**全局钩子:**

**1. 怎么进入到我们的程序里来的?**

答：假如有个程序A安装了WH\_GETMESSAGE的全局钩子，钩子函数在xxx.dll中，那么当其它程序在调用GetMessage函数从自己的消息队列中取消息的时候，系统发现程序WH\_GETMESSAGE的全局钩子已经安装，就会检查调用 GetMessage的进程是否加载了xxx.dll，如果没有，就调用LoadLibrary进行加载，然后调用xxx.dll中的钩子过程。这样，钩子dll 就会在所有调用GetMessage的进程中加载

全局钩子必须为动态库的形式存在。

1. **阻止全局钩子：**

答：阻止系统调用LoadLibrary的时候返回失败。

这样做有一个问题，如何确定什么时候让LoadLibrary失败？

经过研究发现，正常的加载dll函数调用都是从 kernel32.dll中来的，而只有加载钩子过程是在user32.dll中进行的(winxp系统下，以后的不知道是否也是这样)。我们可以判断一 下LoadLibrary函数的返回地址，如果是在user32.dll的地址空间，就认为是钩子dll的加载，直接返回0就可以了。

**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序以监听指定窗口的某种消息，而且所监听的窗口可以是由其他

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。

**运行机制**

**1、钩子链表与钩子子程**

每个Hook都有一个与之相关的指针列表，称之为钩子链表，它由系统来维护，这个链表的指针指向由程序指定的回调函数，当消息来到时，就会调用这个回调函数。

这些回调函数被称为钩子的子程。

一些HOOK子程只监听消息，或者修改消息，或者停止消息前进，防止这些消息传递到下一个HOOK子程或者目标窗口。

最近安装的钩子安装在链表最前面，最早安装的放在链表的最后面，也就是说后加入链表的钩子会先获得控制权。

Windows并不要求卸载钩子的顺序与安装的顺序相反。第当一个钩子被卸载时，Windows便释放这个钩子所占的内存，并更新整个HOOK链表。

如果安装了钩子，但在卸载它之前就结束了，那么操作系统会自动将其所占内存释放。

**2、钩子安装与释放**

使用API SetWindowsHookEx将应用程序定义的钩子安装到钩子链表中。SetWIndowsHookEx总是在HOOK链表开头安装HOOK子程。当指定的HOOK消息发生时，

系统就会调用与这个HOOK相关联的HOOK链表开头的HOOK子程。每一个HOOK链中的子程都可以决定将这个消息是否传递到下一个子程，如果要将这个消息

传递到下一个子程，就要调用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。如果为NULL，则表明就钩自己进程内的消息。

// 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。

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。

);

钩子在使用完之后需要用UnHookWindowsHookEx()卸载，否则会造成麻烦

**3、一些运行机制**

在Win16环境中，DLL的全局数据对每个载入它的进程来说都是相同的；而在Win32环境中，情况却发生了变化，

DLL函数中的代码所创建的任何对象（包括变量）都归调用它的线程或进程所有。

当进程加载DLL时，它会把这个DLL的二进制映射到自己的私有进程地址空间中，也就是进程的虚拟地址空间。而且复制该DLL的全局数据的一份拷贝

到该进程地址空间，也就是说，它会重新初始化该DLL的全局数据。每个DLL拥有相同名称的全局数据，但它们的值有可能不同。而且互不相关。

因此，在Win32环境下要想在多个进程中共享数据，就必须进行必要的设置。在访问同一个Dll的各进程之间共享存储器是通过存储器映射文件技术实现的。

也可以把这些需要共享的数据分离出来，放置在一个独立的数据段里，并把该段的属性设置为共享。

必须给这些变量赋初值，否则编译器会把没有赋初始值的变量放在一个叫未被初始化的数据段中。

#pragma data\_seg预处理指令用于设置共享数据段。例如：

#pragma data\_seg("SharedDataName")

HHOOK hHook=NULL;

#pragma data\_seg()

在#pragma data\_seg("SharedDataName")和#pragma data\_seg()之间的所有变量将被访问该Dll的所有进程看到和共享。再加上一条指令#pragma comment(linker,"/section:.SharedDataName,rws"),

那么这个数据节中的数据可以在所有DLL的实例之间共享。

所有对这些数据的操作都针对同一个实例的，而不是在每个进程的地址空间中都有一份。

当进程隐式或显式调用一个动态库里的函数时，系统都要把这个动态库映射到这个进程的虚拟地址空间里(以下简称"地址空间")。

这使得DLL成为进程的一部分，以这个进程的身份执行，使用这个进程的堆栈。

**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。

如果要实现系统钩子（全局钩子），那么就要把安装钩子，卸载钩子，钩子处理函数都要放到DLL中，那么就要保证这个DLL尽可能的小，因为它会

映射到别的进程地址空间中去。

那么如果才能通知到我们自己写的应用程序呢？不要指望设置回调函数了。就算你在钩子DLL中保存一个回调函数地址，也是不行的，因为当这人DLL

映射到别的进程地址空间中时，它的所有全局变量都会被重新初始化，就算你用内存共享将这个变量共享了，那个回调的地址很可能是被映射进程地址空间中的，也是不行的。

一个很好的办法是保存一个我们程序的线程ID，把这个ID共享了，当收到HOOK消息时，调用PostThreadMessage把这些消息发送到我们程序所在的线程。

这样就能实现监听消息了。

**5、钩子的不足**

1）由于全局钩子会把一段DLL的二进制代码映射到别的EXE中，这可能会很危险。

2）32位的DLL不能映射到64位EXE中，64位DLL也不能映射到32位的EXE中，所以就会导致会写两个版全的监听程序32位和64位。32位DLL与64位DLL必须有不同的名字。

3）钩子会消耗消息处理时间，会导致性能差。

**6、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。