节能软件*

作者：艾瑞克·萨克斯
译者：郭　耀

关键词：节能软件　功耗

拥有功耗管理功能的硬件有助于节省能量消耗，那么软件开发人员又应该如何应对能耗问题呢？

功耗管理功能的发展速度大大超出了人们的预期。从最小的传感器和手持设备，到数据中心内的“大铁疙瘩”服务器，几乎任何规模和类型的计算机系统都提供了很多降低、度量和控制功耗的功能。如果没有这些功能，风扇的噪音就可能会破坏办公环境，离开了充电器的笔记本电脑用不了几个小时，而且用户还得忍受它的热度，而数据中心的电费和冷却开销则会居高不下。

虽然我们通常会认为功耗管理的功能是与硬件密不可分的，但是软件对提高整个系统效能所起的作用已经功不可没。尽管“软件的功耗效率”这个概念听起来似乎有些怪异（因为软件并不会直接消耗电源），然而软件与消耗电能的系统资源之间的交互作用方式对降低能耗起了重要作用。

我们的讨论从把软件划分成人们熟知的两种生态系统角色开始，这两种角色即资源管理者（生产者）和资源请求者（消费者）。下面我们要探讨每种角色对整个系统的效能产生什么影响。

电源管理的历史起源于较小的系统和移动领域。按照现今的标准，这些系统都是相对比较简单的，部件的数量很少，例如一个单核CPU，以及一个可以降速的磁盘。由于这些系统拥有的资源很少，因此在实际使用中常表现为双态：系统资源要么在使用中，要么闲置。因此，针对这些资源进行功耗管理的策略也就可以相当简单，并且很有效。

例如，可以用一个后台程序对系统的利用率进行周期性监控，如果发现空闲时段超过某个时间阈值，就降低CPU频率和磁盘转速。所采用的方式不大需要（或根本不需要）与其他和资源管理相关的子系统（例如：调度器、文件系统等等）整合即可完成，因为如果利用率是0，也就不需要进行什么资源管理。

与之相比，现代系统的拓扑结构要复杂得多。随着“CPU频率可以不断增长”这个“免费的性能午餐”成为历史，我们正在面临多核革命。结果是即使最小的便携设备也包含了需要进行管理的多个逻辑CPU。随着系统规模不断扩大（也就会包含更多的功耗可管理的资源），经常会遇到这样的情形：系统的一部分是忙的，而其余部分则闲置着。当然，CPU只是功耗可管理系统资源中的一个例子，物理内存中的不同部分也会成为功耗管理的对象，磁盘存储和I/O设备也如此。在更庞大的数据中心的环境中，甚至系统自身也可以作为资源功耗管理的对象。

要对现代系统进行有效的资源管理，就需要至

* 本文译自Communications of the ACM 文章Power-Efficient Software Vol. 53 No. 2, Pages 44-48

少在资源管理器的某些层次上感知不同资源的功耗状态的差异，并且设法加以利用（实际上，有效资源管理要求对资源功耗差异的总体感知，资源不同的功耗状态造成了这种差异）。根据被管理的对象不同，资源管理需要分别考虑空间、时间，或者二者同时考虑。

空间上的考虑

空间上的考虑需要决定应该提供哪些资源来及时满足消费者的请求。对于操作系统线程调度器/分配器来说，可能是决定要运行的线程应该分配给哪些CPU，以及确定线程在整个系统的物理处理器上的最佳分布模式，以实现某种策略目标（性能、能耗效率，等等）；对于虚拟内存子系统来说，类似问题涉及物理内存的使用；对于文件系统/卷管理器来说，意味着跨磁盘的块分配策略；对于数据中心，要考虑如何在不同物理系统之上分配虚拟机。图1中给出了这些不同类型的资源管理器。

空间上需要考虑的一个问题是可用资源的当前功耗状态。在某种意义上可以说，一个资源的功耗状态表示一组权衡折中。有些状态可以提供一种机制来支持系统在性能和功耗效率间权衡折中（例如CPU频率调节），而其他状态下可能提供（为空闲资源）在功耗消耗与恢复时延（例如，使用ACPI的C-State）之间权衡。因此，资源管理器（根据功耗状态）选择何种资源的决策是做出权衡的重要手段。理想状态下，这应当成为对于单个资源进行功耗管理策略的补充。

在什么样的资源粒度上进行功耗管理是另一个重要的空间考虑。如果多核处理器只能在插槽级别（socket level）进行功耗管理，那么就可能需要把系统负载合并到尽量少的插槽之上。合并会使得有些资源的利用率提高，而使另一些资源停用。这样就可以对停用的资源进行管理，把功耗（连同性能）都“转移”到系统中投入使用到的那一部分之上。

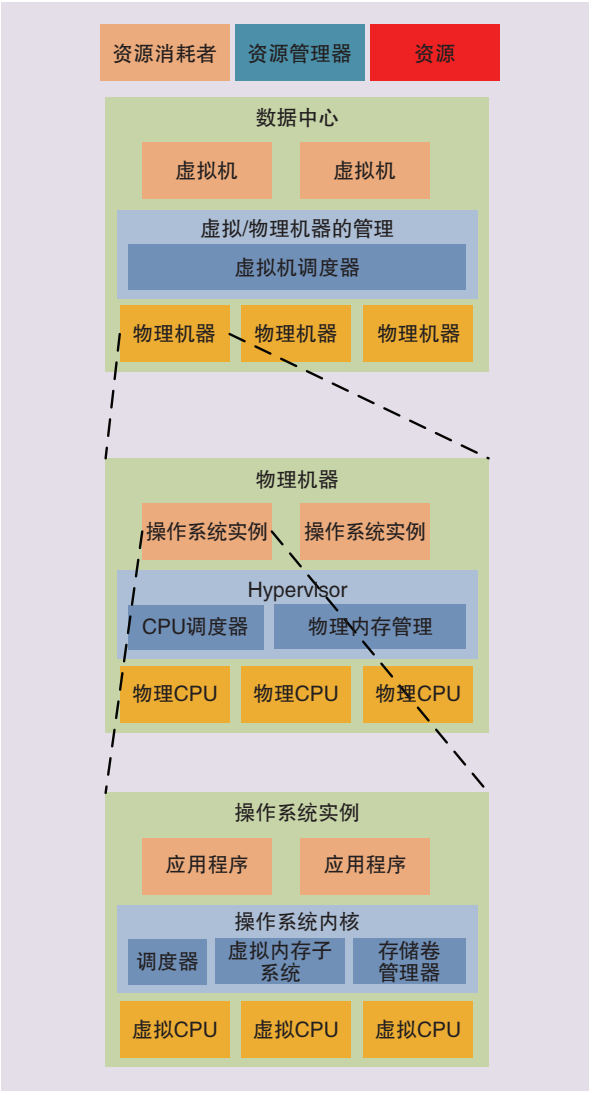

图 1 资源管理器的层次结构

影响单个资源的选择和利用率分配的决策的另外一个因素是正在使用的资源的负载特性。例如，资源管理器对系统利用率的整合要掌握到什么“火候”才不致（因为带来资源竞争而）对性能产生负面影响，或者所使用的资源的功耗调整到什么程度才不致影响消费者可利用的性能。

时间考虑

有些资源管理器可能会按照时间以及空间（或

者仅按照时间而不考虑空间）来分配资源。例如，计时器子系统可能允许客户把过程安排到未来某时间点（或者某时间段内）执行，或者由一个任务队列子系统提供异步或延迟执行的手段。传统上这样的子系统的接口是很窄的，而且都是预先规定好的，因此难以在时间上优化。一种解决方案是使客户端接口更具有可描述性。下面是一个用来精确描述事件和发生时间的计时器狭窄接口的示例[†]：

```
int
schedule_timer(
  void (*what)(), time_t when);
```

或者用一个计时器接口来确定在什么时间需要做什么，同时又描述清楚约束条件：什么时候应该有什么事件发生，如：

```
int
schedule_timer(
  void (*what)(), time_t about_when,
  time_t deferrable_by,
    time_t advancable_by);
```

与把负载合并到更少的插槽之上来提供空间资源的空闲率类似，提供时间上的范围区分就使计时器子系统得以对进程的消耗进行合并和批处理。这样在一个给定时间间隔中就不需要把CPU唤醒n次来处理n个计时器（每次唤醒都会带来一些开销），计时器子系统可以唤醒CPU一次就批处理所有符合（更宽松的）约束的计时器，从而可以降低CPU的时间开销，增加功耗可管理的状态时段（参见图2）。

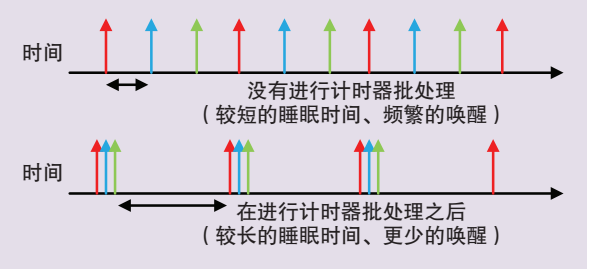

图 2 对周期性计时器进行批处理的好处

提高资源效能

很显然，资源管理器会给系统整体效率（的提高）做出很大贡献，但最终它们还是要在系统的资源消费者所制定的约束和要求下工作。如果由于这些约束太强而造成资源被过度分配或者没有能够有

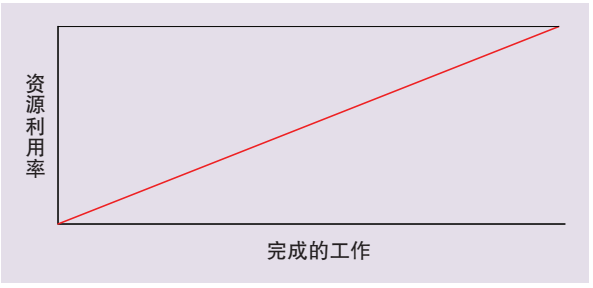

图 3 好的效率

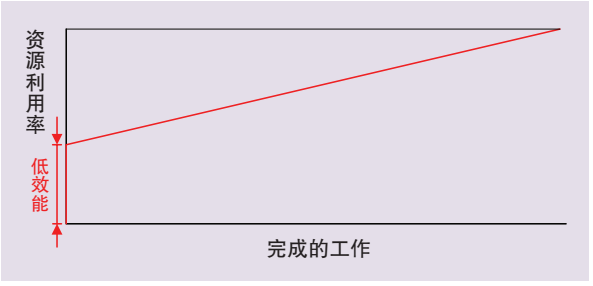

图 4 “空闲”时的低效能

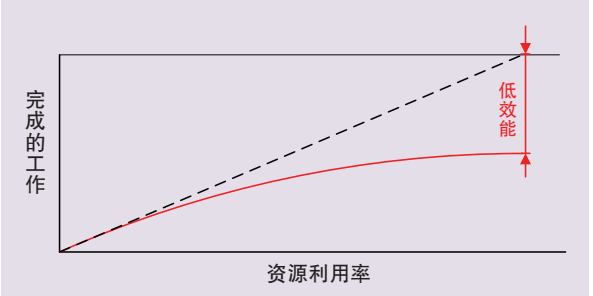

图 5 伸缩时的低效能

[†] 下面两段示意代码有错，我们做了订正—编者

效使用，那么即使是最复杂的功耗管理功能也无法发挥作用，整个系统的效率就只能凑合了。

设计良好而且高效的软件是很美的，它可以使所消耗的资源量与所体现出的性能相称。对于理想的软件来说，这种关系应当是完全线性的——在没有执行任何任务时的资源消耗为0；而随着资源使用的提高，所完成的任务也会按比例增加（参见图3）。

但是，真正的软件不是“乌托邦”，要想让软件消耗资源为0的唯一方式是根本不去运行它。在实践中，即使是以最小需求来运行设计得很好的软件，也会需要额外的资源开销。

与之相反，低效的软件则表现为资源利用率和所完成任务之间不成比例。以下是一些软件运行中常见的影响效率的情况：

1. 一个进程处于等待状态，它可能是在等待某个条件的满足，并且会使用计时器周期性地唤醒自己来检查该条件是否满足。在它等待的过程中并没有执行任何有用的任务，但是它每次醒来检查时，都迫使CPU从空闲转为忙碌。而如果该进程决定以提高唤醒频率来“使延迟最小化”，则情况更糟（参见图4）。

2. 一个应用程序使用多个线程来提高并发性，同时扩展其吞吐率。即使由于内部的瓶颈使得其能够利用的线程无法超过某一限量，只要系统中还有CPU可用，程序仍旧盲目地创建更多线程。而更多的线程意味着必须唤醒更多的CPU来运行，尽管此时添加一个线程所提供的性能提高已经很少，甚至为零（参见图5）。

3. 某个服务在缓慢地泄漏内存。随着时间增长，它的堆会增长而消耗系统中大量的物理内存，尽管其中大部分甚至全部都未被使用。结果，由于绝大多数内存都被分配掉了，所以很难对内存进行功耗管理。

从生态系统角度观察软件的低效能

要对软件效率进行综合分析，必须观察资源利用率和完成的工作之间的比例。对“工作”进行度量是与负载相关的。有些负载（例如Web服务器和数据库）的“工作”度量是基于吞吐量。对于这样的负载，可以将吞吐量（例如，每秒完成的事务处理数）与资源消耗{cpu|内存|带宽|存储}的关系绘制成曲线。如果在曲线中存在“拐点”（资源利用增加，而吞吐量不相应增加），则有可能存在使用更少的资源来完成相同的工作，或使用相同资源来完成更多的工作的途径。

对于使用并发来提高处理速度的并行计算负载来说，可以绘制计算时间和所消耗资源的关系曲线，使用类似的技巧来找到回报率下降的点，并加以避免。

除了使用负载分析之外，另外一种有用的技术是查看在0利用率（系统空闲）时的全系统资源利用行为。根据定义，系统此时并没有做任何有用的事情，因此还在主动消耗CPU周期的任何软件都是可疑的。PowerTOP是英特尔公司开发的一个开源实用程序，专门用来支持这种分析方法（参见图6）。如果在一个应当是空闲的系统上运行该工具，在理想情况下，系统的处理器100%时间下都是处在功耗管理之下的。但是在实践中，低效的软件（通常是执行基于时间的周期性轮询）会使得CPU部分处于忙碌状态。PowerTOP会显示浪费的程度，同时找到浪费是哪些软件造成的。然后系统用户就可以把这些观察到的浪费作为软件缺陷（bug）提出报告，并且可以选择运行更高效的软件。

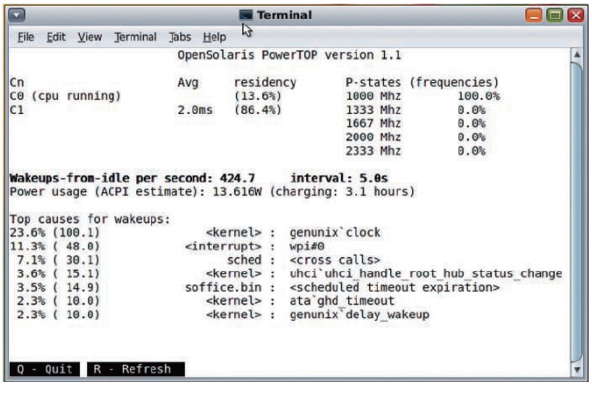

图 6 PowerTOP

设计高效能的软件

把效能当做软件设计和优化的着眼点，人们在开始时会觉得有些不习惯，所以我们可以把它先同一些更为成熟的目标，例如性能和可伸缩性，进行一些对比：

1. 在给定资源的条件下， 高性能软件会使完成的有用工作最大化（或者使完成该任务所需的时间最小化）。

2. 可伸缩软件则意味着：使用更多资源时，性能也会成比例提高。

高效能软件可以被认为既是高性能，也是可伸缩的，但同时满足有关资源利用率的一些附加约束：

1. 给定固定的性能级别（要完成的工作量，或是完成它所需的时间），软件会使用最小的资源集合。

2. 随着性能需求的降低，资源使利用率会成比例下降。

这意味着，除了要查看在给定资源利用率时性能的数量和比例关系之外，要想获得高效能，软件设计者还需要考虑给定性能情况下资源利用的数量和比例关系。如果觉得这些看起来太抽象，我们可以将之归结成下面这些更具体的因素：

当设计由程序预取资源的软件时，要确保程

序理解哪些资源是完成工作所必需的，而在不再需要的时候把这些资源释放以支持空闲资源的功耗管理。如果预取的资源只是断断续续需要，则该软件应当启动能向资源管理器发出提示的功能，这些提示可以包括资源什么时候会（或不会）被使用，从而可以为主动的功耗管理提供支持。

对于CPU利用率：

1. 当线程在等待某个条件时，应尝试利用一种事件触发的机制来消除基于时间的轮询。尽量不要使用意思是“我们现在可以执行了吗？”的语句。

2. 如果做不到，则可以尝试使轮询不那么频繁。

3. 如不能完全消除轮询，则应尝试至少保证所有周期性的和轮询的活动都会进行批处理。利用计时器子系统功能来提供优化的范围，例如加大时间段划分粒度，或者允许计时器提前或延时。

对于内存利用率：

1. 监视内存泄露。

2. 释放或卸载不再需要的内存。有些操作系统提供了关于内存利用率的指导性接口，例如Solaris中的madvise(3c)。

对于I/O利用率：

如果可能的话，对I/O请求进行缓冲或批处理。

建立高效的系统软件栈

过去的经验表明，在硬件设计上的演进和创新总会为软件带来新的机会和挑战。现在即使在最大型的系统中，也普遍具备了降低利用率较低的系统资源能耗的硬件功能，负责管理这些资源的软件层必须跟着演进——实现新的策略以提高正在工作的资源的性能，同时降低利用率较低资源的能耗。

除了资源管理器之外，资源消费者也有很大的机会对更广范围软件栈的效能产生正面或负面的影响。类似于PowerTop的工具已经提供了一个很好的开始，使程序员和管理人员可以更好地观察软件的低效能，找到优化的参考点，并且认识到软件在高效能计算中所起的重要作用，虽然让程序员重新思考软件的设计方式不仅是一个技术问题。■

在queue.acm.org的相关文章：

Powering Down

Matthew Garrett, http://queue.acm.org/detail.cfm?id=1331293

Maximizing Power Efficiency with Asymmetric Multicore Systems

Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto, http://queue.acm.org/detail.cfm?id=1658422

Modern System Power Management

Andrew Grover, http://queue.acm.org/detail.cfm?id=957774

作者：

艾瑞克·萨克斯（Eric Saxe）

Sun Microsystems（升阳）公司Solaris内核开发部门的工程师。

译者 郭 耀

中国计算机学会会员。北京大学信息科学技术学院、高可信软件技术教育部重点实验室副教授。主要研究方向为操作系统、低功耗设计、软件工程等。yaoguo@sei.pku.edu.cn