附录 A　Linux 2.4

内核的开发人员们并没有因为我在写这本关于 Linux 的书而暂停他们的工作。（我是多么希望他们这样做……。）说真的，内核的发展进展神速，就在这本书即将出版之际，它的最新稳定版本，2.4.0，也应该面世了。尽管在写作本书时要将对内核的成千种修改都一一涵盖到是不可能的，但是这篇附录却总结了发生在本书所涉及内核部分的最令人感兴趣的那些改变。这些修改中的绝大部分已经在内核的 2.3.12 版本里实现了，它们被包括在附赠的 CD-ROM 上。

作者十分感谢 Joe Pranevich 的论文，”Wonderful World of Linux 2.4”（**http://linuxtoday.Com/stories/8191.html**），在准备这篇附录时它提供了无价的帮助。

更少的“惊扰（stampedes[1]）”

请读者回忆一下第 2 章里，函数**__wake_up**（26829 行）唤醒等待在等待队列上的所有进程。可以考虑一个诸如 Apache 一样的 Web 服务器，它试图通过在同一端口派生出许多进程监听连接申请以缩短响应时间（正好与等待连接请求到达，并且与只通过 **fork** 派生一个进程来响应的方式相反）。所有这些进程都在一个等待队列里，等待连接请求到达，而当请求确实到达的时候，它们都会被唤醒。它们当中只有一个能够为请求提供服务，所以剩下的又都将返回休眠状态。

这种“惊扰”现象会浪费 CPU 的时钟周期——只唤醒一个等待进程会更好，因为不管怎样只有一个能够运行。因此，一个新的任务状态位 **TASK_EXCLUSIVE** 就被引进；在一次调用**__wake_up** 里，最多有一个设置了 **TASK_EXCLUSIVE** 位的任务会被唤醒。**TASK_EXCLUSIVE** 位是附加在其他任务状态位上的——它并不代表一个新的任务状态，它只是为了方便而保存在任务状态信息组里的信息而已。

现在**__wake_up** 要检查它正在唤醒的进程是否设置了 **TASK_EXCLUSIVE** 位，并且在它唤醒该进程之后便不再唤醒其他进程（通过使用 **break** 中断循环）。**TASK_EXCLUSIVE** 位现在只被用在有关网络的等待队列上，而且它在那种环境中效果也的确很好。对于大多数其他等待队列来说，你需要所有进程对等待着的资源都有机会进行占用，这样可以避免饥饿现象。不过等待在同一个端口的服务程序通常与 Apache 境遇相同：所有等待的任务都是一样的，而且重要的是只要它们中有一个能够处理请求就可以，即使每次都是同一个任务（也无所谓）。

再见吧，Java

正如第 7 章里所预言的那样，Java 二进制处理程序（binary handler）已经从内核里消失了，它是因被杂项二进制处理程序所替代而失效的。二元处理程序的常规用法没有变化，Java 执行体仍然可被杂项二进制处理程序的适当配置所完全支持。

[1] “stampede”原意是指：动物受到惊吓后纷纷逃窜。作者使用该词说明：在等待队列中休眠的进程同时被唤醒的情况。

ELF 权能位

已有非官方的修补程序可以用于（增强）ELF 执行体中权能的存储。它们还没有成为正式内核版本的一部分，部分原因在于是否 ELF 文件头是保存这些信息的恰当位置的问题还处于讨论之中。不过，这些修补程序也有可能已经成为正式版本的一部分了。

调度程序提速

已被高度优化了的 **schedule** 函数（26686 行）又被作了进一步优化。大多数修改只是在 **system_call**（171 行）系列调用的基础上进行了重新组织——也就是说，通过允许普通情况直接通过以及把遍布在函数里的大部分 **if** 语句体都分散开来的方式提高代码的（运行）速度。举例来说，第 26706 和 26707 行，如果又要运行的代码它们就运行底下的下半部分（bottom halves），现在采用的是这种形式：

P563—1

这样一来，如果下半部分程序必须运行，控制流程就会跳转到新的 **handle_bh** 标记处，然后在运行完下半部分之后再跳转回去。原有方式在无需运行下半部分时的正常情况下也要产生一个分支转移，因为在那种情况下所产生的代码不得不跳过对 **handle_bottom_halves** 的调用。在新的版本里，正常情况只需直接通过，不会产生任何分支。

更多的进程

Linux 2.4 几乎消除了对同时可以运行的进程数目的固有限制。唯一剩下来的硬编码（hard-coded）限制就是 PID 的最大数目了（不要忘记 PID 可被共享）。这个改进使得 Linux 能够更好的适合于高端（high-end）服务器应用程序，包括经常需要同时运行大量进程的 Web 服务。

在第 8 章中说明过，原有 4090 个任务的最高限度是受可以同时保存在全局描述表（GDT）里的任务状态段（TSS）和局部描述表（LDT）的最大数目所影响的：Linux 2.2 在 GDT 里为每个进程存储一个 TSS 和一个 LDT，而且 GDT 总共允许有最多 8192 个条目。Linux 2.4 通过存储 TSS 和 LDT 本身而不是把它们存储在 GDT 里的方法回避了这个限制。现在，GDT 只保持每 CPU 一个 TSS 和一个 LDT，而且把这些条目设置为在每个 CPU 上的当前运行任务所需要的信息。

虽然用于追踪每个 CPU 的空闲进程的 **init_tasks** 数组还存在，但是 **task** 数组（26150 行）现在却没有了。

日益进步的 SMP 支持

每一个锁都是一个 SMP 机器可以被有效地转化为 UP 机器的地方：正等待一个锁的 CPU 对系统的整体性能没有任何贡献。因此，改善 SMP 性能的主要方法就是通过减小锁的作用域或者完全消除它们来增加并行程序的运行机会。

Linux 2.0 只有一个单独的全局内核锁，所以每次只有一个 CPU 可以在内核里执行。Linux 2.2 把这个全局内核锁使用的大部分地方都用更小的、子系统专用的锁来替代了，其

中的一些我们在第 10 章中已经见到过。Linux 2.4 继续了这个趋势，它把可能减小其作用域的锁都作了进一步的分割。比如现在每个等待队列就有一个锁，而不是所有的等待队列才有唯一的一个锁。