行。虽然依旧比较麻烦，但比起在没有任何程序的状态下进行开发，工作量得到了很大的缓解（图 9-1）。

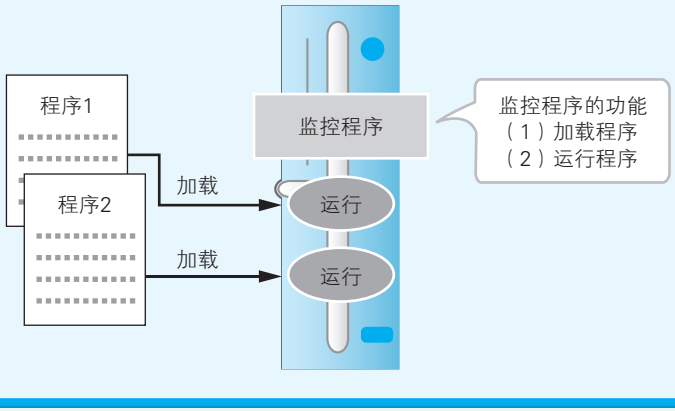

图 9-1　监控程序可以说是操作系统的原型

随着时代的发展，人们在利用监控程序编写程序的过程中，发现很多程序都有共通的部分。例如，通过键盘输入文字数据、往显示器输出文字数据等。这些处理，在任何程序下都是一样的。而如果每编写一个新的程序都要记述相同的处理的话，那真的是太浪费时间了。因此，基本的输入输出部分的程序就被追加到了监控程序中。初期的操作系统就这样诞生了（图 9-2）。

之后，随着时代的进一步发展，开始有更多的功能被追加到监控程序中，比如，为了方便程序员的硬件控制程序、编程语言处理器（汇编、编译、解析）以及各种实用程序等，结果就形成了和现在相差不大的操作系统。因此，操作系统本身并不是单独的程序，而是多个程序的集合体（图 9-3）。

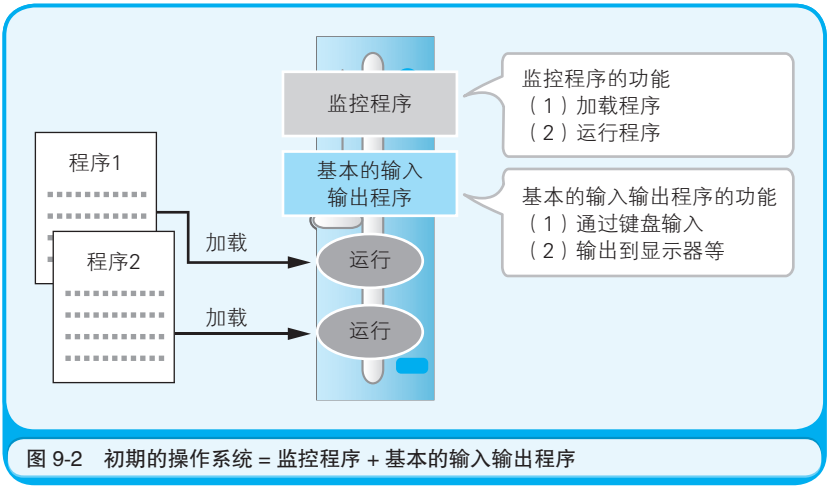

图 9-2 初期的操作系统 = 监控程序 + 基本的输入输出程序

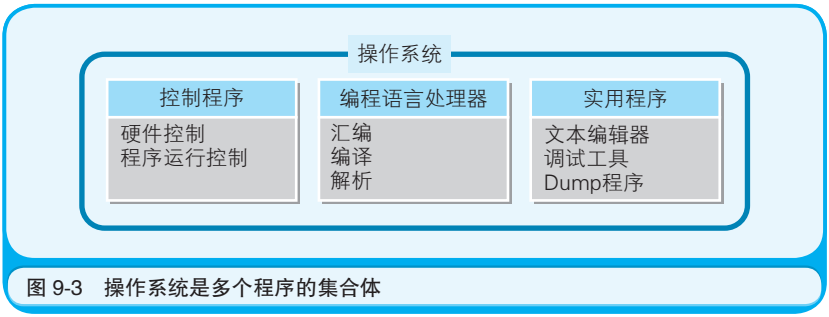

图 9-3 操作系统是多个程序的集合体

9.2 要意识到操作系统的存在

这里，我希望制作应用的程序员们意识到一点，那就是你们制作的不是硬件，而是利用操作系统功能的应用。虽然对程序员来说，掌握硬件的基本知识是必需的，不过，在操作系统诞生以后，就没有必要再编写直接控制硬件的程序了。这样一来，制作应用的程序员就逐渐同硬件隔离开来了。也就是说，程序员是很少关注现实世界（硬件）的。

由于操作系统诞生后，程序员无需再考虑硬件的问题，因此程序员的数量也增加了。哪怕是自称“对硬件一窍不通”的人，也可能会制作出一个有模有样的应用。不过，要想成为一个全面的程序员，有一点需要清楚的是，掌握基本的硬件知识，并借助操作系统进行抽象化，可以大大提高编程效率。否则，遇到问题时，你就无法找到解决办法。操作系统确实为程序员提供了很多方便。不过，仅仅享受方便是不行的，还要了解为什么自己能够这么方便。了解了这一点，就可以尽情地享受方便了。

下面就来看一下操作系统是如何给开发人员带来便利的。代码清单 9-1 表示的是，在 Windows 操作系统下，用 C 语言制作一个具有表示当前时间功能的应用。time() 是用来取得当前日期和时间的函数，printf() 是用来在显示器上显示字符串的函数。程序的运行结果如图 9-4 所示。

代码清单 9-1　表示当前时间的应用

```
#include <stdio.h>
#include <time.h>

void main() {
    // 保存当前日期和时间信息的变量
    time_t tm;

    // 取得当前的日期和时间
    time(&tm);

    // 在显示器上显示日期和时间
    printf("%s\n", ctime(&tm));
}
```

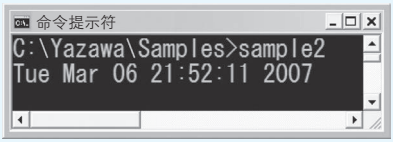

图 9-4 代码清单 9-1 的运行结果

运行代码清单 9-1 的应用时，硬件的受控过程如下所示。

（1）通过 time_t tm;，为 time_t 类型的变量申请分配内存空间。

（2）通过 time(&tm);，将当前的日期和时间数据保存到变量的内存空间中。

（3）通过 printf("%s\n",ctime(&tm));，把变量内存空间的内容输出到显示器上。

应用的可执行文件指的是，计算机的 CPU 可以直接解释并运行的本地代码。不过这些代码是无法直接控制计算机中配置的时钟 IC 及显示器用的 I/O 等硬件的。那么，为什么代码清单 9-1 的应用能够控制硬件呢?

在操作系统这个运行环境下，应用并不是直接控制硬件，而是通过操作系统来间接控制硬件的。变量定义中涉及的内存的申请分配，以及 time() 和 printf() 这些函数的运行结果，都不是面向硬件而是面向操作系统的。操作系统收到应用发出的指令后，首先会对该指令进行解释，然后会对时钟 IC（实时时钟[①]）和显示器用的 I/O 进行控制。

① 计算机中都安装有保存日期和时间的实时时钟（Real-time clock）。本节中提到的时钟 IC 就是指该实时时钟。

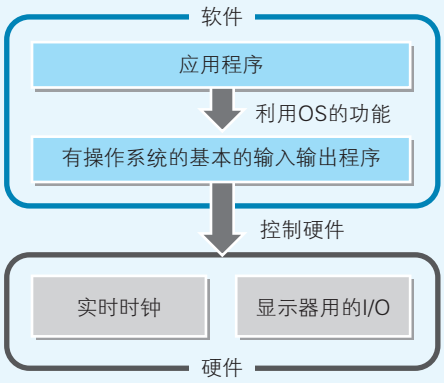

图 9-5　应用程序经过 OS 间接地控制硬件

9.3　系统调用和高级编程语言的移植性

操作系统的硬件控制功能，通常是通过一些小的函数集合体的形式来提供的。这些函数及调用函数的行为统称为**系统调用**（system call），也就是应用对操作系统（system）的功能进行调用（call）的意思。在前面的程序中用到了 time() 及 printf() 等函数，这些函数内部也都使用了系统调用。这里之所以用“内部”这个词，是因为在 Windows 操作系统中，提供返回当前日期和时刻，以及在显示器中显示字符串等功能的系统调用的函数名，并不是 time() 和 printf()。系统调用是在 time() 和 printf() 函数的内部执行的。大家可能会认为这个方法有些绕，不过这是有原因的。

C 语言等高级编程语言并不依存于特定的操作系统。这是因为人们希望不管是 Windows 还是 Linux，都能使用几乎相同的源代码。因此，高级编程语言的机制就是，使用独自的函数名，然后再在编译时将其转换成相应操作系统的系统调用（也有可能是多个系统调用的组合）。

也就是说，用高级编程语言编写的应用在编译后，就转换成了利用系统调用的本地代码（图 9-6）。

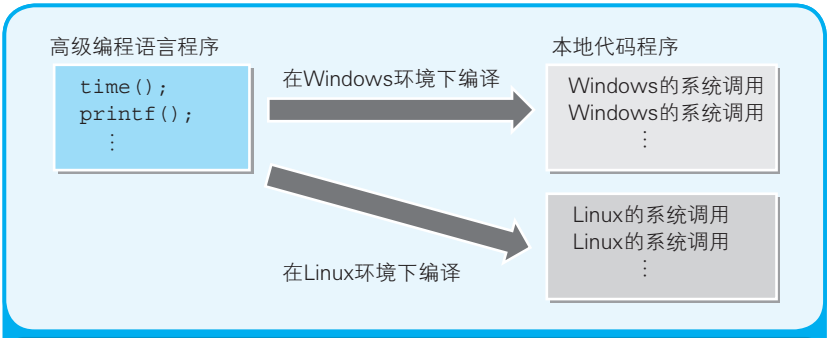

图 9-6 高级编程语言的函数调用在编译后变成了系统调用

在高级编程语言中，也存在可以直接调用系统调用的编程语言。不过，利用这种方式做成的应用，移植性[①]并不友好（也俗称为有恶意行为的应用）。例如，直接调用 Windows 系统调用的应用，在 Linux 上显然是无法运行的。

9.4 操作系统和高级编程语言使硬件抽象化

通过使用操作系统提供的系统调用，程序员就没必要编写直接控制硬件的程序了。而且，通过使用高级编程语言，有时甚至也无需考虑系统调用的存在。这是因为操作系统和高级编程语言能够使硬件抽象化。这是个非常了不起的处理。

下面就让我们来看一下硬件抽象化的具体实例。代码清单 9-2 是用 C 语言编写的往文件中写入字符串的应用。fopen() 是用来打开文件的

① 移植性指的是同样的程序在不同操作系统下运行时需要花费的时间等，费时越少说明移植性越好。

函数，fputs() 是用来往文件中写入字符串的函数，fclose() 是用来关闭文件的函数[①]。

代码清单 9-2　往文件中写入字符串的应用

```
#include <stdio.h>

void main() {
    // 打开文件
    FILE *fp = fopen("MyFile.txt", "w");

    // 写入文件
    fputs(" 你好 ", fp);

    // 关闭文件
    fclose(fp);
 }
```

该应用在编译运行后，MyFile.txt 文件中就会被写入“你好”字符串。文件是操作系统对磁盘媒介空间的抽象化。就如第 5 章中介绍的那样，作为硬件的磁盘媒介，就如同树木的年轮一样，被划分为了多个扇区，并以扇区为单位对磁盘进行读写。如果直接对硬件进行操作的话，那就变成了通过向磁盘用的 I/O 指定扇区位置来对数据进行读写了。

但是，在代码清单 9-2 的程序中，扇区根本没有出现过。传递给 fopen() 函数的参数，是文件名 "MyFile.txt" 和指定文件写入的 "w"。传递给 fputs() 的参数，是往文件中写入的字符串 " 你好 " 和 fp。传递给 fclose 的参数，也仅仅是 fp。也就是说，磁盘媒介的读写采用了文件这个概念，将整个流程抽象化成了打开文件用的 fopen()、写入文件用的 fputs()、关闭文件用的 fclose()（图 9-7）。

① fopen()、fputs()、fclose() 这些函数名分别是 file open、file put string、file close 的略称。string 是字符串的意思。

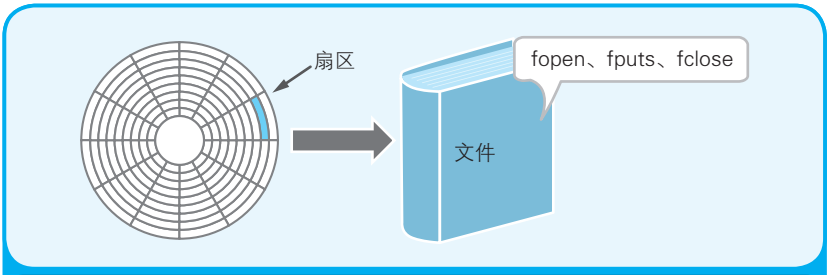

图 9-7 物理上的磁盘媒体被抽象化成了文件

下面让我们来看一下代码清单 9-2 中变量 *fp* 的功能。变量 *fp* 中被赋予的是 fopen() 函数的返回值。该值称为**文件指针**。应用打开文件后，操作系统就会自动申请分配用来管理文件读写的内存空间。这个内存空间的地址可以通过 fopen() 函数的返回值获得。用 fopen() 打开文件后，接下来就是通过指定文件指针来对文件进行操作。正因为如此，fputs() 及 fclose() 的参数中都指定了文件指针（变量 *fp*）。

至于用来管理文件读写的内存空间的内容实际在哪里，程序员则没必要关注。只要能意识到“用来操作磁盘媒介的某些信息在某个地方存储着”，就可以制作应用了。

9.5 Windows 操作系统的特征

考虑到大多数读者使用的都是 Windows 操作系统，这里我们就以 Windows 为例，来详细讲解操作系统的具体功能。Windows 操作系统的主要特征如下所示。

（1）32 位操作系统（也有 64 位版本）
（2）通过 API 函数集来提供系统调用
（3）提供采用了图形用户界面的用户界面

（4）通过 WYSIWYG[①] 实现打印输出

（5）提供多任务功能

（6）提供网络功能及数据库功能

（7）通过即插即用实现设备驱动的自动设定

这里只列出了对程序员有意义的一些特征。接下来将依次对 Windows 操作系统的特征，以及其对编程的影响进行说明。

（1）32 位操作系统

虽然现在的 Windows 也有 64 位版本，但一般广泛普及的还是 32 位版本。这里的 32 位表示的是处理效率最高的数据大小。Windows 处理数据的基本单位是 32 位。习惯在以前的 MS-DOS 等 16 位操作系统下编程的程序员，可能不太愿意使用 32 位的数据类型。因为他们认为处理 32 位的数据，要比处理 16 位的数据更花时间。确实，在 16 位操作系统中处理 32 位的数据时，因为要处理两次 16 位的数据，所以会多花一些时间。而如果是 32 位操作系统的话，那么只需要 1 次就可以完成 32 位的数据的处理了。所以说，凡是在 Windows 上运行的应用，都可以毫无顾虑地尽可能地使用 32 位的数据。

例如，用 C 语言来处理整数数据时，有 8 位的 char 类型、16 位的 short 类型，以及 32 位的 long 类型（还有 int 类型）三个选项。使用位数大的 long 类型的话，虽然内存及磁盘的开销较大，但应用的运行速度并不会下降。这在其他编程语言中也是同样的。

① WYSIWYG 是 What You See Is What You Get 的略写。意思是，显示器上显示的文本及图形等（What You See），是（Is）可以原样输出到打印机上打印（What You Get）的。

（2）通过 API 函数集来提供系统调用

Windows 是通过名为 API 的函数集来提供系统调用的。API 是联系作成应用的程序员和操作系统之间的接口。所以称为 API（Application Programming Interface，应用程序接口）。

当前主流的 32 位版 Windows API 也称为 Win32 API。之所以这样命名，是为了便于和以前的 16 位版的 Win16 API，以及更先进的 64 位版的 Win64 API 区分开来。Win32 API 中，各函数的参数及返回值的数据大小，基本上都是 32 位。

API 通过多个 DLL 文件来提供。各 API 的实体都是用 C 语言编写的函数。因而，C 语言程序的情况下，AIP 的使用更加容易。截至到现在，本书示例程序中用到的 API 中都有 MessageBox()。MessageBox() 被保存在 Windows 提供的 user32.dll 这个 DLL 文件中。

（3）提供采用了 GUI 的用户界面

GUI（Graphical User Interface，图形用户界面）指的是通过点击显示器中显示的窗口及图标等即可进行可视化操作的用户界面。对用户来说，GUI 是图形、鼠标，但对程序员来说，GUI 并不仅是这些。这是因为想要作成一个实现 GUI 的应用，并不是一件容易的事情。曾经有一首俳句是这样的：“GUI，用的时候是天堂，做的时候是地狱”，大家可以想象它的难度了吧。

之所以这样困难，是因为在 GUI 中用户按照怎样的顺序操作是无法确定的。例如，图 9-8 是 Web 浏览器（Internet Explorer 7）的一个窗口。通过多个标签页的切换，就可以进行各种项目设定。从 Web 浏览器的用户角度来说，这样的窗口不仅使用方便，操作也简单，但对负责开发的程序员来说，却决不是简单的事情。