调度策略对它们进行调度。最后，信号处理的代码也属于进程管理部件。

尽管这三个部件在图中被分开，实际上它们高度相互依赖。文件系统一般通过块设备进行文件访问。然而，为了隐藏磁盘读取的严重延迟，文件被复制到内存中的页缓存中。有些文件甚至可能是动态创建的并且只在内存中存在，比如提供运行时资源使用情况的文件。另外，当需要清空一些页时，虚拟存储系统可能依靠一个磁盘分区或者文件内的交换区来备份内存的一部分，因此依赖于I/O部件。当然，还存在着很多其他的组件之间的相互依赖。

除了内核内的静态部件外，Linux支持动态可装载模块。这些模块可以用来补充或者替换缺省的设备驱动程序、文件系统、网络或者其他内核代码。在图10-3中没有显示这些模块。

最后，处在最顶层的是到内核的系统调用接口。所有系统调用都来自这里，其导致一个陷阱，并将系统从用户态转换到受保护的内核态，继而将控制权交给上述的内核部件之一。

10.3 Linux中的进程

前面的几个小节是从键盘的角度来看待Linux，也就是说以用户在xterm窗口中所见的内容来看待Linux。我们给出了常用的shell命令和标准应用程序作为例子。最后，以一个对Linux系统结构的简要概括作为结尾。现在，让我们深入到系统内核，更仔细地研究Linux系统所支持的基本概念，即进程、内存、文件系统和输入/输出。这些概念非常重要，因为系统调用（到操作系统的接口）将对这些概念进行操作。举个例子来说，Linux系统中存在着用来创建进程和线程、分配内存、打开文件以及进行输入/输出操作的系统调用。

遗憾的是，由于Linux系统的版本非常之多，各个版本之间均有不同。在这一章里，我们将摒弃着眼于某一个Linux版本的方法，转而强调各个版本的共通之处。因此，在某些小节中（特别是涉及实现方法的小节），这里讨论的内容不一定同样适用于每个Linux版本。

10.3.1 基本概念

Linux系统中主要的活动实体就是进程。Linux进程与我们在第2章所学的经典顺序进程极为相似。每个进程执行一段独立的程序并且在进程初始化的时候拥有一个独立的控制线程。换句话说，每一个进程都拥有一个独立的程序计数器，用这个程序计数器可以追踪下一条将要被执行的指令。一旦进程开始运行，Linux系统将允许它创建额外的线程。

由于Linux是一个多道程序设计系统，因此系统中可能会有多个彼此之间相互独立的进程在同时运行。而且，每一个用户可以同时开启多个进程。因此，在一个庞大的系统里，可能有成百个甚至上千个进程在同时运行。事实上，在大多数单用户的工作站里，即使用户已经退出登录，仍然会有很多后台进程，即守护进程（daemon），在运行。在系统启动的时候，这些守护进程就已经被shell脚本开启（在英语中，“daemon”是“demon”的另一种拼写，而demon是指一个恶魔）。

计划任务（cron daemon）是一个典型的守护进程。它每分钟运行一次来检查是否有工作需要它完成。如果有工作要做，它就会将之完成，然后进入休眠状态，直到下一次检查时刻来到。

在Linux系统中，你可以把在未来几分钟、几个小时、几天甚至几个月会发生的事件列成时间表，所以这个守护进程是非常必要的。举个例子来说，假定一个用户在下周二的三点钟要去看牙医，那么他就可以在计划任务的数据库里添加一条记录，让计划任务来提醒他，比如说，在两点半的时候。接下来，当相应的时间到来的时候，计划任务意识到有工作需要它来完成，就会运行起来并且开启一个新的进程来执行提醒程序。

计划任务也可以执行一些周期性的活动，比如说在每天凌晨四点的时候进行磁盘备份，或者是提醒健忘的用户每年10月31号的时候需要为万圣节储备一些好吃的糖果。当然，系统中还存在其他的守护进程，他们接收或发送电子邮件、管理打印队列、检测内存中是否有足够的空闲页等。在Linux系统中，守护进程可以直接实现，因为它不过是与其他进程无关的另一个独立的进程而已。

在Linux系统中，进程通过非常简单的方式创建。系统调用fork将会创建一个与原始进程完全相同的进程副本。调用fork函数的进程称为父进程，新的进程称为子进程。父进程和子进程都拥有自己的私有内存映像。如果在调用fork函数之后，父进程修改了属于它的一些变量，这些变化对于子进程来说是不可见的，反之亦然。