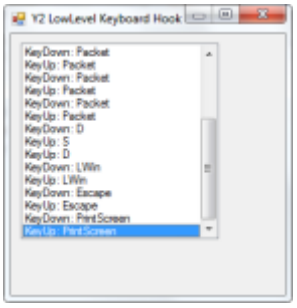
“

YinYang's Programing Blog

"Education is not a preparation for life; education is life itself"

C# – Hướng dẫn cài đặt Low-Level Keyboard Hook



(https://yinyangit.wordpress.com/2011/08/09/csharp-low-level-keyboard-hook/demo-y2-low-level-keyboard-hook-dot-net/)Để sử dụng hook trong .NET, bạn cần phải làm việc với unmanaged-code hay cụ thể là các Windows API (Win32 API) gồm: SetWindowsHookEx, CallNextHookEx và UnhookWindowsEx (Xem bài Giới thiệu kĩ thuật Hook và các khái niệm cơ bản (https://yinyangit.wordpress.com/2011/08/07/win32-introduction-to-hook-and-some-basic-concepts/)). Trong bài viết này, tôi sẽ hướng dẫn cách tạo một ứng dụng hook bàn phím toàn hệ thống.

Ánh xạ kiểu dữ liệu giữa Win32 và .NET

Một vài điểm cần lưu ý khi bạn làm việc với unmanaged-code là việc ánh xạ kiểu dữ liệu giữa Win32 và .NET. Trong Win32 bạn thấy có nhiều kiểu dữ liệu như LRESULT, HWND, HINSTANCE, HHOOK,… thực chất chúng là chỉ là kiểu số nguyên. Chúng được định nghĩa lại cho phù hợp với ý nghĩa và mục đích sử dụng. Trong .NET các kiểu này được ánh xạ tương ứng với kiểu IntPtr. Bạn cũng có thể dùng kiểu int và có thể ép kiểu trực tiếp giữa int và IntPtr.

Tương tự như vậy, con trỏ hàm trong Win32 được ánh xạ tương ứng với delegate. Vậy, với mỗi Hook procedure, bạn cũng phải tạo ra một delegate để làm callback function cho các hàm SetWindowsHookEx và CallNextHookEx.

Một điểm thuận lợi khi dùng các Win32 API trong .NET là bạn có thể tùy chọn kiểu trong khai báo các tham số của hàm. Như bạn có thể thấy trong ví dụ, việc thay thế giữa kiểu int và IntPtr là hợp lệ khi khai báo các hàm như SetWindowsHookEx, UnhookWindowsEx, CallNextHookEx,…

Các Win32 API Function

Sử dụng attribute [DllImport], ta sẽ thêm ba Win32 API cần thiết cho việc sử dụng hook.

```
[DllImport("user32.dll",SetLastError=true)]
private static extern IntPtr SetWindowsHookEx(int idHook,
    LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

[DllImport("user32.dll", SetLastError = true)]
private static extern bool UnhookWindowsHookEx(IntPtr hhk);

[DllImport("user32.dll", SetLastError = true)]
private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode,
    IntPtr wParam, IntPtr lParam);
```

Tham số SetLastError được đặt bằng true để chúng ta có thể lấy được lỗi trong trường hợp cài đặt hook thất bại.

Các Win32 Structure và Constants

Vì tôi sẽ tạo một ứng dụng hook low level keyboard nên cần phải có hằng số xác định kiểu hook là:

private const int WH_KEYBOARD_LL = 13;

Tiếp đến trong ví dụ này tôi chỉ bắt hai thông điệp KeyUp và KeyDown của bàn phím. Hai thông điệp này có giá trị tương ứng là:

private const int WM_KEYDOWN = 0x0100;

private const int WM_KEYUP = 0x101;

Ngoài ra Win32 còn định nghĩa một structure dùng để lưu thông tin của các sự kiện bàn phím ở mức thấp. Structure này được đặt tên là KBDLLHOOKSTRUCT, tuy nhiên theo chuẩn đặt tên của C#, tôi sẽ đặt một tên rõ ràng hơn cho structure này.

```
[StructLayout(LayoutKind.Sequential)]
public struct KeyboardHookStruct
{
    public int VirtualKeyCode;
    public int ScanCode;
    public int Flags;
    public int Time;
    public int ExtraInfo;
}
```

Tham khảo (MSDN): KBDLLHOOKSTRUCT Structure (http://msdn.microsoft.com/en-us/library/ms644967(v=vs.85).aspx)

LowLevelKeyboardProc Callback Function (Hook procedure)

Khi có sự kiện nhấn phím, hàm callback này sẽ được gọi bởi hệ thống. Cú pháp định nghĩa của hàm này có dạng:

LRESULT CALLBACK LowLevelKeyboardProc(
__in int nCode,
__in WPARAM wParam,
__in LPARAM lParam
);

Trong đó:

– nCode: Xác định hook procedure có thực hiện xử lý không. Nếu giá trị nCode nhỏ hơn 0, hook procedure sẽ bỏ qua xử lý thông điệp. Nếu giá trị bằng 0 (HC_ACTION) có nghĩa là hai tham số wParam và lParam sẽ chứa thông tin về thông điệp bàn phím.

– Value	Meaning
HC_ACTION0	The *wParam* and *lParam* parameters contain information about a keyboard message.

– wParam: kiểu thông điệp bàn phím, bao gồm: WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, WM_SYSKEYUP.

– lParam: con trỏ tới **KBDLLHOOKSTRUCT** (http://msdn.microsoft.com/en-us/library/ms644967(v=vs.85).aspx) structure.

Trong ví dụ này tôi sẽ định nghĩa hook procedure này như sau:

```
1  private IntPtr KeyboardHookProc(int nCode, IntPtr wParam, IntPtr lParam)
2  {
3      if (nCode >= 0)
4      {
5      // do something
6      }
7
8      return CallNextHookEx(_hookHandle, nCode, wParam, lParam);
9  }
```

Và tạo một delegate tương ứng để làm nhiệm vụ "con trỏ hàm":

public delegate IntPtr KeyboardHookDelegate(int nCode, IntPtr wParam, IntPtr lParam);

Các điểm cần lưu ý

Lấy handle của tập tin chứa hook procedure

Khi cài đặt global hook, bạn phải có được handle của tập tin PE chứa hook procedure. Trong .NET, tập tin này là một module (một assembly có thể gồm nhiều module). Bạn có thể lấy handle của tập tin bằng cách dùng Win32 API GetModuleHandle, tuy nhiên một cách khác mà .NET hỗ trợ là dùng phương thức static Marshal.GetHINSTANCE().

Dòng lệnh sau cho ta thấy cách để lấy handle của module chính (chứa hook procedure) trong assembly:

```
IntPtr hInstance = Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().GetModules()[0]);
```

Nhận thông báo lỗi của Win32

Để lấy được lỗi khi cài đặt hook thất bại, bạn phải tham số SetLastError bằng true trong attribute [DllImport] của hàm API cần bắt lỗi. Khi đó để lấy được lấy mã lỗi, thay vì dùng Win32 API GetLastError ta có thể dùng phương thức static GetLastWin32Error() của lớp System.Runtime.InteropServices.Marshal.

Khi đã có được mã lỗi, thay vì tìm kiếm bảng dò mã lỗi thì bạn có thể dùng lớp Win32Exception trong namespace System.ComponentModel. Constructor của Win32Exception nhận một số chỉ mã lỗi và sẽ chuyển sang thông điệp lỗi tương ứng.

Ví dụ hàm SetWindowsHookEx trả về handle của hook là 0 nếu cài đặt thất bại, bạn có thể viết như sau để xem nguyên nhân của việc thất bại:

```
if (_hookHandle == IntPtr.Zero)
throw new Win32Exception(Marshal.GetLastWin32Error());
```

Cài đặt Hook

Phần cài đặt rất đơn giản, tôi tạo một phương thức public Install() để gọi phương thức cài đặt hook thực sự là SetupHook(). Sau khi gọi SetupHook(), phương thức Install() sẽ kiểm tra hook handle và sẽ ném ra một Win32Exception nếu cài đặt hook thất bại. Như bạn thấy phương thức SetupHook() chỉ đơn giản là gọi hàm Win32 API SetWindowsHookEx:

```
private KeyboardHookDelegate _hookProc;
private IntPtr _hookHandle = IntPtr.Zero;

// ...

public void Install()
{
    _hookProc = KeyboardHookProc;
    _hookHandle = SetupHook(_hookProc);

    if (_hookHandle == IntPtr.Zero)
        throw new Win32Exception(Marshal.GetLastWin32Error());
}

private IntPtr SetupHook(KeyboardHookDelegate hookProc)
{
    IntPtr hInstance = Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().Ge

    return SetWindowsHookEx(WH_KEYBOARD_LL, hookProc, hInstance, 0);
}
```

Gỡ bỏ Hook

Bạn có thể thấy tôi có vẻ tạo một phương thức dư thừa khi chỉ đơn giản gọi lại một hàm khác. Thực sự phương thức Uninstall() sau chỉ khác nhau về tham số so với hàm UnhookWindowsHookEx:

```
public void Uninstall()
{
    UnhookWindowsHookEx(_hookHandle);
}
```

Phương thức này cũng như phương thức Install() trên được tôi đặt trong một lớp riêng là Y2KeyboardHook. Sẽ an toàn hơn khi bạn tạo hạn chế việc gọi trực tiếp đến các Win32 API do việc truyền tham số sai có thể gây ra nguy hiểm trong một số trường hợp. Việc encapsulation các hàm API cũng giúp cho việc sử dụng lớp bạn tạo ra dễ dàng và thân thiện hơn, như vậy khi sử dụng, bạn không cần phải nhớ tất cả các tham số không cần thiết.

Để bảo đảm việc gỡ bỏ hook được thực hiện, tôi viết thêm một destructor và gọi phương thức Uninstall() trong đó. Destructor sẽ tự động được gọi khi đối tượng bị hủy, bạn không cần phải gọi trực tiếp destructor:

```
// destructor
~Y2KeyboardHook()
{
    Uninstall();
}
```

Thêm các event

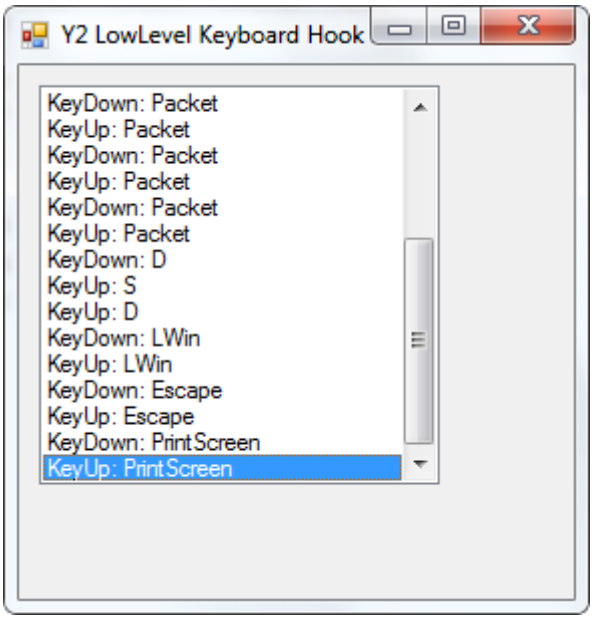
Sẽ tiện lợi hơn nếu bạn thêm các event cho lớp để sử dụng như một control. Net đã cung cấp sẵn delegate KeyEventHandler. Nếu bạn cần tìm hiểu về cách tạo event, có thể tham khảo hướng dẫn tại: Tạo, sử dụng và quản lý Event trong C# (https://yinyangit.wordpress.com/2011/05/24/dotnet-event-and-event-handling-in-csharp/).

public event KeyEventHandler KeyDown;
public event KeyEventHandler KeyUp;

Trong hook procedure, ta sẽ xử lý để kích hoạt event nếu thông điệp tương ứng với event đó xảy ra:

```
1  private IntPtr KeyboardHookProc(int nCode, IntPtr wParam, IntPtr lParam)
2  {
3      if (nCode >= 0)
4      {
5          KeyboardHookStruct kbStruct = (KeyboardHookStruct)Marshal.PtrToStructu
6
7          if (wParam == (IntPtr)WM_KEYDOWN)
8          {
9              if (KeyDown != null)
10                 KeyDown(null, new KeyEventArgs((Keys)kbStruct.VirtualKeyCode))
11         }
12         else if (wParam == (IntPtr)WM_KEYUP)
13         {
14             if (KeyUp != null)
15                 KeyUp(null, new KeyEventArgs((Keys)kbStruct.VirtualKeyCode));
16         }
17     }
18
19     return CallNextHookEx(_hookHandle, nCode, wParam, lParam);
20 }
```

Ví dụ hoàn chỉnh



(https://yinyangit.wordpress.com/2011/08/09/csharp-low-level-keyboard-hook/demo-y2-low-level-keyboard-hook-dot-net/)

Để thực hiện ví dụ này, bạn hãy tạo một dự án Windows Forms Application. Thêm lớp **Y2KeyboardHook** sau vào dự án:

```
1  using System;
2  using System.Runtime.InteropServices;
3  using System.Diagnostics;
4  using System.Windows.Forms;
5  using System.Reflection;
6  using System.ComponentModel;
7
8  namespace HookApp
9  {
10
11     class Y2KeyboardHook
12     {
13
14         #region Win32 API Functions and Constants
15
16         [DllImport("user32.dll", SetLastError = true)]
17         private static extern IntPtr SetWindowsHookEx(int idHook,
18             KeyboardHookDelegate lpfn, IntPtr hMod, int dwThreadId);
19
20         [DllImport("user32.dll", SetLastError = true)]
21         private static extern bool UnhookWindowsHookEx(IntPtr hhk);
22
23         [DllImport("user32.dll", SetLastError = true)]
24         private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode,
25             IntPtr wParam, IntPtr lParam);
26
27         [DllImport("kernel32.dll")]
28         private static extern IntPtr GetModuleHandle(string lpModuleName);
29
30         private const int WH_KEYBOARD_LL = 13;
31
```

```
32          private const int WM_KEYDOWN = 0x0100;
33          private const int WM_KEYUP = 0x101;
34
35          #endregion
36
37          private KeyboardHookDelegate _hookProc;
38          private IntPtr _hookHandle = IntPtr.Zero;
39
40          public delegate IntPtr KeyboardHookDelegate(int nCode, IntPtr wParam,
41
42          [StructLayout(LayoutKind.Sequential)]
43          public struct KeyboardHookStruct
44          {
45              public int VirtualKeyCode;
46              public int ScanCode;
47              public int Flags;
48              public int Time;
49              public int ExtraInfo;
50          }
51
52          #region Keyboard Events
53
54          public event KeyEventHandler KeyDown;
55          public event KeyEventHandler KeyUp;
56
57          #endregion
58
59          // destructor
60          ~Y2KeyboardHook()
61          {
62              Uninstall();
63          }
64
65          public void Install()
66          {
67              _hookProc = KeyboardHookProc;
68              _hookHandle = SetupHook(_hookProc);
69
70              if (_hookHandle == IntPtr.Zero)
71                  throw new Win32Exception(Marshal.GetLastWin32Error());
72          }
73          private IntPtr SetupHook(KeyboardHookDelegate hookProc)
74          {
75              IntPtr hInstance = Marshal.GetHINSTANCE(Assembly.GetExecutingAsse
76
77              return SetWindowsHookEx(WH_KEYBOARD_LL, hookProc, hInstance, 0);
78          }
79
80          private IntPtr KeyboardHookProc(int nCode, IntPtr wParam, IntPtr lPar
81          {
82              if (nCode >= 0)
83              {
84                  KeyboardHookStruct kbStruct = (KeyboardHookStruct)Marshal.Ptr
85
86                  if (wParam == (IntPtr)WM_KEYDOWN)
```

```
87                  {
88                      if (KeyDown != null)
89                          KeyDown(null, new KeyEventArgs((Keys)kbStruct.Virtual
90                  }
91                  else if (wParam == (IntPtr)WM_KEYUP)
92                  {
93                      if (KeyUp != null)
94                          KeyUp(null, new KeyEventArgs((Keys)kbStruct.VirtualKe
95                  }
96              }
97
98              return CallNextHookEx(_hookHandle, nCode, wParam, lParam);
99          }
100
101         public void Uninstall()
102         {
103             UnhookWindowsHookEx(_hookHandle);
104         }
105
106     }
107 }
```

Trong tập tin **Form1.cs**, bạn hãy sửa lại như sau:

```
using System;
using System.Windows.Forms;
using HookApp;
using System.Drawing;

namespace WindowsFormApplication1
{
    public partial class Form1 : Form
    {
        Y2KeyboardHook _keyboardHook;

        public Form1()
        {
            InitializeComponent();

            this.TopMost = true;

            ListBox listBox1 = new ListBox();
            listBox1.Location = new Point(10, 10);
            listBox1.Size = new Size(200, 200);

            this.Controls.Add(listBox1);

            _keyboardHook = new Y2KeyboardHook();
            _keyboardHook.Install();

            _keyboardHook.KeyDown += (sender, e) =>
                {
                    listBox1.Items.Add("KeyDown: " + e.KeyCode);

                    listBox1.SelectedIndex = listBox1.Items.Count - 1;
                };

            _keyboardHook.KeyUp += (sender, e) =>
                {
                    listBox1.Items.Add("KeyUp: " + e.KeyCode);

                    listBox1.SelectedIndex = listBox1.Items.Count - 1;
                };
        }

    }
}
```

Đoạn mã mới thêm vào Form1 sẽ thêm một ListBox vào Form, đồng thời tạo ra một đối tượng Y2KeyboardHook và xử lý hai event KeyDown và KeyDown bằng lambda expression. Bạn có thể chuyển sang ứng dụng khác và nhấn phím bất kì, các sự kiện tương ứng sẽ được thêm vào ListBox.

Kết luận