在这里，给读者也布置一个任务，在 MSDN 查找“Win32 Application”和“Win32 Console Application”的说明，弄清楚它们之间的区别。

出现问题的解决办法：

（1）当然是重新建一个工程，选择正确的工程类型（相信读者看了下面的解决方法后，不会选择这一种 ☺）。

（2）在 VC++集成开发环境中，单击菜单【Project】→【Settings】，选择“Link”选项卡，在最下方的“Project Options”列表框中找到“/subsystem:console”后，将其删除，或者修改为“/subsystem:windows”，单击“确定”按钮。重新编译运行程序。

1.6　消息循环的错误分析

有不少初学者学完第 1 章后，编写了下面的代码：

```
…
HWND hwnd;
    hwnd=CreateWindow(…);
…
    MSG msg;
    while(GetMessage(&msg,hwnd,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
…
```

注意代码中以粗体显示的部分。这段代码基于这样一个想法：第 1 章的程序只有一个窗口，而我们前面说了 GetMessage 函数的 hWnd 参数是用于指定接收属于哪一个窗口的消息，于是不少人就在消息循环中为 GetMessage 函数的 hWnd 参数指定了 CreateWindow 函数返回的窗口句柄。

读者可以用上述代码中的消息循环部分替换 1.5 节代码中的消息循环部分，然后运行程序，关闭程序。你会发现你的机器变慢了，同时按下键盘上的 Ctrl + Alt + Delete 键，启动 Windows 的任务管理器，切换到“进程”选项卡，单击“CPU”项进行排序，你会发现如图 1.7 所示的情况。

从图 1.7 中可以看到，WinMain.exe 的 CPU 占用率接近 100，难怪机器“变慢了”。那么这是什么原因呢？实际上这个问题的答案在 MSDN 中就可以找到，并且就在 GetMessage 函数的说明文档中。不少初学者在遇到问题时，首先是头脑一片空白，接着就去找人求助，这种思想用在程序开发的学习中，没有什么好处。笔者经常遇到学员问问题，结果有不少问题的答案在 MSDN 关于某个函数的解释中就可看到（由于显示器的限制，有的答案需要滚动窗口才能看到 ☺）。所以在这里，笔者也建议读者遇到问题一定要记得查看 MSDN，学会使用 MSDN 并从中汲取知识，将使你受用无穷。

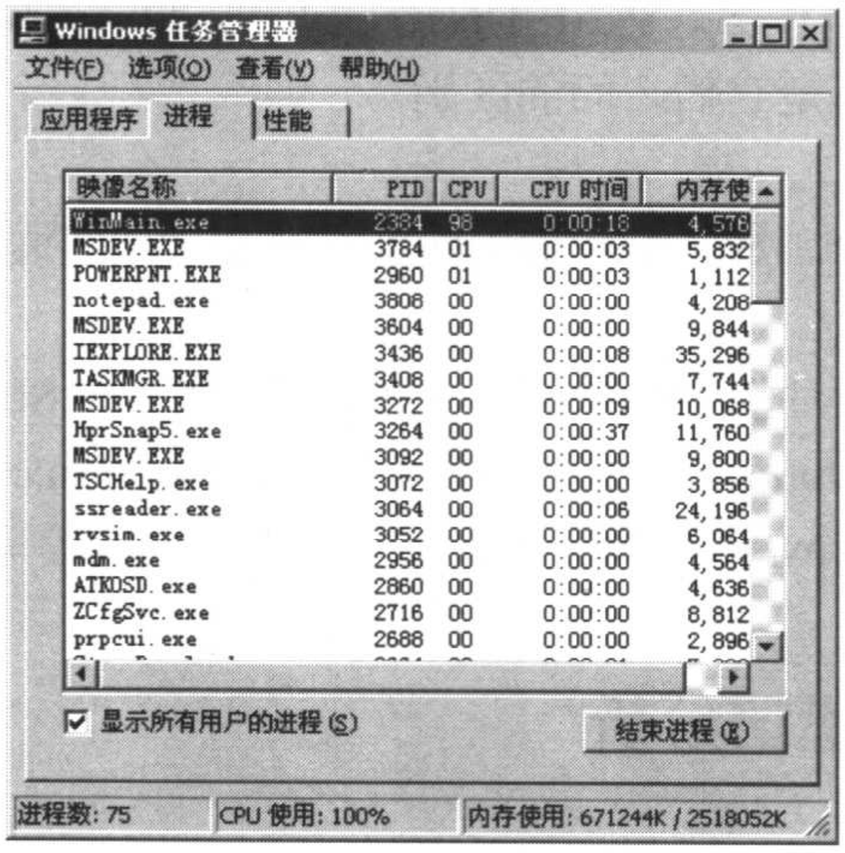

图 1.7　WinMain.exe 的 CPU 占用率接近 100

回到正题，在 1.4.3 节介绍 GetMessage 函数时，曾说过如果 hWnd 参数是无效的窗口句柄或 lpMsg 参数是无效的指针时，GetMessage 函数将返回–1。当我们关闭窗口时，调用了 DestroyWindow 来销毁窗口，由于窗口被销毁了，窗口的句柄当然也就是无效的句柄了，那么 GetMessage 将返回–1。在 C/C++语言中，非 0 即为真，由于窗口被销毁，句柄变为无效，GetMessage 总是返回–1，循环条件总是为真，于是形成了一个死循环，机器当然就“变慢了”。☺

在 MSDN 关于 GetMessage 函数的说明文档中给出了下面的代码：

```
BOOL bRet;

    while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
    {
        if (bRet == -1)
        {
            // handle the error and possibly exit
        }
        else
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
```

针对我们这个问题，可以修改上述代码如下：

```
…
    HWND hwnd;
    hwnd=CreateWindow(…);
    …
    MSG msg;
```

```
BOOL bRet;

while( (bRet = GetMessage( &msg, hwnd, 0, 0 )) != 0)
{
    if (bRet == -1)
    {
        // handle the error and possibly exit
        return -1;
    }
    else
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
...
```

读者可以再次运行修改后的程序，看看运行的结果。

1.7 变量的命名约定

由于 Windows 程序一般很长，编程人员在一段时间后自己都有可能忘记所定义的变量的含义。为了帮助大家记忆与区分变量，微软公司创建了 Windows 的命名约定，称之为匈牙利表示法（Hungarian notation）。匈牙利表示法提供了一组前缀字符，如表 1.1 所示，这些前缀也可以组合起来使用。

表 1.1 匈牙利表示法

前 缀	含 义
a	数组
b	布尔值（int）
by	无符号字符（字节）
c	字符（字节）
cb	字节记数
rgb	保存 RGB 颜色值的长整型
cx,cy	短整型（计算 x,y 的长度）
dw	无符号长整型
fn	函数
h	句柄
i	整数（integer）
m_	类的数据成员
n	短整型或整型
np	近指针
p	指针

续表

前　缀	含　义
l	长整型
lp	长指针
s	字符串
sz	以零结束的字符串
tm	正文大小
w	无符号整型
x,y	无符号整型（表示 x 或 y 的坐标）

1.8 小结

这一章详细介绍了 Windows 程序运行的内部机制。建议读者花点时间把本章的内容消化理解了。在学习本章内容的时候，可以结合配套光盘中的 VC 视频第 1 课一起学习。如果读者没有完全掌握本章的内容，也不要灰心，这里涉及到的一些内容，在后续章节中还会讲到。学习 VC++的路程是艰苦的，必须具有一定的毅力并不断努力，才有可能精通 VC++编程。

下面我们再次为读者总结一下创建一个 Win32 应用程序的步骤。

1. 编写 WinMain 函数，可以在 MSDN 上查找并复制。
2. 设计窗口类（WNDCLASS）。
3. 注册窗口类。
4. 创建窗口。
5. 显示并更新窗口。
6. 编写消息循环。
7. 编写窗口过程函数。窗口过程函数的语法，可通过 MSDN 查看 WNDCLASS 的 lpfnWndProc 成员变量，在这个成员的解释中可查到。