Windows 编程经验

韩振宇　（清华大学自动化系）

本文就学习 Windows 编程的三个阶段，结合几个实例介绍笔者的一些经验，供大家参考。

一、避开陷阱

学习编制 Windows 程序的第一步是理解，包括理解 Windows 的消息机制、Windows 函数的使用方法以及如何正确运用 C 函数。但只学习“手册”、“指南”有时会只知其一不知其二，弄不好还会落入陷阱，出现事与愿违的结果。下面有三个实例，借以说明“想当然”编程是不行的。

1. WM-LBUTTONDBLCLK 与 WM-LBUTTONUP 消息

考虑一个程序窗口处理鼠标左键释放消息 WM-LBUTTONUP 的过程中要弹出一个消息框，简化程序如下：

```
switch (message)
{
...
case WM-LBUTTONUP:
    MessageBox ( hWnd," Text"," Title", MB-OK-CANCEL);
    break;
...
}
```

用户希望在鼠标左键释放时弹出一个消息框。从逻辑上讲此程序没有错误，但实际结果可能出人意料。消息框确实如愿弹出，但用鼠标左键双击消息框左上角的控制菜单时，消息框不能正常关闭！实际上，双击后，前一个消息框确实关闭了，但一个新的消息框又马上产生，总不能做到完全关闭。这可不是所希望的结果。

原因很简单。一次双击鼠标左键系统将产生四个消息：WM-LBUTTONDOWN、WM-LBUTTONUP、WM-LBUTTONDBLCLK 以及 WM-LBUTTONUP 消息，但是最后两个消息却分别发给了不同窗口！WM-LBUTTONDBLCLK 发给了消息框，消息框关闭并返回 IDCANCEL；最后一个 WM-LBUTTONUP 消息没有发给消息框，也没有被取消，而是发给了位于鼠标下面的主窗口。主窗口过程处理 WM-LBUTTONUP 时又会产生新的消息框。即一次双击鼠标操作，在关闭旧消息框的同时又创建了新的消息框。

需要说明的是在上面实验中，用鼠标点取消息框的 OK、CANCEL 按键或控制菜单的 Close 命令，消息框可正常关闭。此时 WM-LBUTTONUP 消息发给了消息框而不会遗留给主窗口，因此不会重复弹出新的消息框了。

实验很简单，但却提醒大家开发 Windows 程序过程中在处理 WM-LBUTTONUP 消息时一定要十分注意，千万不要以为一个窗口在收到 WM-LBUTTONUP 消息之前一定会先收到 WM-LBUTTONDOWN 或 WM-LBUTTONDBLCLK 消息。要想到，WM-LBUTTONUP 消息可能是其它窗口甚至是其它应用程序遗留的。例如，在 FileManager 中双击运行程序，双击操作的 WM-LBUTTONDBLCLK 消息被 FileManager 处理，而双击操作的 WM-LBUTTONUP 消息则遗留给了应用程序。

2. GetInstanceData 函数

GetInstanceData 函数用于将先前实例的数据拷贝到当前实例数据缓冲区中来。此函数原型如下：

```
int GetInstanceData(hinst, npbData, cbData)
HINSTANCE hinst：先前实例句柄
BYTE * npbData：当前实例数据缓冲区地址
int cbData：要传递的字节数
```

GetInstanceData 函数的一个重要应用是获取前一个实例的主窗口句柄。许多 Windows 应用程序 WinMain 函数执行时都要首先判断当前是否为第一个实例，若不是，则激活前一实例的主窗口，而不再运行新的实例，以保证一个程序只有一个实例运行。其中最关键的一步就是获取前一个实例的主窗口句柄。考虑下面的方法：

```
GetInstanceData(hPrevious, //前一实例句柄
(BYTE * )&hWndMain, //变量地址
sizeof(HWND)); //变量大小
```

许多 Windows 应用程序都是如此获得 hWndMain 句柄的，但若使用不当结果可能会事与愿违。该方法有一个非常重要的前提条件，即必须以小模式编译。如果用大模式编译会有什么后果呢？若上面语句在 WinMain 函数中，hWndMain 将得到错误结果，程序运行不正常。而若上面语句在一子程序中，则不但 hWndMain 得到错误结果，而且还会造成更严重的后果，使系统弹出“程序错误”提示框，指出该程序“caused a General Protection Fault”。

原因很简单。小模式下指针 npbData 为 2 字节，大模式下指针 npbData 变为 4 字节，函数调用时将造成参数错位，使 GetInstanceData 的参数“先前实例句柄”被“当前实例数据缓冲区地址”的 far 指针所覆盖，使 hWndMain 得到错误结果。而更严重的是，由于 GetInstanceData 函数是 PASCAL 类型，函数本身将负责

返回后的堆栈清除工作。该函数默认 npbData 为 near 类型，故只清除 6 字节，而大模式下参数共有 8 字节，GetInstanceData 函数返回后，将遗留 2 字节在堆栈中，造成堆栈错误。WinMain 函数出口要调整堆栈指针 SP，故问题不大，但一般函数则没有那么幸运，堆栈错位将造成程序跑飞，引起"Protection Fault"。

因此提醒大家一定要注意使用不同模式编译时指针大小不同可能引起的负作用。

3.mkdir 函数

编制 Windows 程序一般都离不开 C 函数，但有些 C 函数使用时不小心会造成问题。下面以 mkdir 函数为例。

一个安装程序运行过程中常常要由用户输入路径，并使用 mkdir 建立新目录。大家都知道在 DOS 下建目录时不允许路径中间有空格，如果"md a b"系统将显示出错信息"Too many parameters - b"，但是如果在程序中使用 mkdir("a b")建立目录竟然可以成功！真的在当前目录下建成了"a b"子目录。当然，该目录虽然建立了但却不能正常操作。

因此提醒大家注意，在使用该函数之前一定要做些准备工作，如滤除用户输入的路径中间的空格，以防止出错。

二、使用用户定义资源

学习编制 Windows 程序的第二步是发挥，即能够灵活运用系统提供的功能。下面以用户定义资源为例。

一个 Windows 应用程序一般都要或多或少与用户交流，因此显示一些信息是必不可少的。最常用的方法是在程序中使用 MessageBox 函数，由用户提供消息框文本字符串、标题字符串以及消息框式样。若这些内容都在 C 程序中提供则存在一些缺点：一是不可避免重复；二是占用程序数据段内存；三是不利于修改。如果能在资源文件中定义为用户资源，则不但使用方便，而且以上问题可迎刃而解。

附一定义了三个"REMINDER"型用户资源，每个资源包括三段数据，即消息框文本字符串、标题字符串以及消息框式样。可以发现，定义用户资源很简单。用户可以在资源文件中包含程序所要使用的所有消息框内容，既清晰又直观，且便于修改。需要注意的是定义用户资源有一些限制条件，如：资源号必须大于 255，字符串要以'\0'结束，等等。具体内容请参见"Windows 3.1 SDK"。

附二是 UserReminder 函数的源代码，该函数根据用户提供的资源标识获取数据，显示消息框，并返回用户按键。从中可以看出用户资源的使用是很简单的，只需调用 FindResource、LoadResource 和 LockResource 做些准备工作，而使用完以后也只要一个 FreeResource 调用即可。

三、为 Windows 增加消息

学习编制 Windows 程序的第三步是提高，即充分利用现有手段，增加系统不具备的功能，如状态条、活动图标、定制对话框等。下面谈谈如何为 Windows 增加消息。

不少人可能使用过 WORD 或中文之星等软件，它们都有一个特别的功能，即当鼠标移到按钮上时，程序将显示相应的"说明"，方便且直观。类似地，有些程序要求当鼠标移到窗口中某些特殊位置时系统有特定反应，而当鼠标移出时又恢复正常。此时，判断鼠标何时移入何时移出就非常重要。判断鼠标是否移入窗口比较容易，有诸如 WM-MOUSEMOVE 等消息可供使用。但是，鼠标何时移出则无法知道，造成虽然有时鼠标已经移出窗口，但显示信息却无法更新。解决的方法之一是通过建立一鼠标挂钩为系统增加一"鼠标移出窗口"消息。下面有三点说明：

1.鼠标挂钩的安装与撤销

安装鼠标挂钩可使用 SetWindowsHookEx 函数，而撤销鼠标挂钩可使用 UnhookWindowsHookEx 函数。为保证当鼠标移出应用程序时也能获得消息，应使用系统挂钩，需要编制动态链接库程序(DLL)。

2.新消息的注册

为系统加入新的"鼠标移出消息"，在动态链接库与 b 应用程序中都要加入下面语句：

WM-MOVEOUT = RegisterWindowMessage("WM-MOVEOUT");

在应用程序消息处理部分 WM-MOVEOUT 不能放在 case 语句中，而应放在 default 语句中使用 if 判断。

若只是自己使用，鼠标挂钩程序也可发送 WM-COMMAND 消息到对应窗口，而无需注册新消息。

3.判断条件

对于单一窗口判断鼠标是否移出时可使用下面形式：

```
if (hWndTest ! = hWndCur)
{
... //发送鼠标移出窗口消息
}
```

中文之星就是采用上面方法判断的。而对于含有控制的对话框判断鼠标是否移出时，最好同时判断当前窗口是否为该对话框的控制(子窗口)，若是则不必发消息，

以减少消息发送频度(参见附三)。

附一:用户定义资源文件

```
#include <windows.h>

#define REMINDER 256
#define Rem0 100
#define Rem1 101
#define Rem2 102

Rem0 REMINDER
BEGIN
    "Text. \0",
    "Title\0",
    MB-ICONINFORMATION | MB-OK
END

Rem1 REMINDER
BEGIN
    "Icon Stop and Yes No Cancel Button. \0",
    "Error Message 1\0",
    MB-ICONSTOP | MB-YESNOCANCEL
END

Rem2 REMINDER
BEGIN
    "SystemModal. \0",
    "Error Message 2\0",
   MB-ICONHAND | MB-SYSTEMMODAL | MB-ABORTRETRYIG-
NORE
END
```

附二:显示消息框函数

```
int UserReminder(HWND hWnd, int ReminderID)
{
    HINSTANCE hInstance; ///程序实例句柄
    HRSRC Resource; // 用户资源句柄
    HGLOBAL hRes; // 内存对象句柄
    char far *pRes; // 资源数据指针
    char far *pText; //消息框文本字符串指针
    //消息框标题字符串指针
    UINT uStyle; // 消息框式样
    int iRet; // 消息框函数返回值

    hInstance = GetWindowWord ( hWnd, GWW-HIN-
STANCE);
    Resource = FindResource(hInstance,
    MAKEINTRESOURCE(ReminderID),
    MAKEINTRESOURCE(REMINDER));
    hRes = LoadResource(hInstance, Resource);
    pRes = LockResource(hRes);
    if (! pRes) return 0;

    pText = pRes; // 消息框文本字符串指针
    while(*pRes++);
    pTitle = pRes; // 消息框标题字符串指针
    while(*pRes++);
    uStyle = *(UINT far*)pRes;// 消息框式样

    iRet = MessageBox(hWnd, pText, pTitle, uStyle);
    FreeResource(hRes);
    return iRet;
}
```

附三:鼠标挂钩 DLL 程序部分函数

```
/*****************************/
/* InforHook: 当鼠标移入待测窗口时, */
/* 通知 DLL 程序,检测鼠标何时移出该窗口 */
//*****************************/
void WINAPI InforHook(HWND hWnd)
{
    hWndTest = hWnd;
}

//*****************************/
/* MouseHookProc: 鼠标挂钩程序 */
/* 当鼠标移出 hWndTest 所指定的窗口时, */
/* 向该窗口发送 WM-MOVEOUT 消息 */
/*****************************/
LRESULT CALLBACK MouseHookProc(int code,
                              WPARAM wParam,
                              LPARAM lParam)
{
    HWND hWndCur;

    if (code >= 0 && hWndTest)
    {
        hWndCur = ((LPMOUSEHOOKSTRUCT)lParam) -
>hwnd;

        if ((hWndTest != hWndCur) &&
          (! IsChild(hWndTest, hWndCur)))
        {
        //鼠标移出待测窗口,发送鼠标移出消息
                    PostMessage(hWndTest, WM-MOVE-
OUT,
                    (WPARAM)hWndCur, 0);
              hWndTest = NULL;
           }
     }
   return CallNextHookEx(hHook, code, wParam, lParam);
```