关于 Hook·使用 SetWindowsHookEx 进行消息 HOOK

一、基本概念：

钩子(Hook)，是 Windows 消息处理机制的一个平台,应用程序可以在上面设置子程以监视指定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的。当消息到达后，在目标窗口处理函数之前处理它。钩子机制允许应用程序截获处理 window 消息或特定事件。

钩子实际上是一个处理消息的程序段，通过系统调用，把它挂入系统。每当特定的消息发出，在没有到达目的窗口前，钩子程序就先捕获该消息，亦即钩子函数先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。

二、运行机制：

1、钩子链表和钩子子程：

每一个 Hook 都有一个与之相关联的指针列表，称之为钩子链表，由系统来维护。这个列表的指针指向指定的，应用程序定义的，被 Hook 子程调用的回调函数，也就是该钩子的各个处理子程。当与指定的 Hook 类型关联的消息发生时，系统就把这个消息传递到 Hook 子程。一些 Hook 子程可以只监视消息，或者修改消息，或者停止消息的前进，避免这些消息传递到下一个 Hook 子程或者目的窗口。最近安装的钩子放在链的开始，而最早安装的钩子放在最后，也就是后加入的先获得控制权。

Windows 并不要求钩子子程的卸载顺序一定得和安装顺序相反。每当有一个钩子被卸载，Windows 便释放其占用的内存，并更新整个 Hook 链表。如果程序安装了钩子，但是在尚未卸载钩子之前就结束了，那么系统会自动为它做卸载钩子的操作。

钩子子程是一个应用程序定义的回调函数(CALLBACK Function),不能定义成某个类的成员函数，只能定义为普通的 C 函数。用以监视系统或某一特定类型的事件，这些事件可以是与某一特定线程关联的，也可以是系统中所有线程的事件。

钩子子程必须按照以下的语法：

```
LRESULT CALLBACK HookProc
(
    int nCode,
    WPARAM wParam,
    LPARAM lParam
);
```

HookProc 是应用程序定义的名字。

nCode 参数是 Hook 代码，Hook 子程使用这个参数来确定任务。这个参数的值依赖于 Hook 类型，每一种 Hook 都有自己的 Hook 代码特征字符集。

wParam 和 lParam 参数的值依赖于 Hook 代码，但是它们的典型值是包含了关于发送或者接收消息的信息。

2、钩子的安装与释放：

使用 API 函数 SetWindowsHookEx()把一个应用程序定义的钩子子程安装到钩子链表中。SetWindowsHookEx 函数总是在 Hook 链的开头安装 Hook 子程。当指定类型的 Hook 监视的事件发生时，系统就调用与这个 Hook 关联的 Hook 链的开头的 Hook 子程。每一个 Hook 链中的 Hook 子程都决定是否把这个事件传递到下一个 Hook 子程。Hook 子程传递事件到下一个 Hook 子程需要调用 CallNextHookEx 函数。

```
HHOOK SetWindowsHookEx(
        int idHook,      // 钩子的类型，即它处理的消息类型
        HOOKPROC lpfn,   // 钩子子程的地址指针。如果 dwThreadId 参数为 0
  // 或是一个由别的进程创建的线程的标识，
  // lpfn 必须指向 DLL 中的钩子子程。
  // 除此以外，lpfn 可以指向当前进程的一段钩子子程代码。
  // 钩子函数的入口地址，当钩子钩到任何消息后便调用这个函数。
        HINSTANCE hMod,  // 应用程序实例的句柄。标识包含 lpfn 所指的子程的 DLL。
  // 如果 dwThreadId  标识当前进程创建的一个线程，
  // 而且子程代码位于当前进程，hMod 必须为 NULL。
  // 可以很简单的设定其为本应用程序的实例句柄。可以使用 AfxGetInstanceHandle（）获取当
//前句柄
        DWORD dwThreadId // 与安装的钩子子程相关联的线程的标识符。
  // 如果为 0，钩子子程与所有的线程关联，即为全局钩子。
              );
```

函数成功则返回钩子子程的句柄，失败返回 NULL。

以上所说的钩子子程与线程相关联是指在一钩子链表中发给该线程的消息同时发送给钩子子程，且被钩子子程先处理。

在钩子子程中调用得到控制权的钩子函数在完成对消息的处理后，如果想要该消息继续传递，那么它必须调用另外一个 SDK 中的 API 函数 CallNextHookEx 来传递它，以执行钩子链表所指的下一个钩子子程。这个函数成功时返回钩子链中下一个钩子过程的返回值，返回值的类型依赖于钩子的类型。这个函数的原型如下：

```
LRESULT CallNextHookEx
  (
   HHOOK hhk;
   int nCode;
   WPARAM wParam;
   LPARAM lParam;
    );
```

hhk 为当前钩子的句柄，由 SetWindowsHookEx()函数返回。
NCode 为传给钩子过程的事件代码。
wParam 和 lParam 分别是传给钩子子程的 wParam 值，其具体含义与钩子类型有关。

钩子函数也可以通过直接返回 TRUE 来丢弃该消息，并阻止该消息的传递。否则的话，其他安

装了钩子的应用程序将不会接收到钩子的通知而且还有可能产生不正确的结果。

钩子在使用完之后需要用 UnHookWindowsHookEx()卸载，否则会造成麻烦。释放钩子比较简单，UnHookWindowsHookEx()只有一个参数。函数原型如下：

```
UnHookWindowsHookEx
 (
 HHOOK hhk;
 );
```

函数成功返回 TRUE，否则返回 FALSE。

3、实例：

```
#pragma data_seg("mydata")
HHOOK hook=NULL;              //安装的鼠标勾子句柄
#pragma data_seg()
#pragma comment(linker,"/SECTION:mydata,RWS")

extern "C" _declspec (dllexport) bool    SetHook()
{
    hook=SetWindowsHookEx(WH_SHELL,ShellProc,glhInstance,0);
    if(NULL==hook)
    {
        ::MessageBox(NULL,"SetWindowsHookEx!","Error!",MB_ICONERROR);
        return false;
    }
    ::MessageBox(NULL,"注册表实时监控开启成功！","通知",MB_OK);
    return true;
}

extern "C" _declspec (dllexport) bool    UnSetHook()
{
    bool ret=false;
    if(hook)
    {
        ret=UnhookWindowsHookEx(hook);
        if(!ret)
        {
            ::MessageBox(NULL,"UnhookWindowsHookEx!","Error!",MB_ICONERROR);
            return false;
        }

        return true;
    }
    return false;
```

```
}

LRESULT CALLBACK ShellProc(int nCode, WPARAM wParam,LPARAM lParam)
{
    if(nCode==HSHELL_WINDOWCREATED)        //  HOOK 的目的只在于映射进 DLL，这里后面的处理也可以，
    {
        PID=GetCurrentProcessId();        //  但是这里只做 API 注入，就不用了
        hProcess = OpenProcess(PROCESS_ALL_ACCESS,0, PID);
        Init();
        Inject();
    }
    return CallNextHookEx(hook,nCode,wParam,lParam);
}
```

4、系统钩子与线程钩子：

SetWindowsHookEx()函数的最后一个参数决定了此钩子是系统钩子还是线程钩子。

线程勾子用于监视指定线程的事件消息。线程勾子一般在当前线程或者当前线程派生的线程内。

系统勾子监视系统中的所有线程的事件消息。因为系统勾子会影响系统中所有的应用程序，所以勾子函数必须放在独立的动态链接库(DLL) 中。系统自动将包含"钩子回调函数"的 DLL 映射到受钩子函数影响的所有进程的地址空间中，即将这个 DLL 注入了那些进程。

几点说明：

（1）如果对于同一事件（如鼠标消息）既安装了线程勾子又安装了系统勾子，那么系统会自动先调用线程勾子，然后调用系统勾子。

（2）对同一事件消息可安装多个勾子处理过程，这些勾子处理过程形成了勾子链。当前勾子处理结束后应把勾子信息传递给下一个勾子函数。

（3）勾子特别是系统勾子会消耗消息处理时间，降低系统性能。只有在必要的时候才安装勾子，在使用完毕后要及时卸载。

三、钩子类型

每一种类型的 Hook 可以使应用程序能够监视不同类型的系统消息处理机制。下面描述所有可以利用的 Hook 类型。

1、WH_CALLWNDPROC 和 WH_CALLWNDPROCRET Hooks

WH_CALLWNDPROC 和 WH_CALLWNDPROCRET Hooks 使你可以监视发送到窗口过程的消息。系统在消息发送到接收窗口过程之前调用 WH_CALLWNDPROC Hook 子程，并且在窗口过程处理完消息之后调用 WH_CALLWNDPROCRET Hook 子程。

WH_CALLWNDPROCRET Hook 传递指针到 CWPRETSTRUCT 结构，再传递到 Hook 子程。CWPRETSTRUCT 结构包含了来自处理消息的窗口过程的返回值，同样也包括了与这个消息关联

的消息参数。

2、WH_CBT Hook

在以下事件之前，系统都会调用 WH_CBT Hook 子程，这些事件包括：

1. 激活，建立，销毁，最小化，最大化，移动，改变尺寸等窗口事件；
2. 完成系统指令；
3. 来自系统消息队列中的移动鼠标，键盘事件；
4. 设置输入焦点事件；
5. 同步系统消息队列事件。

Hook 子程的返回值确定系统是否允许或者防止这些操作中的一个。

3、WH_DEBUG Hook

在系统调用系统中与其他 Hook 关联的 Hook 子程之前，系统会调用 WH_DEBUG Hook 子程。你可以使用这个 Hook 来决定是否允许系统调用与其他 Hook 关联的 Hook 子程。

4、WH_FOREGROUNDIDLE Hook

当应用程序的前台线程处于空闲状态时，可以使用 WH_FOREGROUNDIDLE Hook 执行低优先级的任务。当应用程序的前台线程大概要变成空闲状态时，系统就会调用 WH_FOREGROUNDIDLE Hook 子程。

5、WH_GETMESSAGE Hook

应用程序使用 WH_GETMESSAGE Hook 来监视从 GetMessage or PeekMessage 函数返回的消息。你可以使用 WH_GETMESSAGE Hook 去监视鼠标和键盘输入，以及其他发送到消息队列中的消息。

6、WH_JOURNALPLAYBACK Hook

WH_JOURNALPLAYBACK Hook 使应用程序可以插入消息到系统消息队列。可以使用这个 Hook 回放通过使用 WH_JOURNALRECORD Hook 记录下来的连续的鼠标和键盘事件。只要 WH_JOURNALPLAYBACK Hook 已经安装，正常的鼠标和键盘事件就是无效的。WH_JOURNALPLAYBACK Hook 是全局 Hook，它不能象线程特定 Hook 一样使用。WH_JOURNALPLAYBACK Hook 返回超时值，这个值告诉系统在处理来自回放 Hook 当前消息之前需要等待多长时间（毫秒）。这就使 Hook 可以控制实时事件的回放。WH_JOURNALPLAYBACK 是 system-wide local hooks，它們不會被注射到任何行程位址空間。

7、WH_JOURNALRECORD Hook

WH_JOURNALRECORD Hook 用来监视和记录输入事件。典型的，可以使用这个 Hook 记录连续的鼠标和键盘事件，然后通过使用 WH_JOURNALPLAYBACK Hook 来回放。WH_JOURNALRECORD Hook 是全局 Hook，它不能象线程特定 Hook 一样使用。WH_JOURNALRECORD 是 system-wide local hooks，它們不會被注射到任何行程位址空間。

8、WH_KEYBOARD Hook

在应用程序中，WH_KEYBOARD Hook 用来监视 WM_KEYDOWN and WM_KEYUP 消息，这些消息通过 GetMessage or PeekMessage function 返回。可以使用这个 Hook 来监视输入到消息队列中的键盘消息。

9、WH_KEYBOARD_LL Hook

WH_KEYBOARD_LL Hook 监视输入到线程消息队列中的键盘消息。

10、WH_MOUSE Hook

WH_MOUSE Hook 监视从 GetMessage 或者 PeekMessage 函数返回的鼠标消息。使用这个 Hook 监视输入到消息队列中的鼠标消息。

11、WH_MOUSE_LL Hook

WH_MOUSE_LL Hook 监视输入到线程消息队列中的鼠标消息。

12、WH_MSGFILTER 和 WH_SYSMSGFILTER Hooks

WH_MSGFILTER 和 WH_SYSMSGFILTER Hooks 使我们可以监视菜单，滚动条，消息框，对话框消息并且发现用户使用 ALT+TAB or ALT+ESC 组合键切换窗口。WH_MSGFILTER Hook 只能监视传递到菜单，滚动条，消息框的消息，以及传递到通过安装了 Hook 子程的应用程序建立的对话框的消息。WH_SYSMSGFILTER Hook 监视所有应用程序消息。

WH_MSGFILTER 和 WH_SYSMSGFILTER Hooks 使我们可以在模式循环期间过滤消息，这等价于在主消息循环中过滤消息。