



Separación del flujo de información en un dispositivo de entrada bajo Microsoft Windows.

Iván Cabrera Altamirano, Tanibet Pérez de los Santos Mondragón, *Sistemas Colaborativos Distribuidos.*

Dra. Sonia Guadalupe Mendoza Chapa, CINVESTAV-IPN. Departamento de Computación

Resumen— El presente documento describe la implementación de una aplicación en Microsoft Windows capaz de separar el flujo de información proveniente de los dispositivos de entrada conectados al sistema.

Con el fin de mostrar de manera clara el propósito de este trabajo hemos conectado varios ratones e implementado una interfaz sencilla de dibujo donde podrá visualizarse la operación de los múltiples dispositivos.

Para la realización de este trabajo hemos programado nuestra aplicación en Microsoft Visual C# 2008 Express Edition utilizando la API de Windows denominada Windows User Interface: Raw Input.

I. INTRODUCCIÓN

EN los sistemas de manejo de ventanas tradicionales no es posible tener más de un dispositivo del mismo tipo actuando de manera independiente, es decir podemos conectar 2 teclados al sistema, y este los reconocerá y ambos funcionaran, sin embargo funcionaran como un solo teclado, ya que el flujo de información producido por estos será visto por el sistema como un flujo único.

Esta característica antes descrita nos presenta algunas limitantes en el desarrollo de aplicaciones con características de trabajo colaborativo, dado que no es posible asignar un dispositivo de entrada a cada participante y este pueda realizar una acción en el sistema, un ejemplo claro serian los videojuegos.

Dado que el sistema no separa el flujo de información para los dispositivos de entrada de forma nativa, utilizaremos llamadas al sistema contenidas API de Microsoft Windows para poder realizar esta tarea.

Existen otro tipo de soluciones, a nivel del sistema operativo, donde se busca modificar el controlador del dispositivo para que Microsoft Windows lo reconozca como un dispositivo diferente, sin embargo esta solución no será tratada en este trabajo.

II. ANTECEDENTES

Para la realización de este trabajo haremos uso de la interfaz de programación de aplicaciones (por sus siglas en inglés API) de Microsoft Windows, mas en especifico de las funciones para el procesamiento de entrada de datos crudos (RawInput) pertenecientes a la categoría de interface de usuario (Windows User Interface). En la figura 1 podemos apreciar la estructura de la API de Windows.

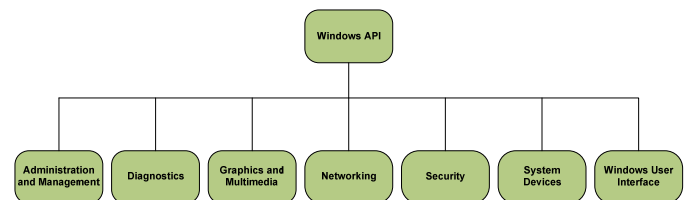


Figura 1. Estructura de la API de Windows

La API de Microsoft Windows, nos permite tener aplicaciones que exploten al máximo el poder del sistema operativo. Usando esta interfaz, uno puede desarrollar aplicaciones que se ejecuten satisfactoriamente en todas las versiones de Microsoft Windows.

Las funciones pertenecientes a la categoría de interface de usuario (Windows User Interface) nos permiten que las aplicaciones desarrolladas puedan crear y manejar una interface de usuario.

Estas funciones crean y usan ventanas para mostrar información, preguntar por información de entrada, y de llevar todo lo necesario para soportar la interacción de la aplicación con el usuario.

Las aplicaciones definen un comportamiento general y una determinada apariencia de sus ventanas creando clases con sus correspondientes procedimientos para las ventanas.

En estas clases se define como es que la ventana debe responder a un evento, como puede ser al clic del ratón.

Las aplicaciones generan salida para una ventana utilizando funciones de la interfaz de dispositivos gráficos (por sus siglas en inglés GDI).

Dado que todas las ventanas comparten el área de visualización, las aplicaciones no reciben acceso a la pantalla entera. En lugar de esto, el sistema alinea y adjunta toda la salida para ajustarla dentro de la correspondiente ventana.

Las aplicaciones pueden dibujar en una ventana en respuesta a una petición del sistema o mientras el sistema procesa mensajes de entrada.

Cuando el tamaño o posición de la ventana cambia, el sistema típicamente envía un mensaje a la aplicación haciendo la petición de que dibuje cualquier área no expuesta de esta ventana.

Las aplicaciones reciben la entrada del teclado y del ratón en forma de mensajes. El sistema traduce los movimientos y clics del ratón, y el presionado de las teclas en mensajes de entrada para posteriormente depositar estos mensajes en la cola de mensajes propia de cada aplicación.

El sistema automáticamente provee una cola de mensajes para cada aplicación. La aplicación utiliza las funciones para el manejo de mensajes con el fin de extraer mensajes de la cola correspondiente y despachar al procedimiento adecuado de la ventana para su procesamiento.

Por último las aplicaciones pueden procesar la entrada del ratón y del teclado directamente o dejarle al sistema que traduzca esta entrada de bajo nivel en mensajes de comando.

Las aplicaciones comúnmente responden a los mensajes de comando preguntando al usuario por información adicional con cajas de dialogo.

Una caja de dialogo es una ventana temporal que despliega información o pide entrada de datos. Una caja de dialogo comúnmente incluye controles, que representan botones y cajas mediante las cuales el usuario provee información. Existen controles para escribir texto, seleccionar componentes de una lista, entre otros.

Una aplicación puede compartir datos útiles, como mapas de bits, iconos, fuentes y cadenas de caracteres, agregando estos datos como recursos al archivo de la aplicación o mediante una DLL.

Existen gran cantidad de dispositivos de entrada para el usuario aparte del tradicional teclado y ratón. Por ejemplo, el usuario puede introducir datos a través de un control de juegos, una pantalla táctil, un lector de código de barras, un micrófono u algún otro dispositivo que permiten una gran flexibilidad en la entrada del usuario.

Estos dispositivos colectivamente son conocidos como dispositivos de interface humana (por sus siglas en ingles HID). El API de *RawInput* provee una forma estable y robusta de aceptar entrada cruda desde cualquier dispositivo HID, incluyendo el teclado y ratón.

Previamente, el teclado y el ratón típicamente generan los datos de entrada. El sistema interpreta los datos provenientes de estos dispositivos de una manera en la que elimina los detalles específicos del dispositivo de la información cruda.

Por ejemplo, el teclado genera un código de escaneo específico para el dispositivo pero el sistema provee a la aplicación con un código de escaneo virtual.

El modelo de entradas crudas (*RawInput*) es diferente del modelo original de Microsoft Windows para procesar datos de entrada provenientes del teclado y mouse.

En el modelo de entrada de datos original, una aplicación recibe entrada independiente del dispositivo en forma de mensajes que son enviados a su respectiva ventana.

Los mensajes típicos para este modelo son:

- WM_CHAR
- WM_MOUSEMOVE
- WM_APPCOMMAND

En contraste para la entrada de datos crudos la aplicación debe registrar los dispositivos de los cuales quiere obtener datos de entrada. También la aplicación obtiene los datos de entrada crudos a través del mensaje *WM_INPUT*.

Hay distintas ventajas del modelo de entrada de datos crudos:

- La aplicación no necesita detectar o abrir el dispositivo de entrada.
- La aplicación obtiene los datos directamente del dispositivo, y procesa estos datos de acuerdo a sus necesidades.
- La aplicación puede distinguir la fuente de entrada, aun cuando provenga de dispositivos del mismo tipo. Por ejemplo 2 ratones.
- La aplicación maneja el tráfico de datos obteniendo datos solo de los dispositivos que desee trabajar.
- Los dispositivos HID pueden ser usados desde el momento en que estén disponibles en el mercado, sin tener que esperar por nuevos tipos de mensaje o un sistema operativo actualizado que tenga los nuevos comandos en *WM_APPCOMMAND*.

Por omisión, ninguna aplicación recibe entrada de datos crudos. Para recibir datos crudos de un dispositivo, la aplicación debe registrar el dispositivo.

Registro de Dispositivos *RawInput*

Para registrar dispositivos, la aplicación primero debe crear un arreglo de estructura *RAWINPUTDEVICE* que especifican una colección de alto nivel para el dispositivo que lo requiera.

La aplicación debe hacer una llamada a la función *RegisterRawInputDevices* para registrar los dispositivos deseados.

Es de notarse que una aplicación puede registrar un dispositivo que actualmente no está conectado al sistema. Cuando el dispositivo es conectado, el administrador de Windows automáticamente enviara la entrada cruda del dispositivo a la aplicación.

Para obtener la lista de los dispositivos de entrada cruda del sistema, la aplicación debe llamar a la función *GetRawInputDeviceList*.

Usando el parámetro *hDevice* desde esta llamada, la aplicación llama a la función *GetRawInputDeviceInfo* para obtener la información del dispositivo.

Mediante el miembro *dwFlags* de la estructura *RAWINPUTDEVICE*, una aplicación puede seleccionar los dispositivos de los cuales desea escuchar y e ignorar los dispositivos de los que no desea recibir entrada.

Para obtener el estado de un dispositivo en una aplicación se puede llamar a la función *GetRegisteredRawInputDevices* en cualquier instante de tiempo.

Leyendo *RawInput*

Cuando una aplicación recibe entrada cruda, su cola de mensajes obtiene un mensaje *WM_INPUT*. Una aplicación puede recibir datos cuando esta ejecutándose en primer o segundo plano.

Existen 2 maneras de leer datos crudos:

- **Unbuffered Mode (Tradicional):** Obtiene los datos crudos a través de una estructura *RawInput* en cualquier instante de tiempo y esta forma es adecuada para la mayoría de los dispositivos HID. La aplicación llama a la función *GetMessage* para obtener el mensaje *WM_INPUT*. Entonces la aplicación llamada a la función *GetRawInputData* usando el manejador *RawInput*.
- **Buffered Mode:** Obtiene un arreglo de estructuras *RawInput* en cualquier instante de tiempo. Esta forma es adecuada para dispositivos que producen largas cantidades de entrada cruda. En este método, la aplicación realiza una llamada a la función *GetRawInputBuffer* para obtener un arreglo de estructuras *RawInput*. La macro *NextRawInputBlock*

es usada para recorrer el arreglo de estructuras *RawInput*.

Para interpretar la entrada cruda, es necesario tener información detallada del dispositivo HID. La aplicación obtiene la información mediante la llamada a la función *GetRawInputDeviceInfo* con el manejador del dispositivo.

III. IMPLEMENTACIÓN

Para la implementación de nuestra aplicación se utilizo Microsoft Visual C# 2008 Express Edition.

La aplicación desarrollada consta de 3 clases:

- ***RawInput*:** Esta clase obtiene todos los dispositivos de tipo *Raw* (Teclado, Ratón, etc) y los almacena en un arreglo llamado *Devices*.

Esta clase define la función *GetRawInputDevices* la cual hace la tarea principal de la clase, definida anteriormente. Además hace uso de llamadas al sistema contenidas en la DLL "user32.dll".

- ***RawMouseInput*:** Esta clase hereda de *RawInput*. De los dispositivos que obtuvo anteriormente *RawInput*, los toma y busca los de tipo *Mouse*, y los almacena en un arreglo llamado *mice*.

Esta clase define *GetRawInputMice* la cual hace la tarea principal de la clase, definida anteriormente.

Además define las siguientes funciones:

- ***RegisterForWM_INPUT*:** Registra la aplicación (ventana) para recibir mensajes *WM_INPUT* del ratón.
- ***UpdateRawMouse*:** Actualiza el estado del ratón dado un manejador de ratón con datos crudos "*RawMouse*".
- ***RawMouse*:** Esta clase define un "ratón", con sus coordenadas X, Y y Z, nombre, manejador, etc. El programa principal utiliza objetos de este tipo asociados a localidades del arreglo *mice* contenido en *RawMouseInput*.

Cada ratón físico es definido por un objeto de tipo *RawMouse* (al podríamos denominar como un ratón lógico), y este ratón lógico es lo que utiliza el programa para dibujar en pantalla.

El programa principal `mouses2`, funciona de la siguiente manera:

En el constructor del objeto:

- Creo una instancia de la clase `RawMouseInput`:
`_rmInput = new RawMouseInput();`
- Registro el dispositivo `RawInput` para esta ventana:
`_rmInput.RegisterForWM_INPUT(this.Handle);`
- Actualizo la pantalla:
`UpdateDisplay();`

Al recibir un mensaje `WM_INPUT`:

```
switch (m.Msg) {
    case WM_INPUT:
        // Actualizo el estado del mouse.
        _rmInput.UpdateRawMouse(m.LParam);
        // Actualizo la pantalla
        UpdateDisplay();
        break;
}
```

La función `updateDisplay()` quedo conformada así:

Es necesario crear los objetos `RawMouse`.

```
// Creo los ratones
RawMouse mouse1 = (RawMouse)_rmInput.Mice[1];
RawMouse mouse2 = (RawMouse)_rmInput.Mice[3];
```

Los dispositivos enumerados por el sistema fueron:

```
\\?\Root#RDP_MOU#0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}

\\?\HID#{00001124-0000-1000-8000-00805f9b34fb}_VID&0002046d_PID&b006&Col01#9&bfb42fe&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}

\\?\HID#VID_0A5C&PID_4503&Col01#7&24cac6e5&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}

\\?\ACPI#PNP0F13#4&707880&0#{378de44c-56ef-11d1-bc8c-00a0c91405dd}
```

Sin embargo estos 4 dispositivos no corresponden a dispositivos físicos, en este caso solo el dispositivo 1 y el dispositivo 3 son entradas asociadas a dispositivos conectados.

Por lo tanto al objeto `mouse1` se le asigno el registro asociado al índice 1 del arreglo `mice`, mientras que al objeto `mouse2` se le asigno el registro asociado al índice 3.

Y dibujo en el objeto grafico que se ha agregado a la forma principal, mediante la llamada a la siguiente función:

```
g.FillEllipse(Brush, new Rectangle(X,Y,Width,Height));
```

De acuerdo al botón presionado y al mouse que este dibujando la aplicación pintara en la pantalla con diferentes colores.

Para acceder a la coordenada X del mouse1 lo haríamos de la siguiente forma:

```
mouse1.X;
```

de la misma forma para Y y Z así como para `mouse2`.

```
mouse1.Y; mouse1.Z;
```

IV. RESULTADOS

La aplicación al iniciarse situa dos puntos de distinto color al centro de la pantalla, cada una para un ratón, tal y como se muestra en la figura 2.

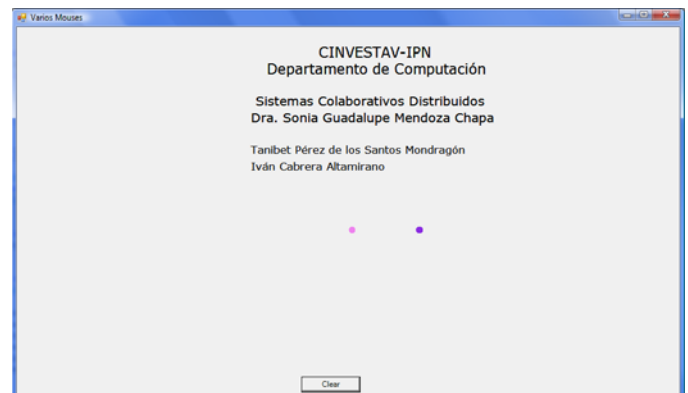


Figura 2. Pantalla inicial

La ejecución de nuestra aplicación se muestra en la figura 3. Con distintos colores para cada evento es posible distinguir entre los los diferentes ratones y los eventos (clicks) asociados a cada ratón.

Los colores utilizados fueron:

Mouse 1: Violeta
 Mouse 1 – Clic izquierdo: Amarillo
 Mouse 1 – Clic derecho: Rojo

Mouse 2: AzulVioleta
 Mouse 2 – Clic Izquierdo: Verde
 Mouse 2 – Clic Derecho: Naranja

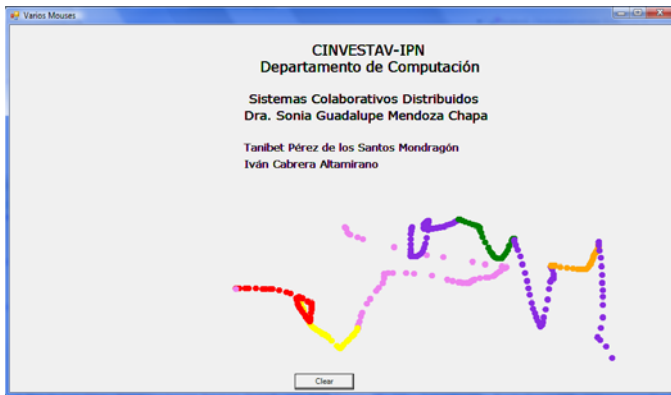


Figura 3. Ejecución de aplicación que maneja 2 ratones.

Al presionar el botón clear, el area de dibujo se borra.

V. DISCUSIÓN

El sistema implementado dio los resultados esperados, fue posible separar el flujo de información de los dispositivos de entrada y darles un uso determinado.

Sin embargo una limitante de esta implementación es que aunque puedan manejarse distintos dispositivos de entrada, solo es posible efectuar las distintas acciones en la misma ventana que tiene los dispositivos registrados, no es posible que estos múltiples dispositivos actúen sobre cualquier ventana.

Otra limitación es que el sistema solo admite 2 dispositivos de entrada de tipo ratón, sin embargo este puede ser escalable al máximo soportado el sistema, aunque el desempeño del sistema se vería notablemente disminuido.

Una forma de sobrellevar la limitación del área de trabajo es utilizando soluciones tales como las presentadas por Michael Westergaard en el sitio web:

<http://cpnmouse.sourceforge.net/>

Esta solución consiste en modificar el archivo controlador para dispositivos de entrada e instalarlo para cada dispositivo, con esto es posible que cada ratón pueda tomar control sobre cualquier objeto visible en la pantalla.