1.6.4 各种系统调用

有各种的系统调用。这里介绍系统调用中的一部分。chdir调用改变当前的工作目录。在调用

```
chdir("/usr/ast/test");
```

之后，打开xyz文件，会打开/usr/ast/test/xyz。工作目录的概念消除了总是键入（长）绝对路径名的需要。

在UNIX中，每个文件有一个保护模式。该模式包括针对所有者、组和其他用户的读-写-执行位。chmod系统调用可以改变文件的模式。例如，要使一个文件对除了所有者之外的用户只读，可以执行

```
chmod("file",0644);
```

kill系统调用供用户或用户进程发送信号用。若一个进程准备好捕捉一个特定的信号，那么，在信号到来时，运行一个信号处理程序。如果该进程没有准备好，那么信号的到来会杀掉该进程（此调用名称的由来）。

POSIX定义了若干处理时间的过程。例如，time以秒为单位返回当前时间，0对应着1970年1月1日午夜（从此日开始，没有结束）。在一台32位字的计算机中，time的最大值是$2^{32}-1$秒（假设是无符号整数）。这个数字对应136年多一点。所以在2106年，32位的UNIX系统会发狂，与在2000年造成对世界计算机严重破坏的知名的Y2K问题是类似的。如果读者现在有32位UNIX系统，建议在2106年之前的某时刻更换为64位的系统。

1.6.5 Windows Win32 API

到目前为止，我们主要讨论的是UNIX系统。现在简要地考察Windows。Windows和UNIX的主要差别在于编程方式。一个UNIX程序包括做各种处理的代码以及从事完成特定服务的系统调用。相反，一个Windows程序通常是一个事件驱动程序。其中主程序等待某些事件发生，然后调用一个过程处理该事件。典型的事件包括被敲击的键、移动的鼠标、被按下的鼠标或插入的CD-ROM。调用事件处理程序处理事件，刷新屏幕，并更新内部程序状态。总之，这是与UNIX不同的程序设计风格，由于本书专注于操作系统的功能和结构，这些程序设计方式上的差异就不过多涉及了。

当然，在Windows中也有系统调用。在UNIX中，系统调用（如read）和系统调用所使用的库过程（如read）之间几乎是一一对应的关系。换句话说，对于每个系统调用，差不多就涉及一个被调用的库过程，如图1-17所示。此外，POSIX有约100个过程调用。

在Windows中，情况就大不相同了。首先，库调用和实际的系统调用是几乎不对应的。微软定义了一套过程，称为应用编程接口（Application Program Interface，Win32 API），程序员用这套过程获得操作系统的服务。从Windows 95开始的所有Windows版本都（或部分）支持这个接口。由于接口与实际的系统调用不对应，微软保留了随着时间（甚至随着版本到版本）改变实际系统调用的能力，防止使已有的程序失效。由于Windows 2000、Windows XP和Windows Vista中有许多过去没有的新调用，所以究竟Win32是由什么构成的，这个问题的答案仍然是含混不清的。在本节中，Win32表示所有Windows版本都支持的接口。

Win32 API调用的数量是非常大的，数量有数千个。此外，尽管其中许多确实涉及系统调用，但有一大批Win32 API完全是在用户空间进行。结果，在Windows中，不可能了解哪一个是系统调用（如由内核完成），哪一个只是用户空间中的库调用。事实上，在某个版本中的一个系统调用，会在另一个不同版本中的用户空间中执行，或者相反。当我们在本书中讨论Windows的系统调用时，将使用Win32过程（在合适之处），这是因为微软保证：随着时间流逝，Win32过程将保持稳定。但是读者有必要记住，它们并不全都是系统调用（即陷入到内核中）。

Win32 API中有大量的调用，用来管理视窗、几何图形、文本、字型、滚动条、对话框、菜单以及GUI的其他功能。为了使图形子系统在内核中运行（某些Windows版本中确实是这样，但不是所有的版本），需要系统调用，否则只有库调用。在本书中是否应该讨论这些调用呢？由于它们并不是同操作系统的功能相关，我们还是决定不讨论它们，尽管它们会在内核中运行。对Win32 API有兴趣的读者应该参阅一些书籍中的有关内容，（例如，Hart，1997；Rector和Newcomer，1997；Simon，1997）。

我们在这里介绍所有的Win32 API，不过这不是我们关心问题的所在，所以我们做了一些限制，只将那些与图1-18中UNIX系统调用大致对应的Windows调用列在图1-23中。