Desarrollo de un Manejador de Dispositivo Utilizando el Modelo Windows Driver Model (wdm) para Windows 98 y 2000

M. en C. Rubén Peredo Valderrama, Rubén Aquino
CIC-IPN
peredo@cic.ipn.mx

El objetivo de éste trabajo es el desarrollo de un manejador de dispositivo que se encargue del control del puerto paralelo de una computadora personal (PC) bajo el esquema de *Windows Driver Model* (WDM) usado en Windows 98 y Windows 2000. Un manejador de dispositivo (*Device Driver,* DD) proporciona una interfase de software para el hardware conectado a una computadora. Los programas de usuario tienen acceso al hardware de una forma establecida, sin tener que preocuparse de los detalles de control del hardware; esto es, el manejador del dispositivo oculta las particularidades del acceso al mismo.

Un manejador es un programa que normalmente se vuelve parte del kernel del sistema operativo al momento de la carga, y hace disponibles uno o más dispositivos a los programas de usuario, representando una pieza de hardware, lógica o física.

En Windows, un manejador siempre hace que el dispositivo aparezca como un archivo para los programas que quieren utilizarlo. Así, para tener acceso al dispositivo solo se requiere abrir el manejador del mismo, e interactuar con él mediante peticiones de lectura y/o escritura.

El modelo compartido de WDM tiene beneficios significativos, ya que soporta ambas plataformas (w98 y w2000), reduciendo el tiempo de desarrollo para los programadores. Tres son los requerimientos para el desarrollo de DD con WDM: multiprocesador, independiente del procesador y Plug and Play(PnP).

Un driver WDM escrito correctamente requiere únicamente de una recompilación para funcionar en un ambiente Windows NT (*New Tecnology*), manteniendo compatibilidad hacia atrás.

1. INTRODUCCIÓN

El WDM ha tenido diversas modificaciones siguiendo el desarrollo del sistema operativo Windows. Windows 3.1 intentaba dar soporte a una gran cantidad de controladores SCSI diferentes, mientras que NT manejaba miniports; Windows 3.1 fue lanzado sin soporte para miniports SCSI. Se buscaba un modelo compartido para los drivers que comenzó a funcionar con Windows 95, ya que los miniports SCSI y NDIS son compatibles a nivel binario con NT. Uno de los beneficios de un modelo compartido (WDM) es que los programadores de DD pueden reducir el tiempo de desarrollo en un 50%, además de que está diseñado para ser compatible hacia atrás.

El WDM proporciona un marco de device drivers para Windows 98 y 2000. Estos sistemas en apariencia son similares, pero trabajan de manera muy diferente: Windows 98 esta basado en DOS, evolucionando desde Windows 1.0 y hasta Windows 95, con un enfoque hacia el área de computadoras de escritorio. Windows 2000 parte de NT y se destina principalmente a equipos servidores.

La figura 1 muestra la arquitectura de Windows 2000. El software se puede ejecutar en cualquiera de los dos modos; en el modo usuario es

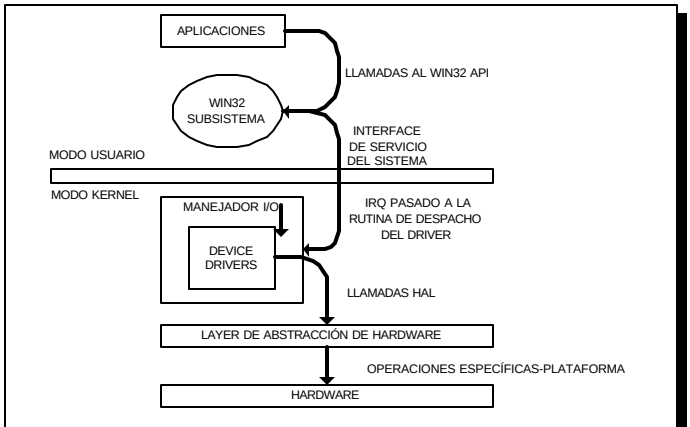

Figura 1. Arquitectura de Windows 2000.

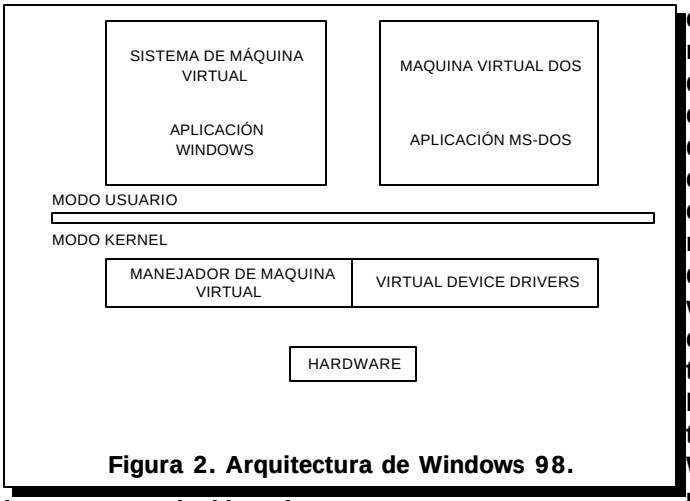

Figura 2. Arquitectura de Windows 98.

era tratar en forma virtual a un dispositivo específico para ayudar a la VMM a crear una simulación, de tal forma que cada una de las máquinas virtuales tenga un complemento total del hardware. La misma arquitectura VMM de Windows 3.0 se utiliza en w98, inseguro y restringido a ciertas actividades, mientras que en modo kernel es totalmente confiable y capaz de realizar cualquier tarea.

La figura 2 muestra la arquitectura de Windows 98. El kernel se denomina Virtual Machine Manager, VMM, porque su trabajo principal es crear una o más máquinas virtuales que comparten el mismo hardware de la máquina física. El propósito original de un manejador virtual de dispositivo (*virtual device driver*, VxD) en Microsoft Windows 3.0 pero con una agrupación para manejar nuevo hardware y aplicaciones de 32-bits.

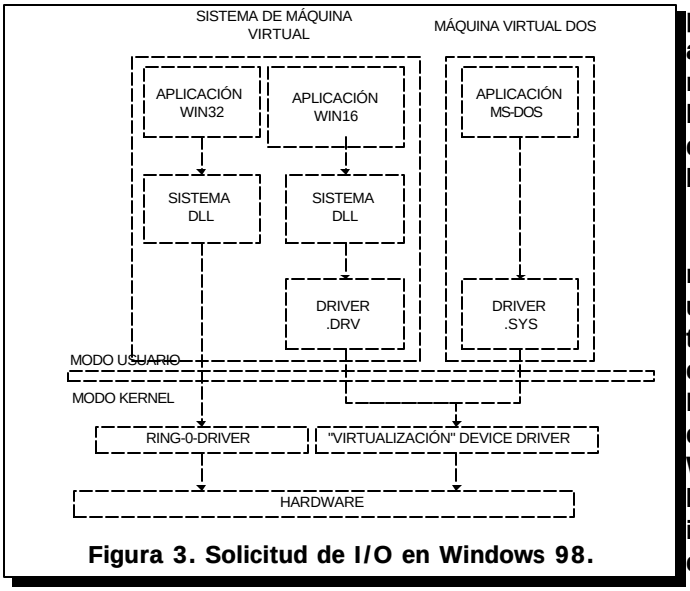

Figura 3. Solicitud de I/O en Windows 98.

La figura 3 muestra como una aplicación tanto WIN32 como WIN16 y DOS realiza solicitudes de I/O en Windows 98. En la columna de la izquierda se indica como una aplicación WIN32 accesa a las solicitudes de I/O por medio de un sistema DLL como KERNEL32.DLL. Pero las aplicaciones pueden únicamente usar ReadFile para leer archivos de disco, puertos de comunicación, y dispositivos que tengan drivers WDM. Para cualquier otra clase de dispositivo, una aplicación debe usar un mecanismo basado en Device IoControl. La columna central corresponde a una aplicación WIN16 ejecutando una solicitud de I/O y en la parte derecha se muestra una aplicación DOS ejecutando una solicitud de I/O.

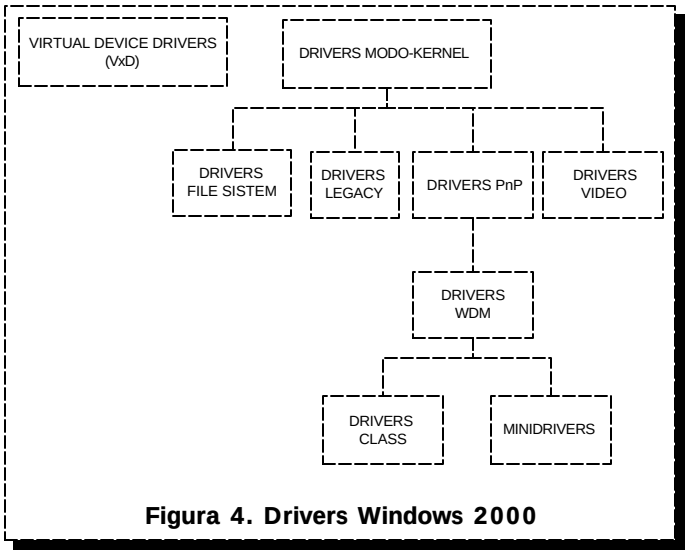

Figura 4. Drivers Windows 2000

La figura 4 muestra los diferentes tipos de drivers que maneja Windows 2000, los cuales son:

- Virtual Device Driver (VDD), es un componente en modo-usuario que permite a las aplicaciones tipo DOS tener acceso al hardware sobre plataformas x86.

- Drivers Modo-Kernel, los cuales incluyen muchas subcategorías. Un driver *Plug and Play* (PnP) es un driver en modo-kernel que maneja PnP.

- Driver WDM, es un driver PnP compatible con Windows 98 y Windows 2000. Dentro de la categoría de los drivers WDM, también se distingue entre los drivers tipo class y los minidrivers, los cuales proveen ayuda a un driver class.

- Video drivers: son drivers en modo-kernel para dispositivos de despliegue e impresoras; su función principal es visualizar datos.

- Drivers del sistema de archivos: implementan el modelo de siste-

ma de archivos estándar para la PC en discos duros o sobre conexiones de red.

- Legacy device drivers: son manejadores en modo-kernel, que directamente controlan un dispositivo del hardware sin ayuda de otros drivers. Esta categoría incluye esencialmente drivers para las primeras versiones de Windows NT, que están ejecutándose sin cambio en Windows 2000.

2. Desarrollo Del Trabajo y Tecnología Utilizada

2.1 Manejador De Dispositivo WDM

Microsoft proporciona varios manejadores genéricos que realizan tareas comunes. Los manejadores de dispositivos pueden usar las características de estos elementos estándar.

La labor de escribir un manejador, por lo tanto, inicia regularmente con la identificación de qué manejador genérico se puede usar. Un manejador de bus de bajo nivel puede usarse para todas las comunicaciones básicas con el hardware. Un manejador de clase intermedia podría proporcionar las características que son comunes a una categoría completa de dispositivos.

2.2 El Modelo De Manejador De Windows (WDM)

El WDM tiene dos aspectos separados pero igualmente importantes. Primero, este modelo describe la estructura estándar para manejadores de dispositivos; segundo, Microsoft proporciona una serie de manejadores de bus y clase para dispositivos comunes.

El modelo WDM describe cómo se instalan y configuran inicialmente los dispositivos y cómo deben servir a las peticiones de usuario e interactuar con el hardware. Un manejador WDM debe encajar en el sistema Plug and Play (PnP) que permite a los usuarios conectar dispositivos que pueden configurarse por software.

2.2 Compatibilidad De Código y Binaria

Originalmente, Microsoft estableció que los manejadores WDM serían compatibles a nivel binario entre Windows 98 y Windows 2000 x86, y compatibles en código para Windows 2000 en plataformas Alfa. Sin embargo, la compatibilidad de binarios no está garantizada.

Es probable que se tengan problemas cuando no se usan las herramientas para el mismo sistema operativo; esto es, es mejor usar el DDK para Windows 98 para manejadores en Windows 98 y el DDK W2000 para W2000.

2.3 Manejadores Estilo WDM Vs. NT

La principal diferencia en el código del dispositivo de estos dos estilos radica en la forma en que se reconocen los dispositivos.

En un driver WDM, el Administrador Plug and Play avisa cuando se agrega o quita un dispositivo del sistema. El administrador Plug and Play usa archivos de instalación INF para encontrar el manejador correcto para el nuevo dispositivo. En contraste, un manejador estilo NT tiene que encontrar sus propios dispositivos, regularmente en su rutina de inicialización. Este tipo de manejadores usualmente se instalan usando un programa especial.

Los nuevos manejadores de bus y clase sólo están disponibles para los manejadores WDM. Los manejadores WDM nuevos y los de NT deben soportar las características de administración de energía y Administración de Windows.

2.4 Componentes De Un Manejador De Dispositivo

Estas son algunas de las tareas que puede hacer un manejador de dispositivo:

- Configurarse inicialmente
- Crear y borrar dispositivos
- Procesar solicitudes Win32 para abrir y cerrar un manejador de archivo.
- Procesar solicitudes Win32 de Entrada/Salida
- Serializar el acceso al hardware
- Comunicarse con el hardware
- Invocar a otros manejadores
- Cancelar solicitudes de E/S
- Limitar (time-out) peticiones de E/S
- Responder si un dispositivo se agrega o remueve durante ejecución
- Manejar las solicitudes de administración de energía.

2.5 Entrada/Salida Básica

En Win32, el acceso a un dispositivo es similar al de un archivo, por lo que se usan las mismas funciones, tales como CreateFile, ReadFile, WriteFile, etc.

2.6 Ambiente

Cualquier número de hebras (*Thread*) Win32 podrían tener acceso a un dispositivo en forma simultánea, así que el dispositivo debe contemplar este aspecto y ser capaz de manejarlo. Esto involucra también bloquear el acceso al dispositivo para

que sólo una de estas hebras pueda operar en forma exclusiva el dispositivo en un tiempo determinado. En este sentido, ayuda el administrador de I/O del kernel.

Puntos de Entrada del Dispositivo y Callbacks

El kernel ejecuta regularmente el código del manejador de dispositivo, enviando IRP (I/O Request Packets). La estructura IRP es fundamental para los manejadores.

Un manejador tiene un punto de entrada principal, una rutina que debe llamarse DriverEntry, la cual tiene un prototipo de función estándar. Cuando se carga el driver, el kernel llama a la rutina DriverEntry, y después puede invocar a otras rutinas, denominadas *callbacks*. Los nuevos manejadores también pueden proporcionar una interfase Modelo de Objeto Común (Common Object Model, COM) para el kernel, mediante una serie de rutinas que implementa el manejador.

2.7 Rutinas De Despacho (Dispatch)

La rutina DriverEntry de un manejador debe establecer o configurar una serie de callbacks para procesar IRP. Además, debe establecer las rutinas Unload, AddDevice y StartIo, si se necesitan.

En la **tabla 1** se muestran las funciones de dispositivo de I/O comunes en Win32 y sus IRP correpondientes.

Los manejadores (*handlers*) para las IRP de la **tabla 1** se llaman comúnmente rutinas de despacho (*dispatch routines*), porque con frecuencia solo realizan algún proceso inicial del IRP, como por ejemplo, validar los parámetros.

2.8 Manejadores

Los manejadores pueden generarse en versiones "free" (*release*) o "checked" (*debug*). La versión free debe ser la versión final, es decir, la que ya no tiene símbolos de depuración. La versión checked es la versión de depuración y que no está optimizada, e incluye símbolos de depuración.

El Driver Development Kit (DDK) de Microsoft genera ambos ambientes para la compilación y el enlace, como se muestra en la **figura 5**.

2.9 Lenguaje y Bibliotecas

Un manejador es una Biblioteca de Enlace Dinámico (*Dynamic Link Library*, DLL) con la extensión .sys. Comúnmente está escrito en C o C++ y puede incluir recursos tales como un bloque de versión, mensajes de eventos y definiciones de clase de Instrumentación de Administración de Windows. En Windows 98, el binario de un manejador debe tener un nombre de 8.3, es decir de 8 caracteres como máximo y la extensión .sys.

No debe usarse la palabra clave new de C++, ya que ésta puede estar implementada usando malloc, que no está disponible para los dispositivos en modo kernel.

2.10 Driver Development Kit

Microsoft recomienda usar el Driver Development Kit (DDK) [3] para el desarrollo de manejadores de dispositivos. Esta es una herramienta que contiene documentación acerca de la elaboración de éste tipo de programas, así como algunas herramientas muy útiles para la compilación y el enlace de los programas; es útil también porque maneja algunas características que facilitan la generación del manejador. El DDK trabaja conjuntamente con Visual C++ y es por ello que para su instalación se requiere que dicho programa esté previamente instalado.

En general, existen versiones de DDK para cada una de las versiones de Windows que Microsoft ha liberado. Es recomendable usar el DDK que corresponda a la versión de Windows que se esté utilizando; la razón es que algunas características del kernel han sufrido cambios importantes de ver-

Tabla 1. Función Win32 IRP

CreateFile	Create
CloseHandle	Close IRP
ReadFile, etc.	Read IRP
WriteFile, etc.	Write IRP
DeviceIoControl IOCTL IRP	IOCTL IRP interna

Figura 5. DDK Instalado.

sión a versión y, en un momento dado, es posible encontrarse con algunos problemas.

2.11 Algunas Recomendaciones Para El Código

Es recomendable hacer el código tan sencillo como se pueda y documentarlo bien. Se deben tratar todas las advertencias como si fueran errores a corregir. También hay que asegurarse el cálculo de los valores de retorno para todas las funciones del kernel en uso; se debe tener cuidado en la manera de usar los recursos del kernel.

2.12 La Utilería Build

La utilería build de línea de comandos del DDK es la herramienta para construir manejadores. Esta llama a la utilería nmake para construir el manejador usando las configuraciones adecuadas del compilador y del enlazador.

Para esto, debe crearse un archivo SOURCES, un archivo makefile, la estructura de directorio y, opcionalmente, los archivos makefile.inc y dirs.

La utilería build despliega los detalles del proceso y los errores en la salida estándar. Además, genera una lista de los errores en un archivo llamado build.err, las advertencias en build.wrn y una bitácora en build.log.

En la figura 6 se muestra el ambiente que genera el DDK para la construcción de los manejadores.

2.13 Archivos Makefile

La utilería nmake usa el archivo makefile, en el que se encuentran las instrucciones que determinan qué comandos ejecutar para actualizar un proyecto. Por ejemplo, la tabla 2 muestra que si el programa prueba.cpp se actualiza, entonces se compila en el archivo prueba.obj usando el compilador cc. prueba.obj se enlaza para crear prueba.exe usando el programa link.

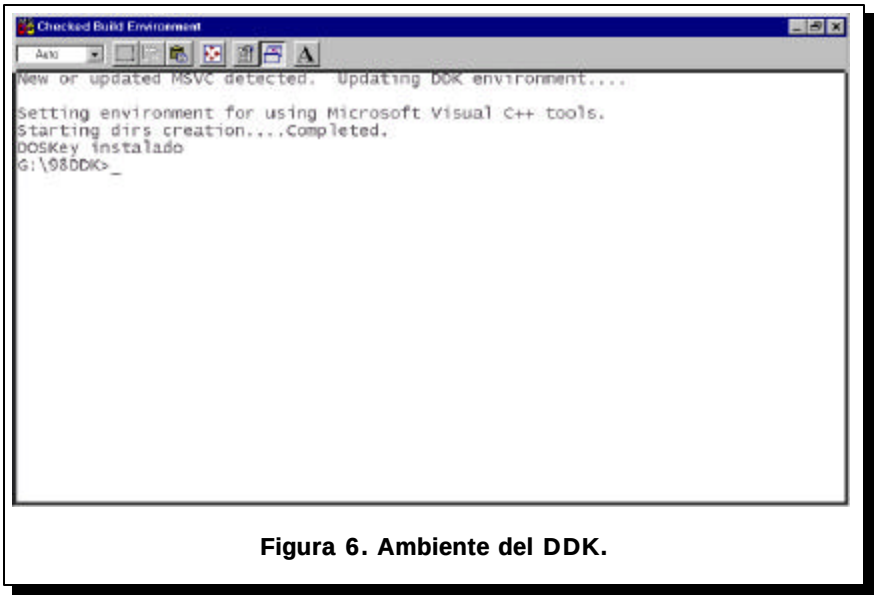

Figura 6. Ambiente del DDK.

Tabla 2. Archivo makefile	
prueba.exe:	prueba.obj link -o prueba.exe prueba.obj
prueba.obj:	prueba.cpp cc prueba.cpp

2.14 Archivo Sources (fuentes)

La utilería build busca un archivo de macroinstrucciones llamado SOURCES en el directorio actual, en el que se especifica qué es lo que se va a construir.

Por ejemplo, para el caso de nuestro manejador, el archivo SOURCES contiene lo siguiente:

```
TARGETNAME=pparalel
  TARGETTYPE=DRIVER
DRIVERTYPE=WDM
TARGETPATH=OBJ
  INCLUDES=$(BASEDIR)\inc;
SOURCES=     Init.cpp \
             Dispatch.cpp \
             Pnp.cpp \
             DeviceIo.cpp \
             DebugPrint.c \
             pparalel.rc
NTTARGETFILES=PostBuildSteps
```

Esto le indica a la utilería build que debe construirse un manejador WDM de nombre pparalel, que se pondrá en el directorio OBJ. Se incluye la ruta del directorio inc del DDK para que busque también archivos de cabecera que puedan requerirse. Luego se listan los archivos que deberán compilarse. NTTARGETFILES especifica algunos pasos posteriores a la construcción del manejador.

2.15 Los Directorios

En Windows 98 y en Windows NT, la herramienta build siempre pone los archivos objeto en el directorio OBJ\i386 para arquitecturas x86

La macro TARGETPATH del archivo SOURCES especifica dónde se alojará el archivo ejecutable final. Si se especifica OBJ, entonces el ejecutable estará en los directorios OBJ\i386\free y OBJ\i386\checked, según sea el caso.

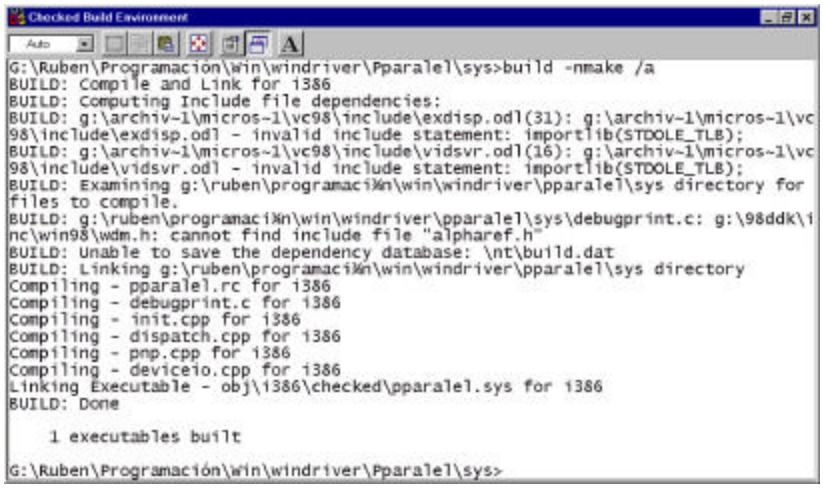

Figura 7. Proceso de compilación del manejador *pparalel* con build.

En Windows 2000, build mantiene separados los archivos objeto free **y** checked**. Si la variable** TARGETPATH **es** OBJ**, los archivos objeto de la versión free x86 y el manejador final estarán en el directorio**OBJ-FREE\i386 **y en** OBJCHK\i386 **para el caso de la versión** checked**.**

En la figura 7 se muestra el proceso de compilación de nuestro manejador *pparalel*, usando la herramienta build.

2.16 Archivos Inf

Un archivo INF contiene toda la información necesaria para instalar un manejador de dispositivo WDM. Esto incluye la lista de los archivos a copiar, las entradas que deben aparecer en el registro de Windows, etc.

Windows proporciona un instalador estándar para la mayoría de las clases de dispositivos y un instalador por omisión, el cual procesa archivos INF.

Un archivo INF es un archivo de texto que se parece a los archivos INI. Tiene secciones, cada una de las cuales inicia con una línea que contiene su nombre encerrado entre paréntesis cuadrados, y luego el contenido de la sección. Cada línea tiene una entrada simple o asigna un valor. Las secciones pueden estar en cualquier orden; a veces puede ser importante el orden de las líneas dentro de una misma sección. Los comentarios se escriben después del punto y coma.

Secciones Estándar

Versión

Regularmente se encuentra al inicio del archivo INF y contiene información sobre la versión del manejador.

Strings

Esta sección define cadenas que se sustituyen en alguna otra parte del archivo.

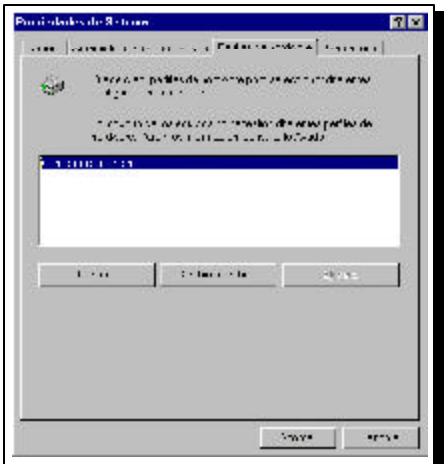

Figura 8. Perfiles de hardware del sistema.

2.17 Instalación Del Manejador

Para instalar el manejador creado, es recomendable copiar el perfil de hardware que se está usando para tener un respaldo y poder realizar modificaciones (ver figuras 8 y 9).

Para modificar la copia del perfil de hardware, al que hemos llamado PTO-Paralelo, es necesario reiniciar la máquina e iniciar con éste perfil de hardware. Al reiniciar, Windows encontrará dos perfiles de hardware y solicitará que se elija con cual debe iniciar, en este caso PTO-Paralelo. Después, en el Panel de Control, debe eliminarse el manejador que ya está controlando dicho puerto, como se muestra en las figuras 10 y 11.

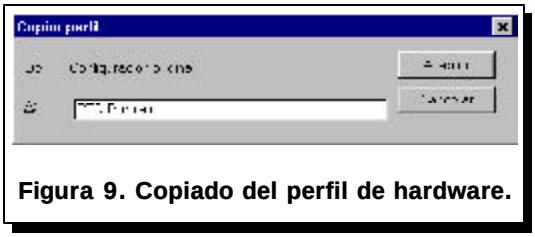

Figura 9. Copiado del perfil de hardware.

En este momento ya puede instalarse el manejador elaborado;para ello es necesario usar la opción Agregar nuevo hardware **del Panel de Control (figura 12).**

Ahí debe indicársele a Windows que el dispositivo y el controlador serán indicados por el usuario, para evitar que Windows los busque. Windows indicará que falta un controlador para el puerto paralelo y mostrará dicho dispositivo. Debe indicársele que no instale al dispositivo, pues de lo contrario, utilizará su manejador

por omisión, como se muestra en las figuras 13 y 14.

Al seleccionar el dispositivo, debe indicarse que se trata de Otros Dispositivos, como se muestra en la figura 15.

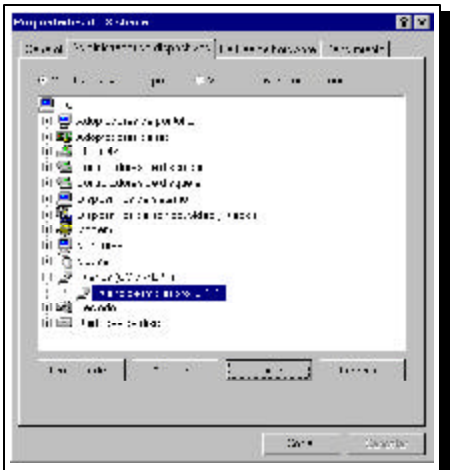

Figura 10. Administrador de dispositivos del sistema.

En el siguiente cuadro de diálogo, figura 16, hay que elegir que se use un disco y entonces se elige el directorio en el que se encuentre el archivo *.inf correspondiente. Este archivo es el que contiene información importante para que Windows sepa de qué tipo de manejador se trata, el nombre, la ubicación, etc.

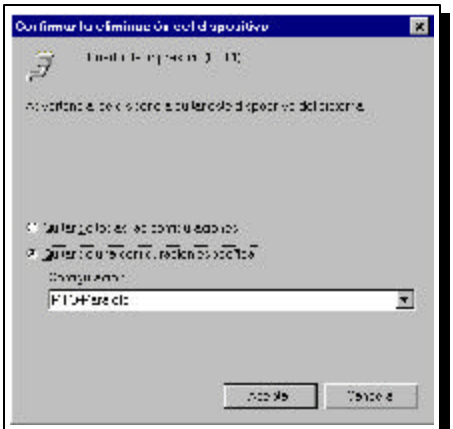

Figura 11. Eliminación del manejador actual del puerto paralelo del sistema.

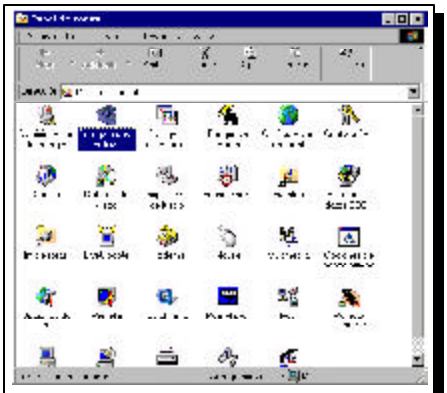

Figura 12. Panel de control, agregar nuevo hardware.

Aparecerá entonces, un cuadro de diálogo con los manejadores para los cuales se encontraron archivos .inf. Se elige entonces la versión checked del manejador CIC-IPN (figura 17), mostrándose los recursos que el manejador usará (figura 18).

Una vez hecho esto, el manejador ya está instalado y es necesario arrancar nuevamente el sistema para que los cambios tomen efecto.

Figura 13. Asistente: Agregar nuevo hardware.

Figura 14. Asistente: Agregar nuevo hardware-seleccionar.

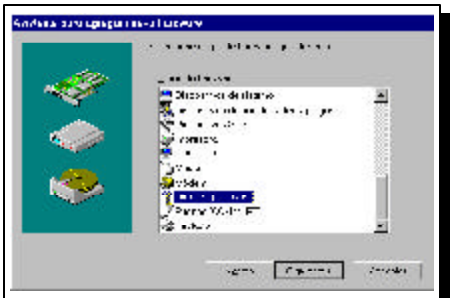

Figura 15. Asistente: Agregar nuevo hardware-otros dispositivos.

Figura 16. Asistente: Agregar nuevo hardware-usar disco.

Podemos observar al reiniciar, que en el Panel de Control, en la parte de Sistema y en la de Administrador de Dispositivos, se encuentra ya instalado el nuevo manejador (figura19).

2.18 Prueba Del Manejador Para Puerto Paralelo Bajo Ambiente Dos En Windows 98

En este caso, se elaboró un programa sencillo que sólo envía datos al puerto paralelo y puede observarse la salida en una serie de LED conectados a él (figura 20). El código del programa se muestra en la tabla 3.

Figura 17. Selección del manejador CIC-IPN para puerto paralelo.

Figura 18. Recursos asignados por el SO para el manejador CIC-IPN.

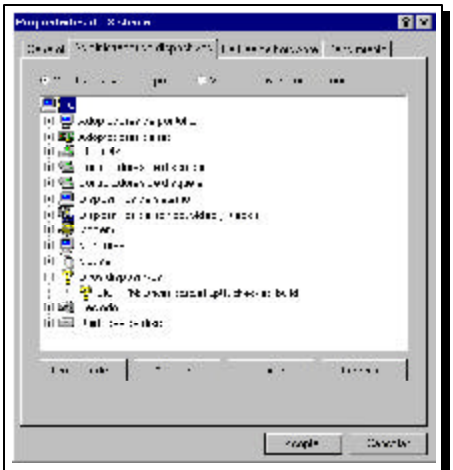

Figura 19. Administrador de dispositivos: manejador para el puerto paralelo CIC-IPN.

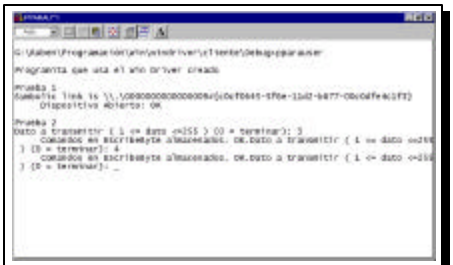

Figura 20. Prueba del nuevo manejador del puerto paralelo (ventana DOS).

2.19 Prueba Del Manejador Para Puerto Paralelo Bajo Ambiente Windows

Tabla 3. Código de comunicación con el manejador

```
do
{
 printf("Dato a transmitir ( 1 <= dato <=255 ) (0 =
        terminar): ");
 scanf("%d",&dato);
 while((dato<0 || dato>255) && dato!=-1)
  {
    printf("Dato no válido. Introduzca
nuevamente el
           número: ");
    scanf("%d",&dato);
  }
 EnviarByte[2]=dato;
 if(DeviceIoControl(hWdmIo,
IOCTL_PHDIO_RUN_
   CMDS,EnviarByte,length(EnviarByte),&num,
1,
   &BytesReturned, NULL))
  {
    printf("Comandos en EscribeByte
almacenados. OK.");
  }
  else{printf("XXX  falló el almacenamiento de
comandos
       EscribeByte %d\n",GetLastError());
}
```

Este programa es similar al anterior, ya que envía datos al puerto paralelo y se observa la salida en una serie de LED. Es importante mencionar que se elaboro el programa utilizando C++Builder de Borland. La ejecución del programa se muestra en la figura 21 y el listado del mismo se incluye en la tabla 4.

3. Sumario

Este trabajo desarrolló un manejador de dispositivo que se encarga del control del puerto paralelo de una PC bajo el nuevo esquema de Windows Driver Model(WDM) usado en Windows 98 y Windows 2000.

El modelo compartido de WDM tiene beneficios significativos, ya que soporta ambas plataformas reduciendo el tiempo de desarrollo para los desarrolladores de Device Driver(DD). Tres son los requerimientos para el desarrollo de un DD WDM: multiprocesador, debe ser independiente del procesador y PnP; pero un driver WDM escrito correctamente requiere únicamente una recompilación para funcionar en un ambiente NT 64-bits, ya que WDM mantiene la compatibilidad hacia atrás.

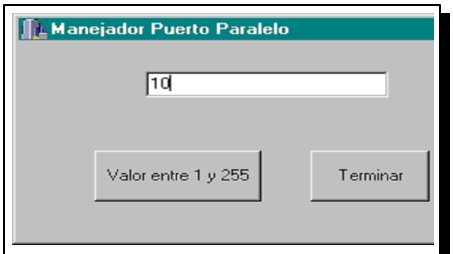

Figura 21. Prueba del nuevo manejador del puerto paralelo (Windows).

Tabla 4. Código de comunicación con el manejador

```
// ABRE EL PUERTO
hComm = CreateFile("LPT1",
GENERIC_READ | GENERIC_WRITE,
0,
0,
OPEN_EXISTING,
0,
0);
```

Se utilizo DDK de Microsoft para el desarrollo de este manejador, además de que fué necesario instalar Visual C++. Así mismo, se utilizaron varias herramientas adicionales del DDK, como son:

- Build
- Makefile
- Inf
- Sources

Una vez terminado el manejador fue necesario darlo de alta en el SO, y posteriormente se realizaron dos programas sencillos para pruebas del manejador, uno en DOS y otro bajo ambiente Windows.

El proceso de elaboración de un manejador WDM es más complicado que utilizar los puertos en un progra-

ma en lenguaje C. Pero los beneficios son que un pequeño programa en C en los diferentes Windows utilizando el puerto paralelo no funciona, mientras que el utilizar DDK y el nuevo modelo WDM funciona adecuadamente en Windows y manteniendo un control de la tarjeta vía el puerto paralelo.

Actualmente, el desarrollo de estos manejadores en el área de ingeniería de hardware es de gran importancia, ya que los diseñadores de tarjetas requieren una interfaz ya no con DOS sino con Windows. Finalmente, el presente trabajo muestra el desarrollo de un manejador para el puerto paralelo, mostrando todos los pasos y herramientas necesarias para un buen desarrollo y pruebas necesarias para su buen funcionamiento.

4. BIBLIOGRAFÍA

[1] *"Writing Windows WDM Device Drivers"*, **Chris Cant, R&D Books.**

[2] *"Programming the Microsoft Windows Driver Model"*, **Walter Oney, Microsoft Press.**

5. REFERENCIAS

[1] **http://msdn.microsoft.com**

[2] **http://msdn.microsoft.com/msdnmag**

[3] **http://www.microsoft.com/ddk**