

# IO.DLL

## Resumen

IO.DLL permite separar las operaciones sobre puertos de E/S en Windows 95/98/2000/XP utilizando la misma librería.

## Introducción

Antes de la aparición de Windows, era relativamente simple acceder a los puertos de E/S en un PC típico. En efecto, casi cada lenguaje soportaba un comando especial para hacerlo. A medida que Windows emergía y se desarrollaba gradualmente, este comportamiento no se podía tolerar por la habilidad del sistema operativo para virtualizar el hardware.

Virtualizar el hardware significa que una aplicación (la típica ventana DOS de Windows) cree que está tratando directamente con el dispositivo físico, pero en realidad está tratando con un driver que simula al hardware, pasando los datos hacia atrás y hacia delante según proceda. Así es como puede abrir docenas de ventanas DOS en su sesión Windows, cada una con la extraña sensación de tener acceso exclusivo a periféricos como el adaptador de video, teclado, tarjeta de sonido e impresora.

Si alguien fuera a volcar datos de forma rudimentaria en un puerto de E/S que Windows pensase que tuviese bajo control total, podría ocurrir la "mala cosa oficial", que es toparse con la severidad del hardware dependiendo del dispositivo exacto al que se está accediendo. En realidad, con la virtualización que se acaba de comentar, es bastante improbable que Windows permita que ocurra algo desagradable.

En realidad Windows 95/98 permite la ejecución de operaciones de E/S a nivel de la aplicación, aunque tendría que encontrar un lenguaje que soportara esto directamente. Típicamente el programador tendrá que recurrir al lenguaje ensamblador para estos controles a bajo nivel. Si sabe lo que está haciendo, esto puede ser una forma rápida y fácil de acceder a los puertos de E/S. Por supuesto, no todo el mundo sabe, o quiere aprender a programar en ensamblador 80x86 solo porque quieren encender una bombilla desde su ordenador. Sin embargo, la desgana para aprender lenguaje ensamblador llega a ser algo trivial cuando se afronta con su hermano mayor 9x's.

Siendo Windows NT/2000/XP el sistema operativo seguro que es, no permite de ningún modo realizar operaciones en los puertos de E/S a nivel de la aplicación. Un programa que contenga instrucciones ensamblador IN y OUT que funciona perfectamente en Windows 95/98, fallará estrepitosamente cuando se pruebe en Windows NT/2000/XP.

Sin embargo, Windows NT/2000/XP permite instrucciones de E/S en con sus drivers modo kernel. Un driver modo kernel ejecuta en el nivel más privilegiado del procesador, y puede hacer lo que se le pida, incluyendo el cierre del sistema después de repararlo, así que escribir un driver modo kernel no es algo sencillo.

Si fuera a hacerlo por sí mismo leyendo la documentación del ddk (driver developer kit) de Windows NT/2000/XP y reconstruye un driver que pudiera llamar por su aplicación para hacer las instrucciones de E/S en nombre de su aplicación, probablemente notaría algo no muy agradable—este tipo de acceso es lamentablemente lento. La llamada desde el nivel de aplicación al nivel de sistema, aproximadamente requiere un milisegundo. Compare esto con el microsegundo que suele requerir un acceso E/S normal. Para aumentar la ofensa, está al antojo del sistema operativo. Si tiene tareas que cree que son de mayor prioridad que su modesta llamada al driver, la ejecutará, pero asignándole unos tiempos de ejecución casi imposibles.

Obviamente, escribir un driver que actúe como proxy para las llamadas de E/S no es la solución más ideal. Sin embargo hay una solución para NT/2000/XP que hace lo mismo que el lenguaje ensamblador contenido en 95/98.

Como se dijo, un driver modo kernel puede hacer lo que quiera. El problema aquí es que si otro driver modo kernel quita el acceso de aplicaciones a los puertos de E/S, debería ser posible que otro driver modo kernel volviera a ponerlo. Aquí es donde IO.DLL introduce la imaginación.

## Licencia

IO.DLL es completamente libre! No obstante, no podría:

- Cobrárselo a otros de alguna forma. Por ejemplo, no puede venderlo como un producto independiente.
- Reclamar que es de su propiedad.

El autor (que soy yo) tampoco se considerará obligado ante fallos de io.dll al ejecutarse. Como con la mayoría del material libre, depende de usted mismo.

## Código Fuente y Modificaciones Especiales

El código fuente está disponible por 1000 \$USA.

Estoy dispuesto a trabajar con gente que necesite una modificación especial de IO.DLL. Por ejemplo, podría tener unos requisitos de tiempo muy estrictos que solo pudieran realizarse en modo kernel. Por un precio, modificaré IO.DLL y/o el driver modo kernel embebido para la tarea que tenga que realizar.

## Descripción de IO.DLL

IO.DLL proporciona un conjunto de comandos útiles para leer y escribir en los puertos de E/S. Esos comandos son coherentes entre 95/98 y NT/2000/XP. Es más, no es necesario que el programador aprenda lenguaje ensamblador o se pelee con los drivers modo kernel. Simplemente lanzará la DLL y llamará a sus funciones. Es así de fácil. Windows NT/2000/XP está acoplado con un pequeño driver modo kernel que libera los puertos para la aplicación a medida que los va necesitando. Este driver está embebido en la DLL y se instala con Windows NT/2000/XP si está destinado a ser el sistema operativo que hay por debajo.

Debido a la sobrecarga implicada al incorporar dinámicamente la librería IO.DLL, y las funciones optimizadas que contiene, el acceso a los puertos de E/S es casi tan rápido como si se hiciese con código ensamblador dentro de su aplicación. Esto se mantiene tanto para Windows 95/98 como para Windows NT/2000/XP.

Antes de continuar, es prudente mencionar que la técnica empleada en IO.DLL para liberar los puertos en el nivel de aplicación, no es estrictamente hablando, la manera adecuada de hacer las cosas. La forma apropiada es tener un driver de dispositivo virtual para Windows 95/98 y un driver modo kernel para Windows NT/2000/XP. Esto no es muy práctico para mucha gente, e incluso ni es necesario. Hay varios productos comerciales en el mercado que hacen exactamente lo mismo que IO.DLL. De todas formas algunos de ellos no contienen buenas explicaciones sobre su producto, además de costar 500 \$ USA o más.

## Descargar

[io.dll](#) 49k

[io.zip](#) 31k (incluye la versión texto de esta documentación)

## Prototipos en C

```
void PortOut(short int Port, char Data);
void PortWordOut(short int Port, short int Data);
void PortDWordOut(short int Port, int Data);
char PortIn(short int Port);
short int PortWordIn(short int Port);
int PortDWordIn(short int Port);
void SetPortBit(short int Port, char Bit);
void ClrPortBit(short int Port, char Bit);
void NotPortBit(short int Port, char Bit);
short int GetPortBit(short int Port, char Bit);
short int RightPortShift(short int Port, short int Val);
short int LeftPortShift(short int Port, short int Val);
short int IsDriverInstalled();
```

## Prototipos en Delphi

```
procedure PortOut(Port : Word; Data : Byte);
procedure PortWordOut(Port : Word; Data : Word);
procedure PortDWordOut(Port : Word; Data : DWord);
function PortIn(Port : Word) : Byte;
function PortWordIn(Port : Word) : Word;
function PortDWordIn(Port : Word) : DWord;
procedure SetPortBit(Port : Word; Bit : Byte);
procedure ClrPortBit(Port : Word; Bit : Byte);
procedure NotPortBit(Port : Word; Bit : Byte);
function GetPortBit(Port : Word; Bit : Byte) : WordBool;
function RightPortShift(Port : Word; Val : WordBool) : WordBool;
function LeftPortShift(Port : Word; Val : WordBool) : WordBool;
function IsDriverInstalled : Boolean;
```

## Prototipos en Visual Basic

```
Private Declare Sub PortOut Lib "IO.DLL" (ByVal Port As Integer, ByVal Data As Byte)
Private Declare Sub PortWordOut Lib "IO.DLL" (ByVal Port As Integer, ByVal Data As Integer)
Private Declare Sub PortDWordOut Lib "IO.DLL" (ByVal Port As Integer, ByVal Data As Long)
Private Declare Function PortIn Lib "IO.DLL" (ByVal Port As Integer) As Byte
Private Declare Function PortWordIn Lib "IO.DLL" (ByVal Port As Integer) As Integer
Private Declare Function PortDWordIn Lib "IO.DLL" (ByVal Port As Integer) As Long
Private Declare Sub SetPortBit Lib "IO.DLL" (ByVal Port As Integer, ByVal Bit As Byte)
Private Declare Sub ClrPortBit Lib "IO.DLL" (ByVal Port As Integer, ByVal Bit As Byte)
Private Declare Sub NotPortBit Lib "IO.DLL" (ByVal Port As Integer, ByVal Bit As Byte)
Private Declare Function GetPortBit Lib "IO.DLL" (ByVal Port As Integer, ByVal Bit As Byte) As Boolean
Private Declare Function RightPortShift Lib "IO.DLL" (ByVal Port As Integer, ByVal Val As Boolean) As Boolean
Private Declare Function LeftPortShift Lib "IO.DLL" (ByVal Port As Integer, ByVal Val As Boolean) As Boolean
Private Declare Function IsDriverInstalled Lib "IO.DLL" As Boolean
```

## Descripción de las Funciones

Por favor, vaya al prototipo adecuado para el lenguaje que esté utilizando.

### **PortOut**

Escribe un byte en el puerto especificado.

**PortWordOut**

Escribe una palabra (16-bits) en el puerto especificado.

**PortDWordOut**

Escribe una palabra doble (32-bits) en el puerto especificado.

**PortIn**

Lee un byte del puerto especificado.

**PortWordIn**

Lee una palabra (16-bits) del puerto especificado.

**PortDWordIn**

Lee una palabra doble (32-bits) del puerto especificado.

**SetPortBit**

Pone el bit del puerto especificado.

**ClrPortBit**

Limpia el bit del puerto especificado.

**NotPortBit**

Niega (invierte) el bit del puerto especificado.

**GetPortBit**

Devuelve el estado del bit especificado.

**RightPortShift**

Desplaza hacia la derecha el puerto especificado. Se devuelve el LSB (bit menos significativo), y el valor pasado se convierte en el MSB (bit más significativo).

**LeftPortShift**

Desplaza hacia la izquierda el puerto especificado. Se devuelve el MSB (bit más significativo), y el valor pasado se convierte en el LSB (bit menos significativo).

**IsDriverInstalled**

Devuelve un valor distinto de cero si io.dll está instalada y funcionando. El objetivo principal de esta función es asegurar que el driver modo kernel NT/200/XP ha sido instalado y está accesible.

**Notas**

- Cuando instale una nueva versión de io.dll, podría ser necesario apagar el driver modo kernel. Para hacer esto, abra una ventana de comandos e introduzca el comando "net stop io.sys".