

<	2016年10月						>
日	一	二	三	四	五	六	
25	26	27	28	29	30	1	
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31	1	2	3	4	5	

昵称: 北极星 - North Star
园龄: 3年10个月
粉丝: 4
关注: 0
[+加关注](#)

搜索

找找看

谷歌搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)
[更多链接](#)

我的标签

[BrowseForFolder\(1\)](#)
[BusinessSkinForm\(1\)](#)
[Delphi\(1\)](#)
[目录对话框\(1\)](#)

键盘鼠标模拟全知道

本文目录如下

一、基于 **windows** 消息机制的鼠标键盘模拟

（一）、应用程序级模拟

（二）、系统级模拟

1、用API函数 **keybd_event** 模拟键盘事件

2、 **SendInput**函数模拟全局键盘鼠标事件

3、用全局钩子模拟键盘消息

二、驱动级模拟

一、基于 **windows** 消息机制的鼠标键盘模拟

我们怎样才能用Delphi来写一个程序，用来代替人们按键的方法呢？那就让我们来先了解一下windows中响应键盘事件的机制。

当用户按下键盘上的一个键时，键盘内的芯片会检测到这个动作，并把这个信号传送到计算机。如何区别是哪一个键被按下了呢？键盘上的所有按键都有一个编码，称作键盘扫描码。当你按下一个键时，这个键的扫描码就被传给系统。扫描码是跟具体的硬件相关的，同一个键，在不同键盘上的扫描码有可能不同。键盘控制器就是将这个扫描码传给计算机，然后交给键盘驱动程序。键盘驱动程序会完成相关的工作，并把这个扫描码转换为键盘虚拟码。什么是虚拟码呢？因为扫描码与硬件相关，不具有通用性，为了统一键盘上所有键的编码，于是就提出了虚拟码概念。无论什么键盘，同一个按键的虚拟码总是相同的，这样程序就可以识别了。简单点说，虚拟码就是我们经常可以看到的像VK_A,VK_B这样的常数，比如键A的虚拟码是65，写成16进制就是&H41，注意，人们经常用16进制来表示虚拟码。当键盘驱动程序把扫描码转换为虚拟码后，会把这个键盘操作的扫描码和虚拟码还有其它信息一起传递给操作系统。然后操作系统则会把这些信息封装在一个消息中，并把这个键盘消息插入到消息列队。最后，要是不出意外的话，这个键盘消息最终会被送到当前的活动窗口那里，活动窗口所在的应用程序接收到这个消息后，就知道键盘上哪个键被按下，也就可以决定该作出什么响应给用户了。

随笔分类(32)

[delphi 基础语法、算法\(11\)](#)
[Delphi-数据库\(3\)](#)
[IDE/VCL组件\(4\)](#)
[多线程\(2\)](#)
[监视后台\(1\)](#)
[键盘鼠标\(1\)](#)
[目录文件操作\(2\)](#)
[其它\(1\)](#)
[资源文件\(1\)](#)
[字符和字符串、字符列表\(6\)](#)

随笔档案(35)

[2015年7月 \(22\)](#)
[2015年6月 \(8\)](#)
[2014年9月 \(1\)](#)
[2014年4月 \(1\)](#)
[2012年12月 \(3\)](#)

文章分类(18)

[Delphi-结构和算法\(2\)](#)
[Delphi-界面 皮肤\(2\)](#)
[Delphi-数据库\(11\)](#)
[Delphi-周边\(2\)](#)
[Windows-API\(1\)](#)
[进程、线程](#)
[系统](#)

arvid 的博客

[天心网delphi](#)
[arvid 的博客](#)
[delphi](#)
[devlyn的博客](#)
[瓢虫Monster](#)
[万一的博客](#)

delphi

阅读排行榜

[1. 键盘鼠标模拟全知道\(9115\)](#)
[2. delphi 基础之三 文件流操作\(303\)](#)
[3. 使用 VCL BDE 组件动态创](#)

这个过程可以简单的如下表示：

用户按下键盘上的一个键 >>>>> 键盘控制器就把这个键的扫描码传给计算机，然后交给键盘驱动程序 >>>>> 键盘驱动程序会把这个扫描码转换为键盘虚拟码(**VK_A,VK_B**这样的常数，比如键**A**的虚拟码是**65**，写成**16**进制就是**&H41**)传给操作系统 >>>>> 操作系统则会把这些信息封装在一个消息中，并把这个键盘消息插入到消息队列 >>>>> 键盘消息被发送到当前活动窗口

明白了这个过程，我们就可以编程实现在其中的某个环节来模拟键盘操作了。在Delphi中，有多种方法可以实现键盘模拟，我们就介绍几种比较典型的。

（一）、应用程序级模拟（只针对某个程序，我称之为局部模拟）

windows提供了几个这样的API函数可以实现直接向目标程序发送消息的功能，常用的有SendMessage和PostMessage，它们的区别是PostMessage函数直接把消息仍给目标程序就不管了，而SendMessage把消息发出去后，还要等待目标程序返回些什么东西才好。这里要注意的是，模拟键盘消息一定要用PostMessage函数才好，用SendMessage是不正确的(因为模拟键盘消息是不需要返回值的，不然目标程序会没反应)，切记切记！

PostMessage函数的delphi声明如下：

```
PostMessage(  
  
hWnd: HWND;           {目标程序上某个控件的句柄}  
  
Msg: UINT;             {消息的类型}  
  
wParam: WPARAM;        {32位指定附加的消息特定的信息}  
  
lParam: LPARAM          {32位指定附加的消息特定的信息}  
  
): BOOL;
```

参数hwnd 是你要发送消息的目标程序上某个控件的句柄，参数Msg 是消息的类型，表示你要发送什么样的消息，最后wParam 和 lParam这两个参数是随消息附加的数据，具体内容要由消息决定。

建数据库表(279)

4. Delphi编写后台监控软件(238)

5. 关于文件、目录操作的函数(234)

推荐排行榜

1. 键盘鼠标模拟全知道(1)

再来看看Msg 这个参数，要模拟按键就靠这个了。

键盘消息常用的有如下几个：

WM_KEYDOWN 表示一个普通键被按下

WM_KEYUP 表示一个普通键被释放

WM_SYSKEYDOWN 表示一个系统键被按下，比如Alt键

WM_SYSKEYUP 表示一个系统键被释放，比如Alt键

如果你确定要发送以上几个键盘消息，那么再来看看如何确定键盘消息中的wParam 和lParam 这两个参数。在一个键盘消息中，wParam 参数的含义较简单，它表示你要发送的键盘事件的按键虚拟码，比如你要对目标程序模拟按下A键，那么wParam 参数的值就设为VK_A ,至于lParam 这个参数就比较复杂了，因为它包含了多个信息，一般可以把它设为0。即

```
PostMessage (MyHwnd, WM_KEYDOWN, key, 0);
```

但是如果你想要你的模拟更真实一些，那么建议你还是设置一下这个参数。那么我们就详细了解一下lParam 吧。

lParam 是一个32 bit的参数，它在内存中占4个字节，写成二进制就是

```
00000000 00000000 00000000 00000000
```

一共是32位，我们从右向左数，假设最右边那位为第0位(注意是从0而不是从1开始计数)，最左边的就是第31位。那么该参数的

0-15位表示键的发送次数等扩展信息，

16-23位为按键的扫描码，

24-31位表示是按下键还是释放键。

大家一般习惯写成16进制的，那么就应该是

&H00 00 00 00，

第0-15位一般为&H0001，如果是按下键，那么24-31位为&H00，释放键则为&HC0，

那么16-23位的扫描码怎么会得呢？这需要用到一个API函数MapVirtualKey，这个函数可以将虚拟码转换为扫描码，或将扫描码转换为虚拟码，还可以把虚拟码转换为对应字符的ASCII码。它的delphi 声明如下：

```
MapVirtualKey(  
    uCode: UINT;                {key code, scan code or virtual key}  
    uMapType: UINT              {flags for translation mode}  
): UINT;                       {returns translated key code}
```

参数uCode 表示待转换的码，参数uMapType 表示从什么转换为什么，如果是虚拟码转扫描码，则uMapType 设置为0，如果是虚拟扫描码转虚拟码，则uMapType 设置为1，如果是虚拟码转ASCII码，则uMapType 设置为2.相信有了这些，我们就可以构造键盘事件的IPParam参数了。

下面给出一个构造IPParam参数的函数：

```
function VKB_param(VirtualKey:Integer;flag:Integer):Integer; //函数名  
  
var  
  
s,Firstbyte,Secondbyte:String;  
  
S_code:Integer;
```

```

Begin
if flag=1 then //按下键
    begin
        Firstbyte := '00'
    end
else //弹起键
    begin
        Firstbyte := 'C0'
    end;
S_code:= MapVirtualKey(VirtualKey, 0);
Secondbyte:='00'+inttostr(s_code);
Secondbyte:=copy(Secondbyte,Length(Secondbyte)-1,2);
s:='$'+Firstbyte + Secondbyte + '0001';
Result:=strtoint(s);
End;

```

使用按键的方法:

说明: **key**为键值, 如1键[不是数字键的1]的值是\$31, **flag**传递的是按键状态, 1是按下, 0是弹起。

```

lparam := VKB_param(key, 1);    {按下键}

PostMessage (MyHwnd, WM_KEYDOWN, key, lparam);

lparam := VKB_param(key, 0);    {松开键}

```

```
PostMessage (MyHwnd, WM_KEYUP, key, lParam);
```

```
var
```

```
    hwnd, lParam:Cardinal;
```

```
begin
```

```
    hwnd:=FindWindowEx(FindWindow(nil,'无标题 - 记事本'),0,'edit',nil);
```

```
lParam := VKB_param(97, 1);    {按下键}
```

```
PostMessage (hwnd,WM_KEYDOWN,vk_F3,lParam) ; //按下F3键
```

```
//PostMessage (hwnd,WM_CHAR,97,lParam);    //输入字符A (edit控件接收字符)
```

```
lParam := VKB_param(key, 0);    {松开键}
```

```
PostMessage (hwnd,WM_KEYUP,vk_F3,lParam); //释放F3键
```

```
end;
```

模拟鼠标点击

Var

P1: Tpoint;

Lparam:integer;

begin

GetCursorPos(P1); // 获取屏幕坐标

P1.X:= P1.X+100;

P1.Y:=P1.Y+100;

lparam:=p1.X+ p1.Y shl 16;

```
SendMessage(h,messages.WM_LBUTTONDOWN ,0,lparam);// 按下鼠标左键
```

```
SendMessage(h,messages.WM_LBUTTONUP ,0, lparam); //抬起鼠标左键
```

```
End;
```

（二）、系统级模拟（针对所有程序的窗口，我称之为全局模拟）

模拟全局键盘消息常见的可以有以下一些方法：

1、 用API函数keybd_event，这个函数可以用来模拟一个键盘事件

```
keybd_event(  
    bVk: Byte;           {virtual-key code}  
    bScan: Byte;         {scan-code}  
    dwFlags: DWORD;      {option flags}  
    dwExtraInfo: DWORD   {additional information about the key}  
);                       {this procedure does not return a value}
```

```
keybd_event(VK_A, 0, 0, 0) '按下A键
```

```
keybd_event (VK_A, 0, KEYEVENTF_KEYUP, 0) '释放A键
```

注意有时候按键的速度不要太快，否则会出问题，可以用API函数Sleep来进行延时，delphi声明如下：

```
Sleep(  

```

```
dwMilliseconds: DWORD           {specifies the number of milliseconds to pause}  
  
);
```

参数dwMilliseconds表示延时的时间，以毫秒为单位。

那么如果要模拟按下功能键怎么做呢？比如要按下Ctrl+C实现拷贝这个功能，可以这样：

```
keybd_event (VK_Ctrl, 0, 0, 0 ); //按下Ctrl键  
  
keybd_event (VK_C, 0, 0, 0);     //按下C键  
  
Sleep(500 );                     //延时500毫秒  
  
keybd_event (VK_C, 0, KEYEVENTF_KEYUP, 0 ); //释放C键  
  
keybd_event (VK_Ctrl, 0, KEYEVENTF_KEYUP, 0 ); //释放Ctrl键
```

好了，现在你可以试试是不是可以骗过目标程序了，这个函数对大部分的窗口程序都有效，可是仍然有一部分游戏对它产生的键盘事件熟视无睹，这时候，你就要用上bScan这个参数了。一般的，bScan都传0，但是如果目标程序是一些DirectX游戏，那么你就需要正确使用这个参数传入扫描码，用了它可以产生正确的硬件事件消息，以被游戏识别。这样的话，就可以写成这样：

```
keybd_event (VK_A, MapVirtualKey(VK_A, 0), 0, 0); //按下A键  
  
keybd_event (VK_A, MapVirtualKey(VK_A, 0), KEYEVENTF_KEYUP, 0 ); //释放A键
```

以上就是用keybd_event函数来模拟键盘事件。

模拟按下鼠标左键

```
mouse_event(MOUSEEVENTF_LEFTDOWN,0,0,0,0);//鼠标左键按下  
mouse_event(MOUSEEVENTF_LEFTUP,0,0,0,0);//鼠标左键弹起
```

2、 SendInput函数也可以模拟全局键盘鼠标事件。

SendInput可以直接把一条消息插入到消息队列中，算是比较底层的了。

SendInput的参数其实很简单，在Windows.pas就有函数的声明如下：

```
function SendInput (  
    cInputs: UINT;           pInputs中记录数组的元素数目  
    var pInputs: TInput;     TInput类型记录数组的第1个元素  
    cbSize: Integer          定义TInput的大小，一般为SizeOf(TInput)  
): UINT; stdcall;
```

cInputs: 定义pInputs中记录数组的元素数目。**pInputs:** TInput类型记录数组的第1个元素。每个元素代表插入到系统消息队列的键盘或鼠标事件。**cbSize:** 定义TInput的大小，一般为SizeOf(TInput)。函数返回成功插入系统消息队列中事件的数目，失败返回0。调用SendInput关键的就是要搞清楚它的几个记录结构的意思，在Windows.pas中对TInput的声明如下：

```
tagINPUT = packed record  
  
    Itype: DWORD;  
  
    case Integer of  
        0: (mi: TMouseInput);  
        1: (ki: TKeybdInput);  
        2: (hi: THardwareInput);  
    end;  
  
    TInput = tagINPUT;
```

其中mi、ki、hi是3个共用型的记录结构，**Itype**指出记录结构中所使用的类型，它有3个值。**INPUT_MOUSE:** 表示使用mi记录结构，忽略ki和hi；**INPUT_KEYBOARD:** 表示使用ki记录结构，忽略mi和hi。

(1)、键盘模拟

TKeybdInput记录结构的声明如下:

```
tagKEYBDINPUT = packed record
```

```
    wVk: WORD;           是要操作的按键的虚键码
```

```
    wScan: WORD;        是安全码, 一般不用
```

```
    dwFlags: DWORD;     指定键盘所进行的操作, 为0时表示按下某键, KEYEVENTF_KEYUP表示放开某键
```

```
    time: DWORD;
```

```
    dwExtraInfo: DWORD; 是扩展信息, 可以使用API函数GetMessageExtraInfo的返回值
```

```
end;
```

```
TKeybdInput = tagKEYBDINPUT;
```

其中wVk是要操作的按键的虚键码。wScan是安全码, 一般不用。dwFlags指定键盘所进行的操作, 为0时表示按下某键, KEYEVENTF_KEYUP表示放开某键。time是时间戳, 可以使用API函数GetTickCount的返回值。dwExtraInfo是扩展信息, 可以使用API函数GetMessageExtraInfo的返回值。例如击键“A”的程序如下:

```
procedure KeyPressA;
```

```
var
```

```
    Inputs : array [0..1] of TInput;
```

```
begin
```

```
    Inputs[0].Itype:=INPUT_KEYBOARD;
```

```
with Inputs[0].ki do
begin
    wVk:=VK_A;

    wScan:=0;

    dwFlags:=0;

    time:=GetTickCount;

    dwExtraInfo:=GetMessageExtraInfo;

end;

Inputs[1].Itype:=INPUT_KEYBOARD;

with Inputs[1].ki do
begin
    wVk:=VK_A;

    wScan:=0;

    dwFlags:=KEYEVENTF_KEYUP;

    time:=GetTickCount;

    dwExtraInfo:=GetMessageExtraInfo;

end;

SendInput(2,Inputs[0],SizeOf(TInput));

end;
```

注意：在Windows.pas单元中并没有字母和数字的虚键码的声明，在我写的SIMouseKeyboard.pas单元文件中对所有的虚键码进行了重新声明，包含了字母、数字和标点符号。

(2)、鼠标模拟

TMouseInput记录结构的声明如下:

```
tagMOUSEINPUT = packed record  
  
    dx: Longint;  
  
    dy: Longint;  
  
    mouseData: DWORD;  
  
    dwFlags: DWORD;  
  
    time: DWORD;  
  
    dwExtraInfo: DWORD;  
  
end;  
  
TMouseInput = tagMOUSEINPUT;
```

其中dx、dy是鼠标移动时的坐标差（不是像素单位），在鼠标移动时有效。mouseData是鼠标滚轮滚动值，在滚动鼠标滚轮时有效。当mouseData小于0时向下滚动，当mouseData大于0时向上滚动，mouseData的绝对值一般设为120。dwFlags指定鼠标所进行的操作，例如，MOUSEEVENTF_MOVE表示移动鼠标，MOUSEEVENTF_LEFTDOWN表示按下鼠标左键，MOUSEEVENTF_LEFTUP表示放开鼠标左键。time是时间戳，可以使用API函数GetTickCount的返回值。dwExtraInfo是扩展信息，可以使用API函数GetMessageExtraInfo的返回值。例如单击鼠标左键的程序如下：

```
procedure MouseClick;  
  
var
```

```
Inputs : array [0..1] of TInput;
begin
  Inputs[0].Itype:=INPUT_MOUSE;
  with Inputs[0].mi do
    begin
      dx:=0;
      dy:=0;
      mouseData:=0;
      dwFlags:=MOUSEEVENTF_LEFTDOWN;
      time:=GetTickCount;
      dwExtraInfo:=GetMessageExtraInfo;
    end;
  Inputs[1].Itype:=INPUT_MOUSE;
  with Inputs[1].mi do
    begin
      dx:=0;
      dy:=0;
      mouseData:=0;
      dwFlags:=MOUSEEVENTF_LEFTUP;
      time:=GetTickCount;
      dwExtraInfo:=GetMessageExtraInfo;
    end;
end;
```

```
SendInput(2,Inputs[0],SizeOf(TInput));  
  
end;
```

鼠标的移动总是很麻烦，上面的dx、dy不是以像素为单位的，而是以鼠标设备移动量为单位的，它们之间的比值受鼠标移动速度设置的影响。具体的解决方法我已经在《Delphi下利用WinIo模拟鼠标键盘详解》一文中进行了讨论，这里不再重复。dwFlags可以设置一个MOUSEEVENTF_ABSOLUTE标志，这使得可以用另外一种方法移动鼠标。当dwFlags设置了MOUSEEVENTF_ABSOLUTE标志，dx、dy为屏幕坐标值，表示将鼠标移动到dx、dy的位置。但是这个坐标值也不是以像素为单位的。这个值的范围是0到65535（\$FFFF），当dx等于0、dy等于0时表示屏幕的最左上角，当dx等于65535、dy等于65535时表示屏幕的最右下角，相当于将屏幕的宽和高分别65536等分。API函数GetSystemMetrics(SM_CXSCREEN)可以返回屏幕的宽度，函数GetSystemMetrics(SM_CYSCREEN)可以返回屏幕的高度，利用屏幕的宽度和高度就可以将像素坐标换算成相应的dx、dy。注意：这种换算最多会出现1像素的误差。例如：将鼠标指针移动到屏幕150,120坐标处的程序如下：

```
procedure MouseMove;  
  
var  
    Input : TInput;  
  
begin  
    Input.Itype:=INPUT_MOUSE;  
  
    with Input.mi do  
    begin  
        dx:=(FFFF div (GetSystemMetrics(SM_CXSCREEN)-1)) * 150;  
        dy:=(FFFF div (GetSystemMetrics(SM_CYSCREEN)-1)) * 120;  
  
        mouseData:=0;  
  
        dwFlags:=MOUSEEVENTF_MOVE or MOUSEEVENTF_ABSOLUTE;  
  
        time:=GetTickCount;
```

```

        dwExtraInfo:=GetMessageExtraInfo;
    end;

    SendInput(1,Input,SizeOf(TInput));
end;

```

(3)、SendInput与WinIo的对比

在《Delphi下利用WinIo模拟鼠标键盘详解》一文中我已经说了WinIo的很多缺点，SendInput几乎没有这些缺点。SendInput的模拟要比WinIo简单的多。事件是被直接插入到系统消息队列的，所以它的速度比WinIo要快。系统也会保证数据的完整性，不会出现数据包混乱的情况。利用“绝对移动”可以将鼠标指针移动到准确的位置，同鼠标的配置隔离不会出现兼容性的问题。SendInput的缺点也是最要命的，它会被一些程序屏蔽。所以说在SendInput与WinIo都可以使用的情况下优先考虑SendInput。另外SendInput与WinIo可以接合使用，一些程序对鼠标左键单击敏感，可以使用WinIo模拟鼠标左键单击，其它操作由SendInput模拟。

3、用全局钩子也可以模拟键盘消息。如果你对windows中消息钩子的用法已经有所了解，那么你可以通过设置一个全局HOOK来模拟键盘消息，比如，你可以用WH_JOURNALPLAYBACK这个钩子来模拟按键。WH_JOURNALPLAYBACK是一个系统级的全局钩子，它和WH_JOURNALRECORD的功能是相对的，常用它们来记录并回放键盘鼠标操作。WH_JOURNALRECORD钩子用来将键盘鼠标的操作忠实地记录下来，记录下来的信息可以保存到文件中，而WH_JOURNALPLAYBACK则可以重现这些操作。当然亦可以单独使用WH_JOURNALPLAYBACK来模拟键盘操作。

SetWindowsHookEx函数，它可以用来安装消息钩子：

SetWindowsHookEx()

```
idHook: Integer;           {hook type flag}
```

lpfn: TFNHookProc; {a pointer to the hook function}

```
hmod: HINST;           {a handle to the module containing the hook function}
```

dwThreadId: DWORD {the identifier of the associated thread}

): HHOOK;

先安装WH_JOURNALPLAYBACK这个钩子，然后你需要自己写一个钩子函数，在系统调用它时，把你要模拟的事件传递给钩子参数IParam所指向的EVENTMSG区域，就可以达到模拟按键的效果。不过用这个钩子模拟键盘事件有一个副作用，就是它会锁定真实的鼠标键盘，不过如果你就是想在模拟的时候不会受真实键盘操作的干扰，那么用用它倒是个不错的主意。

在不需要监视系统消息时需要调用提供UnHookWindowsHookEx来解除对消息的监视。 下面来建立程序，在Delphi中建立一个工程，在Form1上添加3个按钮用于程序操作。另外再添加一个按钮控件和一个Edit控件用于验证操作。

下面是Form1的全部代码

```
unit Unit1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
```

```
StdCtrls;
```

```
type
```

```
TForm1 = class(TForm)
```

```
Button1: TButton;
```

```
Button2: TButton;
```

```
Button3: TButton;
```

```
Edit1: TEdit;
```



```
Button4: TButton;

procedure FormCreate(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

private

    { Private declarations }

public

    { Public declarations }

end;

var

    Form1: TForm1;

    EventArr:array[0..1000]of EVENTMSG;

    EventLog:Integer;

    PlayLog:Integer;

    hHook,hPlay:Integer;

    recOK:Integer;

    canPlay:Integer;

    bDelay:Bool;

implementation
```

```
{ $R *.DFM }

Function PlayProc(iCode:Integer;wParam:wParam;lParam:lParam):LRESULT;stdcall;
begin
    canPlay:=1;
    Result:=0;

    if iCode < 0 then    // 必须将消息传递到消息链的下一个接受单元

        Result := CallNextHookEx(hPlay,iCode,wParam,lParam)
    else if iCode = HC_SYSMODALON then
        canPlay:=0
    else if iCode = HC_SYSMODALOFF then
        canPlay:=1
    else if ((canPlay = 1 )and(iCode=HC_GETNEXT)) then begin
        if bDelay then begin
            bDelay:=False;
            Result:=50;
        end;
        pEventMSG(lParam)^:=EventArr[PlayLog];
    end
    else if ((canPlay = 1)and(iCode = HC_SKIP))then begin
```

```

    bDelay := True;

    PlayLog:=PlayLog+1;

end;

if PlayLog>=EventLog then begin

    UNHookWindowsHookEx(hPlay);

end;

end;

function HookProc(iCode:Integer;wParam:wParam;lParam:lParam):LRESULT;stdcall;

begin

    recOK:=1;

    Result:=0;

    if iCode < 0 then

        Result := CallNextHookEx(hHook,iCode,wParam,lParam)

    else if iCode = HC_SYSMODALON then

        recOK:=0

    else if iCode = HC_SYSMODALOFF then

        recOK:=1

    else if ((recOK>0) and (iCode = HC_ACTION)) then begin

        EventArr[EventLog]:=pEventMSG(lParam)^;

        EventLog:=EventLog+1;

```

```
    if EventLog>=1000 then begin
        UnHookWindowsHookEx(hHook);
    end;
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    Button1.Caption:= '纪录';
    Button2.Caption:= '停止';
    Button3.Caption:= '回放';
    Button4.Caption:= '范例';
    Button2.Enabled:=False;
    Button3.Enabled:=False;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    EventLog:=0;  // 建立键盘鼠标操作消息纪录链

    hHook:=SetWindowsHookEx(WH_JOURNALRECORD,HookProc,HInstance,0);
```

```
    Button2.Enabled:=True;

    Button1.Enabled:=False;

end;


procedure TForm1.Button2Click(Sender: TObject);
begin
    UnHookWindowsHookEx(hHook);

    hHook:=0;


    Button1.Enabled:=True;

    Button2.Enabled:=False;

    Button3.Enabled:=True;

end;


procedure TForm1.Button3Click(Sender: TObject);
begin
    PlayLog:=0; //建立键盘鼠标操作消息纪录回放链


    hPlay:=SetwindowsHookEx(WH_JOURNALPLAYBACK,PlayProc,
        HInstance,0);


    Button3.Enabled:=False;
```

```
end;
```

```
end.
```

二、驱动级模拟（绕过了windows消息）

有一些使用DirectX接口的游戏程序，它们在读取键盘操作时绕过了windows的消息机制，而使用DirectInput.这是因为有些游戏对实时性控制的要求比较高，比如赛车游戏，要求以最快速度响应键盘输入。而windows消息由于是队列形式的，消息在传递时会有不少延迟，有时1秒钟也就传递十几条消息，这个速度达不到游戏的要求。而DirectInput则绕过了windows消息，直接与键盘驱动程序打交道，效率当然提高了不少。因此也就造成，对这样的程序无论用PostMessage或者是keybd_event都不会有反应，因为这些函数都在较高层。对于这样的程序，只好用直接读写键盘端口的方法来模拟硬件事件了。要用这个方法模拟键盘，需要先了解一下键盘编程的相关知识。

在DOS时代，当用户按下或者放开一个键时，就会产生一个键盘中断(如果键盘中断是允许的)，这样程序会跳转到BIOS中的键盘中断处理程序去执行。打开windows的设备管理器，可以查看到键盘控制器由两个端口控制。其中&H60是数据端口，可以读出键盘数据，而&H64是控制端口，用来发出控制信号。也就是，从&H60号端口可以读此键盘的按键信息，当从这个端口读取一个字节，该字节的低7位就是按键的扫描码，而高1位则表示是按下键还是释放键。当按下键时，最高位为0，称为通码，当释放键时，最高位为1，称为断码。既然从这个端口读数据可以获得按键信息，那么向这个端口写入数据就可以模拟按键了！用过QbASIC4.5的朋友可能知道，QB中有个OUT命令可以向指定端口写入数据，而INP函数可以读取指定端口的数据。那我们先看看如果用QB该怎么写代码：

假如你想模拟按下一个键，这个键的扫描码为&H50，那就这样

```
OUT &H64,&HD2 '把数据&HD2发送到&H64端口。这是一个KBC指令，表示将要向键盘写入数据
```

```
OUT &H60,&H50 '把扫描码&H50发送到&H60端口，表示模拟按下扫描码为&H50的这个键
```

那么要释放这个键呢？像这样，发送该键的断码：

```
OUT &H64,&HD2 '把数据&HD2发送到&H64端口。这是一个KBC指令，表示将要向键盘写入数据
```

```
OUT &H60,(&H50 or &H80) '把扫描码&H50与数据&H80进行或运算，可以把它的高位置1，得到断码，表示释放这个键
```

好了，现在的问题就是在delphi中如何向端口写入数据了。因为在windows中，普通应用程序是无权操作端口的，于是我们就需要一个驱动程序来帮助我们实现。在这里我们可以使用一个组件WINIO来完成读写端口操作。什么是WINIO？WINIO是一个全免费的、无需注册

的、含源程序的WINDOWS2000端口操作驱动程序组件(可以到<http://www.internals.com/>上去下载)。它不仅可以操作端口，还可以操作内存；不仅能在VB下用，还可以在DELPHI、VC等其它环境下使用，性能特别优异。下载该组件，解压缩后可以看到几个文件夹，其中Release文件夹下的3个文件就是我们需要的，这3个文件是WinIo.sys(用于win xp下的驱动程序)，WINIO.VXD(用于win 98下的驱动程序)，WinIo.dll(封装函数的动态链接库)，我们只需要调用WinIo.dll中的函数，然后WinIo.dll就会安装并调用驱动程序来完成相应的功能。值得一提的是这个组件完全是绿色的，无需安装，你只需要把这3个文件复制到与你的程序相同的文件夹下就可以使用了。用法很简单，先用里面的InitializeWinIo函数安装驱动程序，然后就可以用GetPortVal来读取端口或者用SetPortVal来写入端口了。好，让我们来做一個驱动级的键盘模拟吧。先把winio的3个文件拷贝到你的程序的文件夹下。

下面给出使用WINIO模拟按键的单元和使用方法：

```
{*****}
```

```
unit MNwinio;
```

```
interface
```

```
const
```

```
KBC_KEY_CMD = $64; //键盘命令端口
```

```
KBC_KEY_DATA = $60; //键盘数据端口
```

```
implementation
```

```
function InitializeWinIo: Boolean; stdcall; external 'WinIo.dll' name 'InitializeWinIo';
```

```
{
```

```
    本函数初始化WioIO函数库。
```

```
    必须在调用所有其它功能函数之前调用本函数。
```

```
    如果函数调用成功，返回值为非零值。
```

如果调用失败，则返回值为0。

```
procedure TForm1.FormActivate(Sender: TObject); //通常在程序启动时调用
begin
    if InitializeWinIo = False then
    begin
        MessageBox(handle, '初始化失败! ', '提示', MB_OK + MB_IconError)
    end;
end;
}

function InstallWinIoDriver(pszWinIoDriverPath: PString; IsDemandLoaded: boolean
= false): Boolean; stdcall; external 'WinIo.dll' name 'InstallWinIoDriver';

function RemoveWinIoDriver: Boolean; stdcall; external 'WinIo.dll' name
'RemoveWinIoDriver';

function GetPortVal(PortAddr: Word; PortVal: PDWord; bSize: Byte): Boolean;
stdcall; external 'WinIo.dll' name 'GetPortVal';

function SetPortVal(PortAddr: Word; PortVal: DWord; bSize: Byte): Boolean;
stdcall; external 'WinIo.dll' name 'SetPortVal';
```



```
function GetPhysLong(PhysAddr: PByte; PhysVal: PDWord): Boolean; stdcall;  
    external 'WinIo.dll' name 'GetPhysLong';
```

```
function SetPhysLong(PhysAddr: PByte; PhysVal: DWord): Boolean; stdcall; external  
    'WinIo.dll' name 'SetPhysLong';
```

```
function MapPhysToLin(PhysAddr: PByte; PhysSize: DWord; PhysMemHandle: PHandle):  
    PByte; stdcall; external 'WinIo.dll' name 'MapPhysToLin';
```

```
function UnMapPhysicalMemory(PhysMemHandle: THandle; LinAddr: PByte): Boolean;  
    stdcall; external 'WinIo.dll' name 'UnmapPhysicalMemory';
```

```
procedure ShutdownWinIo; stdcall; external 'WinIo.dll' name 'ShutdownWinIo';
```

{ 本函数在内存中清除WinIO库

本函数必须在中止应用函数之前或者不再需要WinIO库时调用

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction); //通常在程序关闭时调用
```

```
begin
```

```
    ShutdownWinIo;
```

```
end;
```

```
}
```

```
{*****以上为WINIO.dll中API函数的调用*****}
```

```
procedure KBCWait4IBE; //等待键盘缓冲区为空
```

```
var
```

```
    dwVal: DWord;
```

```
begin
```

```
    repeat
```

```
        GetPortVal($64, @dwVal, 1);
```

{这句表示从&H64端口读取一个字节并把读出的数据放到变量dwVal中. GetPortVal函数的用法是 GetPortVal (端口号,存放读出数据的变量地址,读入的长度)}

```
    until (dwVal and $2) = 0;
```

{上面的是一个根据KBC规范写的过程，它的作用是在向键盘端口写入数据前等待一段时间，后面将会用到。}

```
end;
```

```
procedure MyKeyDown(vKeyCoad: Integer); // 这个用来模拟按下键，参数vKeyCoad传入按键的虚拟码
```

```
var
```

```
    btScancode: DWord;
```

```
begin
```

```
    btScancode := MapVirtualKey(vKeyCoad, 0);
```

```
    KBCWait4IBE; // 发送数据前应该先等待键盘缓冲区为空
```

```
    SetPortVal($64, $D2, 1); // 发送键盘写入命令
```

{SetPortVal函数用于向端口写入数据，它的用法是：SetPortVal(端口号,欲写入的数据，写入数据的长度)}

```

KBCWait4IBE;

SetPortVal($60, btScancode, 1); //写入按键信息,按下键

end;

procedure MyKeyUp(vKeyCoad: Integer); //这个用来模拟释放键, 参数vKeyCoad传入按键的虚拟码
var
    btScancode: DWord;
begin
    btScancode := MapVirtualKey(vKeyCoad, 0);

    KBCWait4IBE;

    SetPortVal($64, $D2, 1);

    KBCWait4IBE;

    SetPortVal($64, (btScancode or $80), 1);

end;

end.

{
    使用方法:

    如模拟按键 1

    MyKeyDown($31);

    Sleep(50);

    MyKeyUp($31);
}

```

分类: 键盘鼠标

好文要顶

关注我

收藏该文



北极星 - North Star

关注 - 0

粉丝 - 4

+加关注


1

推荐

0

反对

« 上一篇: 如何动态建立VFP能够打开的中文字段 dbf 表

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#) 网站首页。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【活动】优达学城正式发布“无人驾驶车工程师”课程

【推荐】融云发布 App 社交化白皮书 IM 提升活跃超 8 倍

【推荐】别再闷头写代码！找对工具，事半功倍，全能开发工具包用起来

【推荐】网易这个云产品做了15年才面世，1年吸引10万+开发者

| 结婚大事先看点评

点评为你准备婚纱摄影，婚庆服务，婚纱礼服一条龙服务。多种优惠，马上预约！

jh.dianping.com



最新IT新闻：

- 关于硅谷精英的真相：他们和你一样买不起房
 - 阿里云发布混合云安全解决方案
 - Windows Server 2016及System Center 2016正式商用
 - 创业十年，把六间房卖了26亿的刘岩说：创业是野蛮者的游戏
 - Magic Leap再爆新专利：头戴设备更轻便
- » [更多新闻...](#)

极光 智能推送全面升级 更快、更稳定、更成熟

了解更多

最新知识库文章：

- 循序渐进地代码重构
- 技术的正宗与野路子
- 陈皓：什么是工程师文化？
- 没那么难，谈CSS的设计模式
- 程序猿媳妇儿注意事项

» 更多知识库文章...