sigaction系统调用也可以用来忽略一个信号，或者恢复为一个杀死进程的缺省操作。

敲击DEL键并不是发送信号的惟一方式。系统调用kill允许一个进程给它相关的进程发送信号。选择“kill”作为这个系统调用的名字其实并不是十分贴切，因为大多数进程发送信号给别的进程只是为了信号能够被捕捉到。

对于很多实时应用程序，在一段特定的时间间隔之后，一个进程必须被打断，系统会转去做一些其他的事情，比如说在一个不可信的信道上重新发送一个可能丢失的数据包。为了处理这种情况，系统提供了alarm系统调用。这个系统调用的参数规定了一个以秒为单位的时间间隔，这个时间间隔过后，一个名为SIGALRM的信号会被发送给进程。一个进程在某一个特定的时刻只能有惟一一个未处理的警报。如果alarm系统调用首先以10秒为参数被调用，3秒钟之后，又以20秒为参数被调用，那么只会生成一个SIGALRM信号，这个信号生成在第二次调用alarm系统调用的20秒之后。第一次alarm系统调用设置的信号被第二次alarm系统调用取消了。如果alarm系统调用的参数为0，任何即将发生的警报信号都会被取消。如果没有捕捉到警报信号，将会采取默认的处理方式，收取信号的进程将会被杀死。从技术角度来讲，警报信号是可以忽略的，但是这样做毫无意义。

有些时候会发生这样的情况，在信号到来之前，进程无事可做。比如说，考虑一个用来测试阅读速度和理解能力的计算机辅助教学程序。它在屏幕上显示一些文本然后调用alarm函数于30秒后生成一个警报信号。当学生读课文的时候，程序就无事可做。它可以进入空循环而不做任何事情，但是这样一来就会浪费其他后台程序或用户急需的CPU时间。一个更好的解决办法就是使用pause系统调用，它会通知Linux系统将本进程挂起直到下一个信号到来。

10.3.3 Linux中进程与线程的实现

Linux系统中的一个进程就像是一座冰山：你所看见的不过是它露出水面的部分，而很重要的一部分隐藏在水下。每一个进程都有一个运行用户程序的用户模式。但是当它的某一个线程调用系统调用之后，进程会陷入内核模式并且运行在内核上下文中，它将使用不同的内存映射并且拥有对所有机器资源的访问权。它还是同一个线程，但是现在拥有更高的权限，同时拥有自己的内核堆栈以及内核程序计数器。这几点非常重要，因为一个系统调用可能会因为某些原因陷入阻塞态，比如说，等待一个磁盘操作的完成。这时程序计数器和寄存器内容会被保存下来使得不久之后线程可以在内核模式下继续运行。

在Linux系统内核中，进程通过数据结构task_struct被表示成任务（task）。不像其他的操作系统会区别进程、轻量级进程和线程，Linux系统用任务的数据结构来表示所有的执行上下文。所以，一个单线程的进程只有一个任务数据结构，而一个多线程的进程将为每一个用户级线程分配一个任务数据结构。最后，Linux的内核是多线程的，并且它所拥有的是与任何用户进程无关的内核级线程，这些内核级线程执行内核代码。稍后，本节会重新关注多线程进程（一般的讲，就是线程）的处理方式。

对于每一个进程，一个类型为task_struct的进程描述符是始终存在于内存当中的。它包含了内核管理全部进程所需的重要信息，如调度参数、已打开的文件描述符列表等。进程描述符从进程被创建开始就一直存在于内核堆栈之中。

为了与其他UNIX系统兼容，Linux还通过进程标识符（PID）来区分进程。内核将所有进程的任务数据结构组织成一个双向链表。不需要遍历这个链表来访问进程描述符，PID可以直接被映射成进程的任务数据结构所在的地址，从而立即访问进程的信息。

任务数据结构包含非常多的分量。其中一些分量包含指向其他数据结构或段的指针，比如说包含关于已打开文件的信息。有些段只与进程用户级的数据结构有关，当用户进程没有运行的时候，它们是不被关注的。所以，当不需要它们的时候，这些段可以被交换出去或重新分页以达到不浪费内存的目的。举个例子，尽管对于一个进程来说，当它被交换出去的时候，可能会有其他进程给它发送信号，但是这个进程本身却不会要求读取一个文件。正因为如此，关于信号的信息才必须永远保存在内存里，即使这个进程已经不在内存当中了。换句话说，关于文件描述符的信息可以被保存在用户级的数据结构里，当进程存在于内存当中并且可以执行的时候，这些信息才需要被调入内存。

进程描述符的信息包含以下几大类：

1) 调度参数。进程优先级，最近消耗的CPU时间，最近睡眠的时间。上面几项内容结合在一起决定