这种算法会导致执行时间较长的任务产生饥饿问题。对于 RTOS 来说这种调度算法也不适用。

时间片轮转调度算法（Round-Robin,RR）。该算法将任务的执行时间划分成固定长度的时间片，每个任务轮流占用时间片长度的 CPU，当一个任务的时间片用完时，调度执行下个任务。这是一个比较公平的调度算法，系统中的任务都能得到适当的执行时间。但是如果时间片设置的较短的话，会有较多的任务切换次数，带来较大的任务切换开销，时间片设置过长又会导致在队列末尾的任务迟迟得不到调度，因此需要确定合适的时间片大小。

速率单调（Rate-Monotonic，RM）算法是静态优先级实时调度算法的代表，该算法常用于周期任务的调度，算法中速率是指周期任务的到达速率，为任务周期的倒数，假设任务的周期为 T，任务的速率就是 1/T。之所以说速率单调算法是静态优先级调度算法，是因为它需要提前知道各个任务的周期，在任务运行前根据任务的周期信息静态地为每个任务分配优先级，任务的周期越短，证明任务的截止时间要求越紧迫，需要分配给其更高的优先级。RM 算法支持抢占调度，高优先级的任务可以抢占低优先级的任务的 CPU 控制权，而且 Liu 和 Layland 证明了如果一组任务在基于 RM 算法的调度器中无法在截止时间之前完成，那么这组任务在基于其他静态调度算法的调度器中也是不能在截止时间完成的，即在所有静态优先级实时调度策略中，RM 算法是最优的。静态优先级调度策略的优点是优先级固定，实现简单，任务调度延迟时间比较稳定。

3 中断处理

中断提取法是一种对中断处理的革新方式，其主要是在架构上的改变，但是页可以看作对中断处理的优化方式。

近些年标准 Linux 内核的进步较为明显，虽然其默认的调度器称为完全公平调度器，但仍然不是完全公平的，仅趋近于公平，系统仍然不能给出确定性的时间响应，仍然属于软 RTOS 的范畴。除此之外，大量系统级别的 I/O 重排序和 I/O 请求组合均是以牺牲实时性能为代价换取吞吐的提升。对于 RTOS 统海量的应用来说，只有其中的一小部分才真正需要硬实时的确定性。例如对高速 PID 回路控制或机械臂控制这些场景才需要硬实时，而用于维护所记录温度的 PID 回路和显示机械臂当前位置的图形绘制

来说，一般并不需要硬实时。

因此可以通过区分实时应用和非实时用的方式在 Linux 上实现硬实时，即将 Linux 内核作为一个低优先级的任务运行在一个精简的实时内核上，实时功能则通过运行在实时内核上的高优先级的任务来处理。对于非实时的功能，如图形显示、文件管理、网络通信等则通过 Linux 内核直接管理，可以节省大量的开发时间。上述方法实际上由实时内核接管了 Linux 内核的中断处理，因此可以认为 Linux 内核关闭了中断，因此称为中断提取法，由于此时系统拥有两个不同的内核，因此也称双内核法。图中给出了中断提取法的架构图，实时内核快速地截取并处理硬件的中断响应，避免硬件中断直接作用到 Linux 内核，将硬件中断转化为虚拟软件中断，此时 Linux 内核不再直接控制硬件中断的启停。

对于中断提取法来说，Linux 内核是一个低优先级的任务，当实时任

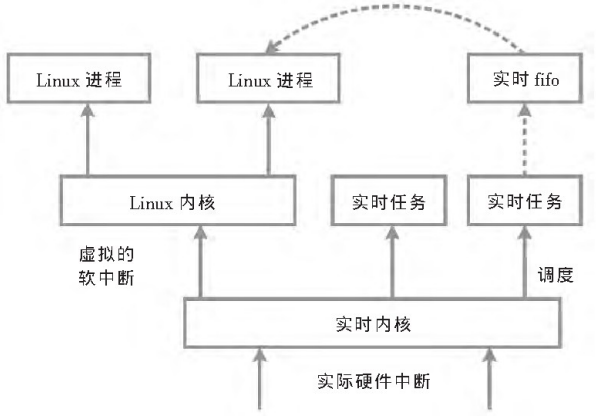

图 1: 中断提取法架构

务不运行或不准备运行时才执行，虽然实时内核处理硬件中断需要花费一些额外的时间，但是相较于由 Linux 内核的处理时间要短的多。由于实时内核非常精简，因此可以提供可靠的延迟上限，属于时硬实时的范畴，此外，该方法虽然需要修改系统内核，但相较于抢占改进法的修改内核工作量来说要小很多。目前中断提取法有两个较为成功的实际应用案例，即由墨西哥矿业及科技学院开发的 RTLinux 和意大利米兰工业大学开发的 RTAI。

4 架构优化

接下来以 Linux 的实例，对这部分进行讲解。

Linux 系统最初是按照分时系统设计并推出的，再加上在历史版本中使用的调度算法目的是公平的分配和使用各种系统资源，保证 CPU 被各个进程公平的使用，所以早期并不支持实时性。但是在后来的 2.6 版本开始，加入了内核抢占的功能，使它的实时性得到了提升，在某种程度上具备了软实时的能力。

由于 Linux 系统内核过于庞大且模块众多，内核中仍然有不少影响实时性的因素，比如使用大量自旋锁、中断禁止、时钟粒度等，使其距离 us 级别的控制精度还有很大的距离。但是也不能因此认定 Linux 系统今后就不能用于实时控制领域。

其实 Linux 内核一路发展过来，在历史的版本主线中仍然有很多技术公司或者大牛为了提升 Linux 系统的实时性而努力着，他们或是在某个版本上发布实时补丁，或是对内核进行一定程度上的改造，具体的代表有 RTLinux、RTAI(Real-Time Application Interface) 和 Xenomai 等。RTLinux 全称叫做 AReal-Time Linux，它由美国新墨西哥矿业及科技学院的 V. Yodaiken 开发。

RTLinux 采用了双内核的做法，可以说是开创了双内核法的先河。简单理解就是系统中存在两个内核，实时核和非实时核。底层硬件资源和实时的内核打交道绕开非实时核，来自硬件的中断源由实时核全面接管，把非实时的 Linux 内核当成实时核上的一个低优先级的进程来运行，通过这种方式确保实时核上的中断和任务得到优先响应，提升了实时性。

Xenomai 也借鉴了 RTLinux 的双内核做法，内部也有实时核和非实时核。但不同的是 Xenomai 在底层硬件和两个内核之间还加了一层硬件抽象层 ADEOS(Adoptive Domain Environment for Operating System)，实时核和非实时核作为硬件抽象层的两个域而存在，Xenomai 内核属于实时域，Linux 内核属于非实时域。ADEOS 在系统的关键路径中对中断进行拦截，优先响应 Xenomai 实时域的中断，当没有实时任务和中断需要处理的时候才会轮到 Linux 内核执行。两者对比如下。

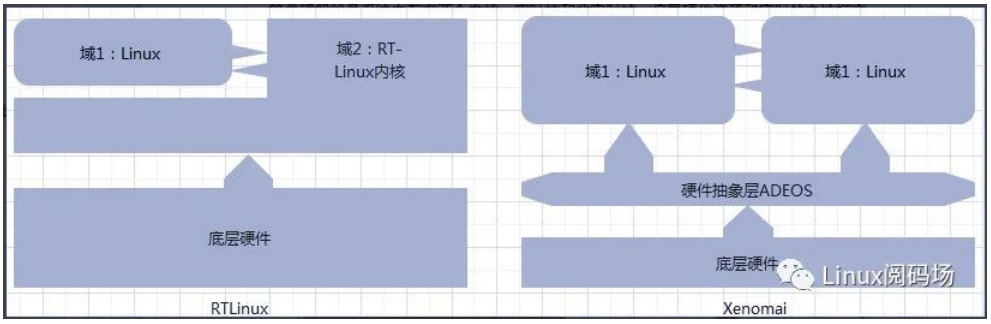

图 2: RTLinux 和 Xenomai 架构对比

5 其他优化方法

5.1 资源访问优化

在实时系统中，共享资源的访问可能会引发优先级翻转现象，从而影响系统的实时性。优先级翻转现象是指高优先级任务需要等待低优先级任务释放共享资源的情况。为了避免这种情况的发生，需要采用一些资源访问优化方法。

可以研究如何优化对共享资源的访问，如通过优先级继承、优先级上升等机制来避免优先级翻转现象的发生，提高资源访问的实时性。例如，可以通过优先级继承机制，将低优先级任务的优先级提升至高优先级任务的优先级，从而避免高优先级任务等待低优先级任务释放资源的情况。还可以通过优先级上升机制，将低优先级任务的优先级提升至高优先级任务的优先级，从而避免低优先级任务长时间持有资源的情况。

5.2 任务切换时间优化

在实时系统中，任务切换时间也是一个重要的指标，它包括保存任务上下文的时间、切换任务的时间、恢复任务上下文的时间等。任务切换时间长，会降低系统的实时性。

可以研究不同的任务切换机制，如基于线程的任务切换和基于进程的任务切换，分析它们的实时性能，提出减少任务切换时间的优化方法。例如，可以通过减少任务切换时保存和恢复任务上下文的数据量、优化任务

切换的算法等手段，减少任务切换时间。还可以通过采用基于协程的任务切换，避免任务的上下文切换，从而进一步减少任务切换时间。

5.3 预分配资源

在实时系统中，预分配资源是一种重要的优化手段，它可以在空闲时间预先分配实时任务需要的资源，如 CPU 时间片、内存等，以减少实时任务的资源等待时间，提高实时性。预分配资源可以使实时系统的资源分配更加合理，避免实时任务和非实时任务的资源冲突，从而提高系统的实时性。

可以研究如何利用空闲时间预先分配实时任务需要的资源，并提出相应的策略。例如，可以根据实时任务的优先级、周期等特征，动态调整预分配资源的策略，以适应实时任务的需求变

参考文献

[1] 马可. 面向实时操作系统的实时性分析方法研究 [D]. 电子科技大学,2022.DOI:10.27005/d.cnki.gdzku.2022.003693.

[2] 程旭, 倪宝成, 张东泽. 基于 Linux 的实时仿真机操作系统改进研究 [J]. 电脑编程技巧与维护,2020,No.425(11):120-123+129.DOI:10.16184/j.cnki.comprg.2020.11.045.

[3] Linux 内核之旅. (2022, 5 月 11 日). 浅谈 Linux 内核的实时性优化. 取自 https://mp.weixin.qq.com/s/Mq_sVpZYM3BrX2NpixBgaw