Windows操作系统中屏幕取词技术的研究

童 强
(湖北师范学院 计算机科学系,湖北 黄石 435002)

摘要:分析 Windows操作系统中屏幕取词技术的原理和实现方法,并对其中涉及到的钩子函数的使用,API拦截以及进程间通讯等关键技术和步骤进行了深入研究。

关键词:钩子;动态连接库;API拦截;屏幕取词
中图分类号: TP316. 7; TP311. 52 **文献标识码**: A **文章编号**: 1009-2714(2006)02-0075-04

0 引言

虽然屏幕取词技术被很多翻译软件及字典软件应用,但是由于其核心涉及到 Windows操作系统的很多底层技术,包括 hook技术,API拦截技术、动态库管理以及进程间的通讯等,实现起来却比较复杂。实现屏幕取词所涉及的这些底层技术也可以运用到其它软件开发中,还有助于剖析和理解 Windows操作系统。

1 实现屏幕取词的原理

要实现屏幕取词首先要了解在 Windows操作系统的文本是如何输出的,实际上 Windows系统下屏幕上不管具体是哪个应用程序的文本输出,最后都是调用 Windows系统的动态库 GDI32. dll中的 TextOutA、TextOutW、ExtTextOutA、ExtTextOutW 等几个 API函数实现的。即不论哪个程序,只要输出文本都会在程序模块中导入动态库 GDI32. dll,并最终利用上述几个函数输出文本。其中 TextOutA 和 ExtTextOutA 用于输出 ANSI字符, TextOutW 和 ExtTextOutW 用于输出 unicode字符。所以实现鼠标屏幕取词的关键就是如何设法在 Windows字符输出的时候取得这几个 API函数的参数。可以通过采用 APIHOOK技术实现对这几个 API函数的挂接,在输出字符的时候截获 API函数的参数,这些参数包括输出的文本的字符串以及及其输出的位置坐标信息。

根据 Windows系统的工作原理,当窗体全部或部分需要刷新时, TextOutA等 API就会被调用重新输出[1],比如一个窗体盖住了另一个窗体,当上面窗体移开后下面的窗体会收到系统的 WM_PAINT 消息而刷新, TextOutA等被调用绘制文字部分,看起来就像真是露出了原来被挡住的部分。因此必须设法在需要的时候让有关窗体的有关区域刷新,具体做法就是在鼠标所在位置画一个小窗体挡住下面的窗体一下(肉眼看不出来),下面的窗体被挡住的字符就会重绘, (另外一种方法是调用 Invalidaerect和 Updada window函数强制区域刷新)那几个已经被挂接的 API就会被调用了,于是在挂接的钩子函数中可以获得其参数,通过计算鼠标位置和输出文本位置就得到了鼠标下的文字了。

主要步骤如下:

1)挂上 API钩子;

收稿日期: 2005—10—12
作者简介:童 强(1968—),男,湖北黄石人,讲师,工程师,硕士,主要研究方向为多媒体应用技术、Web应用技术等。

2)得到鼠标当前位置,并设法让鼠标下的窗口刷新;

3)在 API钩子函数中截获字符串,并根据鼠标位置计算出鼠标下的字符;并调用原型函数完成刷新工作。

为了随时知道鼠标的位置,还要安装一个鼠标钩子以便随时得到鼠标的屏幕坐标。这里用到了两种钩子,即鼠标钩子和前面提到的 API钩子。

2 挂接 API钩子截获 API参数

如前所述,要获得系统 API的输出参数,关键就是对几个有关文本输出的 API函数进行挂接。实现 API挂接的方法有多种,这里采用的方法是通过操作模块的输入节来挂接 API。[2]

模块的输入节包含一组该模块运行的时候需要的 DLL以及该模块从每个 DLL输入的符号的列表。当模块调用一个输入函数时,线程实际上要从模块的输入节中捕获输入函数的地址,然后转移到该地址。

要挂接前面讲述的 GDI32. dll中的 TextOutA、TextOutW、ExtTextOutA、ExtTextOutW 等几个 API函数,只需要改变模块的输入节中这几个函数的地址,改成预先编写的替代函数地址就可以了。下面是对某个模块输入节进行改写的基本步骤:

1)首先对该模块 (例如运行的 word程序)调用 ImageDirectoryEntryToData函数看该模块该模块的输入节,如果返回 NULL说明该模块没有输入节,则不做如何动作结束挂接工作。

2)如果有输入节, ImageDirectoryEntryToData返回该输入节地址,一个类型为 IMAGE_DIRECTORY_ENTRY_IMPORT的指针,实际上是一个包含输入节信息的结构体。扫描这个结构体中所有 dll的名字,看有没有要找的 GDI32. dll(看这个模块是否用到 GDI32. dll),如果扫描结束没有找到,说明没有用到 GDI32. dll,则结束挂接工作。

3)如果 2)中找到 GDI32. dll就可以得到一个包含输入符号信息的 IMAGE_THUNK_DATA结构的数组地址。同上步类似扫描来自 GDI32. dll的所有输入符号,寻找要用的函数 (如 TextOutA)地址相匹配的地址。这就是要修改的地址。

4)用 WriteProcessMemory()函数将输入节中指向 API原型函数的地址改成替代函数的地址。

这里编写的替代函数必须和原函数格式、参数完全一致,以 TextOutA()为例,定义替代函数 NewTextOutA形式如下:

```
BOOL WINAPI NewTextOutA (HDC hdc, int nXStart, int nYStart, LPCTSTR lpString, int cbString)
{
    //首先判断是否是取词程序强制刷新引起的 TextOutA调用
if (bTraceMouse == TRUE)
{ //调用处理函数对截获的字符串及其位置进行处理;
GetText (hdc, nXStart, nYStart lpszString, cbString, &tSize);
}
//最后后调用原型的 TextOutA函数输出;
int nResult = ((PFNTextOutA) (PROC) g_TextOutA) (hdc, nXStart, nYStart, lpszString, cbString);
return (nResult);
};
```

替代函数参数和原型函数 TextOutA完全一样,在替代函数中可以根据这些参数可以计算出截获的字符串的的屏幕坐标,和鼠标钩子中得到的鼠标位置进行比较就可以鼠标所在位置所指的字符串。替代函数中还必须做一件事:就是利用原始函数的地址调用一次原始函数,使字符串在屏幕上输出,否则屏幕上的字符显示不全或消失 (被强制刷新的小窗体遮挡过)。

上述挂接完成后,被挂接的模块中的任何线程要调用 TextOutA等函数的时候就会先调用替代函数,由替代函数调用原型的 TextOutA等,完全不影响被挂接模块的正常工作。

在取词应用程序终止前要撤销挂接,撤销挂接的时候也同样调用 WriteProcessMemory()函数修复被挂接模块的输入节,只是正好相反在输入节中用原型 API函数的地址取代替代函数地址。

API挂接还有一个问题,比如在第一次挂接之前,通过遍历所有进程,对每一个进程都进行相应模块的挂接,但是如果以后还有其它进程产生,比如又打开了新的程序,那么对新进程的模块就没有挂接操作,拦截也就失效了。当然也有办法解决这个问题,就是还要加一段监视系统的代码,当有新的进程产生,就对新进程进行 API挂接操作,这种方法比较麻烦,需要监视系统是否有新进程产生。这里采用一个比较简单有效的方法:在鼠标钩子函数中加一段代码,检测鼠标下的窗体是否已经进行了挂接,如果没有,就在鼠标钩子函数中进行挂接动作,这样就可以保障所有的窗体进程都会有效进行 API挂接,从而保证了拦截的有效。

3 安装鼠标钩子

Windows系统是建立在事件驱动的机制上的,整个系统都是通过消息的传递机制来实现的。而钩子是 Windows系统中非常重要的系统接口,用它可以截获并处理送给其它应用程序的消息,来完成普通应用程序难以实现的功能。钩子的种类很多,每种钩子可以截获并处理相应的消息,键盘钩子可以截获键盘消息,外壳钩子可以截取启动和关闭应用程序的消息等,鼠标钩子则可以截获鼠标坐标等信息。

要实现 Win32的系统钩子,必须调用 API函数 SetWindowsHookEx来安装这个钩子函数,这个函数的原型是:

```
HHOOK SetWindowsHookEx(
int idHook,  //钩子的类型
HOOKPROC lpfn,  //钩子函数的地址
HINSTANCE hMod,  //包含钩子函数的模块句柄
DWORD dwThreadId  //指定监视的线程,全局钩子用 NULL即可
)
```

安装鼠标钩子时则第一个参数为 WH_MOUSE;第三个参数可以用 API函数 GetModuleHandle来获得钩子所在模块的句柄;第四个参数如果指定确定的线程,即为线程专用钩子;如果指定为空,即为全局钩子。在这里使用的的鼠标钩子当然是全局钩子。全局钩子函数必须包含在 DLL(动态链接库)中,

上述第二个参数指向钩子回调函数,消息处理就在得到控制权的钩子函数中完成。下面是鼠标回调函数的格式:

```
LRESULT CALLBACK MouseProc( int nCode, WPARAM wParam, LPARAM lParam)
{//利用 lParam参数的到鼠标的屏幕坐标位置,判断是否移动,控制发出刷新等
……
if (b_mousemove = = true)
drawsmallwin(); //调用刷新程序控制发出刷新
return CallNextHookEx(hMouseHook, nCode, wParam, lParam);
}
```

在鼠标钩子函数中可以随时获得鼠标的屏幕坐标位置从而判断鼠标是否移动,移动到新位置的时候可以调用 SetWindowPos()函数和 ShowWindow()函数将一个小窗体在鼠标所在位置闪一下(相当于遮住了下面的窗体一下),以便鼠标下的字符重绘。(也可以调用 InvalidateRect(), InvalidateRgn()函数实现重绘的目的)另外一般要该消息继续传递,那么在钩子函数的结尾它必须调用另外一个 API函数 CallNextHookEx来传递它。

上述 SetWindowsHookEx函数的调用使钩子程序接入到了系统的钩子链中。在应用程序结束的时候,必须用到 UnhookWindowsHookEx()函数将挂接的钩子程序从系统的钩子链中撤销。

4 在进程间传递信息

由于钩子函数必须存在于在动态库 dll中,而当全局钩子成功挂接时,包含钩子的动态库实际上会映像到它所挂接的进程空间,那么 在钩子函数中所截获的有关信息,要传回主程序,就有一个进程间传递信息的问题。这里 采用内存映像文件在进程间共享资料的方法实现进程间的资料传递。和其它的共享资料的机制相比,这种方法可以达到较小的开销和较高的效能。其建立主要步骤如下:

1)在主程序的初始化的时候用 CreateFileMapping()函数建立一个内存文件映像对象,再调用 MapViewOfFile()函数获得这个共享文档的指针,然后可以对共享数据结构进行初始化,最后用 UnmapViewOfFile()函数和 CloseHandle()函数释放和关闭对共享区的控制。

2)在钩子函数所在的动态库中也可以对上面创建的共享数据结构进行读写操作,只要把要传递的资料写到这个共享的内存映像文件中,然后发送一个自定义消息给主程序,通知它到内存文件映像文件去读取取词消息。

3)不论在主程序中还是钩子函数所在的 DLL 中,每次对内存映像文件读写之前先调用 OpenFileMapping()和 MapViewOfFile()函数获得对内存映像文件的读写控制,读写完成之后 UnmapViewOfFile()和 CloseHandle()函数释放对他的控制。

5 结束语

因为 Windows各种版本的内核不同,许多在 Windows NT/2000/xp 下的特性在 Windows 95/98/me下并部完全支持。上面讨论的屏幕取词方法适用于 32位的 Windows NT/2000/xp 系列。而 Windows 95/98/me系统继承了 16位 Windows操作系统的特性[2],是 16位混合编程的 32位操作系统,本方法在这些系统下不是全部起作用,要完全实现 Windows 95/98/me下的屏幕取词涉及到 16位 Windows的改写代码挂接 API方法[3],本文不另作讨论。

屏幕取词功能的实现是对 API拦截等涉及系统内核相关技术的一种综合应用,除了可以直接用在电子字典中,还运用于中文平台的开发上。这种技术结合 TTS语音技术还可以开发帮助盲人和弱视人使用计算机的语音应用软件。这些涉及操作系统底层的技术对于了解 Windows系统内部工作机制,开发功能强大的软件都有很大的意义。

参考文献:

[1] Charles Petzold. Windows程序设计 [M]. 北京博彦科技发展有限公司 译. 北京:北京大学出版社, 1999.

[2] Jeffrey Richter. Windows核心编程 [M]. 王建华 译. 北京:机械工业出版社, 2000.

[3] 郭 飞,周曼丽. Windows API拦截技术综述 [J]. 计算机工程与应用, 2002, 19: 144～146.

Research on the technique about pick words from screen in the Windows operating system

TONG Qiang

(Department of Computer Science, Hubei Normal University, Huangshi 435002, China)

Abstract: Analyzed the principle and the technique about pick words from screen in the Windows operating system, and the key technologies and steps such as use HOOK function, Intercept Windows API and communication between different process were discussed.

Key words: HOOK; DLL; intercept windows API; pick words from screen