Programación de controladores de dispositivos en Windows

Pablo Gutiérrez Peón
Grupo de Computadores y Tiempo Real
Universidad de Cantabria

4 de noviembre de 2013

Resumen

Este documento pretende servir como toma de contacto en la construcción de controladores de dispositivos para el sistema operativo Windows.

Se repasa brevemente el entorno Windows para continuar con los componentes básicos en la realización de controladores. Para finalizar se da una pequeña guía con los pasos a seguir al desarrollar controladores de dispositivos.

La información contenida en este documento procede del siguiente manual: *Art Baker y Jerry Lozano. The Windows 2000 Device Driver Book* [1].

Índice

1. Introducción a los controladores Windows NT

La familia Windows NT se compone hasta la fecha de los sistemas operativos Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 y Windows 8.

La arquitectura Windows NT se basa en dos modos de operación:

- Modo usuario: Los programas y subsistemas están limitados a los recursos del sistema a los que éste permite el acceso.
- Modo núcleo (kernel): Tiene acceso total a la memoria del sistema y los dispositivos externos. La arquitectura dentro del modo núcleo se compone de los siguientes elementos:
  - Núcleo híbrido.
  - Capa de abstracción del hardware (HAL).
  - Controladores (drivers).
  - Ejecutivo (Executive) sobre el cual son implementados todos los servicios de alto nivel.

Componentes del Ejecutivo

El Ejecutivo provee los servicios básicos del Sistema Operativo (SO) Windows NT. Entre sus componentes se encuentran:

- **System Service Interface.** Es el punto de entrada de modo usuario a modo kernel. Permite al código de usuario invocar servicios del SO.
- **Object Manager.** Prácticamente todos los servicios del SO se modelan como objetos: hilos, eventos, secciones de memoria, entradas del Registro, etc. El Object Manager permite crear/destruir objetos del SO.
- **Configuration Manager.** Modela el hardware y software instalado. La base de datos del Registro se emplea para almacenar este modelo. Los controladores de dispositivos emplean la información recogida en el Registro para descubrir muchos aspectos del entorno donde son ejecutados. Desde la aparición de Plug & Play (PnP), el rol del Registro se ha visto reducido considerablemente.
- **Process Manager.** Maneja los procesos; el entorno donde se ejecutan los hilos manteniendo un espacio de memoria privado.
- **Virtual Memory Manager.** Gestiona la memoria.

- **Local Procedure Call Facility.** Las llamadas entre procesos se realizan entre espacios de memoria diferentes. Este componente del Ejecutivo se encarga de realizar estas llamadas.
- **I/O Manager.** Componente del Ejecutivo que presenta una abstracción uniforme al modo usuario de las operaciones de entrada/salida. Desde el punto de vista del controlador, las peticiones del usuario se presentan como IRP (I/O Request Packet). Estos IRPs constituyen la comunicación entre el código de usuario y el controlador.
- **Active Directory Service.** Forma unificada de identificar recursos de sistema (discos, impresoras, nombres de ficheros, etc.). Está organizado de forma jerárquica.
- **Extensiones al sistema operativo base.** Los servicios del núcleo del SO no están directamente expuestos al usuario. En su lugar se definen diferentes APIs que el código de modo usuario toma como abstracciones del SO:
  - Subsistema Win32. API nativa de Windows NT. Se encarga de las interfaces gráficas de usuario (GUIs), entrada/salida por consola e implementa la API Win32 mediante la cual las aplicaciones y otros subsistemas interaccionan con el Ejecutivo.
  - POSIX. Da soporte a aplicaciones estilo Unix. Lamentablemente un portado directo de aplicaciones Unix no suele funcionar correctamente por lo que hay que escribir en Win32.
  - Otros subsistemas. Virtual DOS Machine (VDM), Windows on Windows (WOW), OS/2.

Tipos de controladores

- Controladores de modo usuario (user-mode drivers): Para dispositivos simulados o virtualizados.
- Controladores de modo núcleo (kernel-mode drivers):
  - Controladores legados (legacy drivers).
  - Controladores Windows Driver Model (WDM) - PnP.

A su vez, los controladores legados y WDM pueden ser de uno de los siguientes tres tipos:

- *High level*: Realizan una abstracción sobre los controladores de nivel inferior. Ej: controlador sobre el sistema de ficheros.

- *Intermediate*: Se insertan entre los high level y low level. Pueden ser mini-controladores que proveen de un mismo servicio a varios controladores de bajo nivel agrupando el contenido común. También se puede tratar de filtros.
- *Low level*: Controladores para los buses hardware.

Por otro lado, citar que existen arquitecturas especiales en Windows NT para controladores de tipo similar:

- Controladores de vídeo.
- Controladores para impresora.
- Controladores multimedia.
- Controladores de red.

2. Aspectos básicos del entorno hardware

Registros de dispositivo

Los controladores se comunican con un periférico leyendo y escribiendo en los registros asociados al dispositivo. Las funciones de estos registros se reparten entre las siguientes:

- **Command:** Los bits de estos registros son los que controlan el dispositivo de algún modo.
- **Status:** Registros típicamente leídos por el controlador para descubrir el estado actual del dispositivo.
- **Data:** Registros empleados para transferir datos entre el dispositivo y el controlador.

Para acceder a los registros del dispositivo se precisa conocer la dirección del primer registro del dispositivo y el espacio de direcciones donde los registros se encuentran.

Normalmente, los registros se encuentran en direcciones consecutivas y el primero se emplea como base para acceder al resto.

Los registros se pueden mapear de dos formas, bien en el espacio de entrada/salida en cuyo caso se denominan *puertos* o bien en la propia memoria.

Existen unas macros que permiten acceder a estos registros Cuadro 1.

Cuadro 1: Macros para acceso a puertos/registros.

Función	Descripción
`READ_PORT_XXX`	Lee un valor de un puerto de E/S
`WRITE_PORT_XXX`	Escribe un valor de un puerto de E/S
`READ_REGISTER_XXX`	Lee un valor de un registro de E/S
`WRITE_REGISTER_XXX`	Escribe un valor de un registro de E/S

Interrupciones de dispositivo

Las interrupciones permiten al dispositivo requerir la atención de la CPU de forma asíncrona. Se asigna al menos una línea que el dispositivo puede activar para comunicar la interrupción. Es después responsabilidad de la CPU salvar el contexto en el que se encuentre antes de saltar a la rutina de tratamiento de interrupción suministrada por el controlador del dispositivo.

Las interrupciones son un mecanismo muy útil para mejorar el rendimiento del sistema. Los dispositivos que no emplean interrupciones pueden causar considerables degradaciones en el rendimiento al emplear ciclos de CPU que se podrían estar evitando.

Mecanismos de transferencia de datos

Para transferir datos desde la CPU o memoria a un dispositivo o viceversa se pueden emplear las siguientes técnicas:

- **E/S programada:** Transferencia de datos directamente sobre los registros del dispositivo. El código del controlador debe utilizar una instrucción de E/S para leer o escribir. Este mecanismo de transferencia está restringido a la transferencia de pequeñas cantidades de datos sobre dispositivos lentos.
- **Acceso directo a memoria (DMA):** Mecanismo de transferencia de datos entre el dispositivo y la memoria sin la intervención directa de la CPU. El encargado de desplazar los datos de una localización a otra es el controlador DMA, que libera de esta carga de trabajo a la CPU.
- **Buffer compartido:** El dispositivo puede contar con una región de memoria que compartir y mapear dentro del espacio de memoria de la CPU.

Auto-reconocimiento y auto-configuración del dispositivo

Cada dispositivo consume una serie de recursos del sistema. Estos recursos consisten en bien un rango de direcciones de E/S o memoria, interrupciones o canales de DMA.

Dado que cada dispositivo puede requerir de distintos recursos, es casi inevitable un conflicto a la hora de su asignación.

Los primeros PCs requerían que el usuario configurara cada dispositivo mediante *jumpers*, interruptores, etc. de forma que a cada uno de ellos le fuera asignado un recurso de forma única.

Las nuevas arquitecturas de bus han introducido la tecnología que permite que un dispositivo se auto-reconozca con el objetivo de que puedan reportar su presencia al sistema y se auto-configure de forma software para la asignación de recursos.

Un dispositivo debe identificarse y dar una lista de recursos a consumir:

- Identificador del fabricante.
- Identificador del tipo de dispositivo.
- Requisitos de E/S.
- Requisitos de interrupciones.
- Requisitos de canal DMA.
- Requisitos de memoria de dispositivo.

El dispositivo debe, en conjunción con el bus donde se localice, generar una señal de notificación cada vez que es insertado o retirado.

Buses y Windows NT

Un bus es una colección de líneas de datos, direcciones y control que permite a los dispositivos comunicarse.

Windows NT da soporte a múltiples buses, entre los que se encuentran ISA, PCI, USB o Firewire.

3. Procesado de E/S en modo kernel

Contextos de ejecución en modo kernel

El contexto describe el estado del sistema cuando una instrucción de la CPU está en ejecución. Incluye el contenido de todos los registros de la CPU y el modo de procesador entre otros datos.

El código a ejecutar debe ser consciente del contexto donde se ejecuta. En Windows NT existen tres contextos de ejecución posibles:

- **Contexto para traps o excepciones:** Cuando ha ocurrido una trap o excepción.
- **Contexto para interrupciones:** Cuando se recibe una interrupción.
- **Contexto para hilos de modo kernel:** Cuando una porción de código corre en un hilo del kernel.

Llamada a procedimientos diferidos (DPCs)

Cuando una porción de código del kernel se ejecuta a una prioridad elevada, cualquier otro código de menor prioridad ve imposibilitada su ejecución. Si demasiado código se ejecuta a elevada prioridad, el rendimiento del sistema puede verse perjudicado de forma considerable. El uso de DPCs permite que si una rutina de tratamiento de interrupción se va a extender durante un amplio periodo de tiempo, este trabajo se ejecute a una menor prioridad.

Acceso a buffers de usuario

Cuando un hilo de modo usuario hace una petición de E/S, suele pasar la dirección del buffer de datos localizado en el espacio de usuario en donde se copiarán o de donde se leerán los datos.

Varios problemas pueden surgir para acceder a este buffer desde el modo kernel. Desde posibles cambios en la tabla de páginas a que directamente el buffer esté en zona de intercambio, fuera de la memoria RAM e imposibilitando el acceso.

Para solucionar estos problemas, el I/O Manager permite dos métodos de acceso a los buffers de usuario. Cuando el controlador de dispositivo se inicializa, es necesario que le comunique al I/O Manager qué estrategia se quiere seguir de entre las siguientes:

- **E/S mediante buffer:** El I/O Manager copia el buffer de usuario por completo en la memoria RAM dedicada del sistema. El dispositivo puede hacer uso de esta copia y tras terminar su trabajo, el I/O Manager copia de vuelta estos datos en la memoria del usuario. Esta técnica se emplea con dispositivos generalmente lentos y que no requieren de transferir grandes cantidades de datos.
- **E/S directa (DMA):** Esta técnica evita la copia del buffer al permitir el acceso directo del dispositivo al buffer de usuario. El I/O Manager bloquea este buffer para evitar que se mueva de la RAM y provoque fallos de página. De esta forma se puede acceder a estos datos de forma segura. La técnica se emplea en dispositivos rápidos que transiferen grandes bloques de datos.

Estructura de un controlador de modo kernel

Un controlador se compone de una colección de rutinas que son llamadas por el sistema operativo (normalmente por el I/O Manager). Dependiendo del controlador, el I/O Manager puede llamar a las rutinas del controlador en situaciones como la carga y descarga del controlador, cuando un dispositivo es insertado o retirado, cuando el usuario hace peticiones de E/S, etc.

Rutinas de inicialización y limpieza del controlador

Cuando un controlador es cargado en el sistema, varias acciones deben llevarse a cabo. Lo mismo ocurre cuando el controlador va a ser retirado.

- **Rutina `DriverEntry`:** Ejecutada cuando el controlador va a ser cargado en el sistema. Los controladores deben poder ser cargados dinámicamente en cualquier momento, no solo durante la primera localización de controladores. Aquí se realizan tareas como localizar el hardware a controlar, reservar los recursos hardware que se van a utilizar (puertos, interrupciones, DMA) y dar un nombre visible al dispositivo dentro del sistema para que pueda ser accedido. Nótese que para controladores PnP, la reserva de hardware se deja a la función AddDevice.
- **Rutina Unload:** Llamada cuando el controlador va a ser eliminado. Debe deshacer cualquier acción que hubiese llevado a cabo la rutina `DriverEntry`.
- **Otras rutinas de esta familia:** Reinitialize, Shutdown y Bugcheck Callback.

Rutinas para tratamiento de peticiones de E/S

Cuando el I/O Manager recibe una petición de una aplicación de modo usuario se llama a una de las rutinas que aparecen en esta sección.

- **Rutinas Open y Close:** Todos los controladores deben contar con una rutina de CreateDispatch para manejar la petición Win32 `CreateFile`. Lo mismo ocurre con la rutina CloseDispatch para manejar `CloseHandle`.
- **Operaciones de dispositivo:** Dependiendo del dispositivo, el controlador puede tener rutinas para tratar las transferencias de datos y control de operación. Estas son `ReadFile`, `WriteFile` y `DeviceIoControl` para que el usuario lea datos, escriba datos o configure el dispositivo respectivamente.

Rutinas de transferencia de datos

Grupo de rutinas que se emplean para tratar transferencias de datos.

- **Rutina Start I/O:** Llamada cuando un dispositivo debe comenzar la transferencia de datos.
- **Rutina de Tratamiento de Interrupción (ISR):** Se accede a estas rutinas cada vez que el dispositivo genera una interrupción. Como ya se ha comentado, la ISR debe durar lo mínimo posible y cualquier trabajo intensivo debe dejarse a una DPC.
- **Rutina para DPC:** Rutinas que deben aparecer siempre que el mecanismo DPC se considere necesario.

Llamadas de sincronización de recursos

Los controladores pueden contar con múltiples hilos que pueden querer hacer uso de recursos comunes de forma simultánea.

A pesar de que en modo usuario es común bloquear un hilo en espera de un recurso, al programar en modo kernel esto no está permitido. El mecanismo empleado consiste en dar la dirección de una rutina que se utiliza para sincronizar el acceso a un recurso. Cuando el recurso se hace disponible, se invoca dicha rutina. Existen tres tipos de rutinas de sincronización: rutina ControllerControl, rutina AdapterControl y rutina SynchCritSection.

4. Objetos de modo kernel para los controladores

I/O Request Packets (IRPs)

Las transacciones en Windows NT se modelan como paquetes, los denominados I/O Request Packets (IRPs).

Con cada petición de E/S del usuario, el I/O Manager crea un IRP en la memoria del sistema. Basándose en el manejador de dispositivo y la operación pedida por el usuario, el I/O Manager pasa el IRP a la rutina de usuario apropiada. Cuando la operación del IRP se completa, el controlador debe indicar en el IRP un código de estado que será devuelto al usuario.

Un IRP cuenta con dos partes:

- Cabecera: Contiene información general de estado. Algunas partes de la cabecera son accesibles por el usuario, mientras que a otras solo tiene acceso el I/O Manager. En la cabecera es donde se indica el estado del IRP procesado y donde se encuentran los campos que permiten el acceso del controlador a buffers de datos.
- Pila: La pila está compuesta de uno o más bloques con parámetros. Estos bloques se utilizan para navegar por las distintas capas de un controlador en el caso de que el IRP deba transmitirse de unas a otras.

Existe un conjunto de funciones que permiten operar sobre el conjunto del IRP, para por ejemplo marcarlo como completado o hacer que éste salte a otro nivel del controlador para que lo atienda.

Objetos de Controlador (Driver Objects)

`DriverEntry` es la única rutina del controlador con un nombre preestablecido y obligatorio. Cuando el I/O Manager necesita localizar otras funciones del controlador, emplea el Objeto de Controlador asociado a un dispositivo específico. Por ello este objeto es básicamente un catálogo que contiene punteros a distintas funciones del controlador.

Objetos de Dispositivo (Device Objects)

Los Objetos de Dispositivo mantienen información sobre las características y estado de un dispositivo. Esto permite al I/O Manager y al controlador saber y manejar el estado del dispositivo en cada momento, por lo que tienen una importancia capital.

Los Objetos de Dispositivo se crean en la rutina `DriverEntry` (AddDevice en PnP) y se destruyen en la rutina Unload.

Tal y como ocurría con los IRPs, esta estructura cuenta con muchos campos que son de acceso exclusivo del I/O Manager. Se dispone de un conjunto de funciones para acceder a este objeto.

Extensiones de Dispositivo (Device Extensions)

Conectada con el Objeto de Dispositivo se encuentra otra importante estructura de datos, la Extensión de Dispositivo. La Extensión es un bloque de memoria que el I/O Manager adjunta a cualquier Objeto de Dispositivo creado. Es el autor del controlador el que especifica tanto el tamaño como contenido de la Extensión de Dispositivo. Se emplea para mantener cualquier información asociada con el dispositivo en particular. Por ejemplo, el nombre que se le ha dado, las direcciones base de los puertos y memoria, etc.

Otros objetos de modo kernel

- **Objeto de Controlador (Controller Object) y Extensión de Controlador (Controller Extension):** Algunos adaptadores de periféricos gestionan más de un dispositivo físico utilizando el mismo conjunto de registros. Surge un problema de sincronización cuando el controlador intenta ejecutar operaciones simultáneas sobre más de uno de los dispositivos conectados. El Objeto de Controlador hace las veces de mutex para solucionar este problema. A su vez, su Extensión puede almacenar datos específicos.
- **Objeto Adaptador (Adapter Object):** Empleado para sincronizar los recursos de DMA.
- **Objeto de Interrupción (Interrupt Object):** Permite al gestor de interrupciones del kernel encontrar la correcta rutina de tratamiento cuando una interrupción sucede.

5. Rutinas de inicialización y limpieza

Escribir una rutina `DriverEntry`

Cualquier controlador en Windows NT debe contar con una rutina de nombre `DriverEntry`. Esta rutina inicializa varias estructuras de datos del controlador y prepara el entorno para otros componentes del mismo. El I/O Manager llama a esta rutina cuando carga el controlador.

Los pasos que realiza `DriverEntry` son los siguientes:

1. `DriverEntry` localiza el hardware que va a controlar. El hardware es asignado marcándose bajo el control del controlador.
2. El Objeto del Controlador es inicializado mediante el anuncio de los otros puntos de entrada del controlador. Estos punteros a las funciones son almacenados directamente en el Objeto del Controlador. Sólo es obligatorio nombrar a la función de entrada como `DriverEntry`. Cualquier otra función puede llevar un nombre definido por el creador del controlador. En este paso se asocia cada función con el nombre dado por el desarrollador.
3. Si el controlador gestiona más de un dispositivo sobre los mismos registros, se crean los Objetos y Extensiones de Controlador.
4. `IOCreateDevice` es usado para crear un Objeto de Dispositivo por cada dispositivo físico o lógico bajo su control. La Extensión de Dispositivo es también iniciada. En este punto se debe escoger también si se quiere hacer la comunicación con el usuario mediante acceso buffereado o directo.
5. El dispositivo creado es hecho visible al subsistema Win32 mediante la llamada a `IoCreateSymbolicLink`.
6. Se conectan las interrupciones si las hubiera.
7. Los pasos 4 a 6 se repiten por cada dispositivo físico o lógico gestionado por el controlador.
8. Se retorna `STATUS_SUCCESS` si todo ha finalizado sin fallos.

Nótese que los pasos 1 y 3-6 no son llevados a cabo por la rutina `DriverEntry` si se trata de un controlador PnP, ya que en ese caso es labor de AddDevice.

Uno de los argumentos de la llamada `IOCreateDevice` es el nombre del dispositivo. Los dispositivos en Windows pueden tener más de un nombre. Sin embargo, de cara al sistema interno el nombre es único y se le da en esta llamada. Este nombre se guarda bajo la sección \Device del Object Manager. Los nombres simbólicos, de los cuales se pueden crear tantos como se quieran se colocan en la sección \??.

Escribir una rutina Unload

El I/O Manager llama esta rutina cuando el controlador va a ser eliminado de forma manual o automática (por ejemplo por un reinicio).

A grandes rasgos, deshace lo que hizo la rutina de `DriverEntry`.

6. Rutinas para tratar peticiones de E/S

El trabajo de un controlador es responder a peticiones de E/S procedentes de aplicaciones de modo usuario u otras partes del sistema. En respuesta a estas peticiones, el I/O Manager llama a las rutinas que aparecen en esta sección.

Antes de que el controlador pueda procesar peticiones de E/S, debe anunciar qué tipo de operaciones soporta. Este anuncio se realiza como en otras ocasiones desde la rutina `DriverEntry`.

Cualquier controlador debe soportar el código de función `IRP_MJ_CREATE` ya que este supone la respuesta a la llamada de Win32 `CreateFile`. Sin esta llamada, no se podría obtener un manejador para el dispositivo. Lo mismo ocurre con `IRP_MJ_CLOSE` para dar soporte a CloseHandle. Los otros códigos de función dependen de la naturaleza del dispositivo a controlar. Entre los disponibles están ReadFile, WriteFile o DeviceIoControl.

Consideraciones al escribir las rutinas de tratamiento de E/S

Todas estas rutinas tienen los mismos parámetros de cabecera. El I/O Manager invoca estas rutinas en respuesta a peticiones procedentes del usuario o del kernel. Antes de proceder a llamarlas, el I/O Manager construye y rellena el IRP con los datos correspondientes, entre los que se encuentra el puntero al buffer de usuario, cuyo acceso está garantizado por el I/O Manager. Recordar que el modo de acceso a esta zona de datos debe indicarse escogiendo entre las siguientes opciones: buffereado o directo.

Acceso al IRP

Si se desea obtener acceso al IRP que originó la llamada, se puede obtener un puntero mediante la llamada `IoGetCurrentIrpStackLocation`. Esto permite, entre otras utilidades, acceder a los datos de entrada o devolver datos de salida. Suele ser aconsejable realizar validaciones sobre los parámetros recibidos, de forma que concuerden con lo esperado en el controlador.

Salida de la rutina

Cuando la rutina procesa un IRP, hay tres posibles salidas:

- Los parámetros de entrada no pasan la validación del controlador y la petición se rechaza.
- La petición puede ser procesada sin intervención del dispositivo. Un ejemplo: leer cero bytes.
- El dispositivo debe ser utilizado para procesar la petición.

Para estas situaciones, en cada caso se realiza una de las siguientes acciones:

- **Señalizar error:** Al detectar un problema con el IRP, éste debe ser rechazado y se debe informar al llamador. Para ello, se le notifica el código de error concreto en el campo `Status` del IRP (existen códigos predefinidos o el usuario puede especificar los suyos propios [2]), se marca el IRP como completado con `IoCompleteRequest` y se devuelve el código de error en la rutina.
- **Completar la petición:** En el caso en que sea necesaria una interacción breve con el dispositivo, ésta se realiza y a continuación se notifica el IRP como en el caso anterior pero indicando un estado exitoso `STATUS_SUCCESS`.
- **Planificar una operación del dispositivo:** Es necesaria una interacción con el dispositivo que va a tomar un tiempo suficientemente largo como para que la espera pudiera bloquear el resto del sistema. En este caso, se debe marcar al IRP como en proceso con `IoMarkIrpPending`. Después, el IRP se encola para que sea tratado por la rutina Start I/O. Finalmente, se sale de la rutina con estado `STATUS_PENDING`.

Procesamiento de peticiones de lectura y escritura

Las peticiones de E/S más básicas son aquellas que permiten intercambiar datos entre un buffer de usuario y un dispositivo.

Los códigos de función correspondientes son `IRP_MJ_READ` y `IRP_MJ_WRITE`.

Como ya se ha tratado previamente, existen dos tipos de acceso al buffer de usuario. El acceso se realiza por medio de un campo en las `Flags` del IRP. Se distinguían dos tipos de acceso:

- **E/S mediante buffer:** La dirección con la que se obtiene acceso a este buffer se encuentra en el campo del IRP `AssociatedIrp.SystemBuffer`.

- **E/S directa:** En este caso se crea una estructura de datos denominada Lista de Descriptores de Memoria (MDL). La dirección de esta estructura se encuentra en el campo `MdlAddress` del IRP. Mediante la función `MmGetSystemAddressForMdl` se obtiene una dirección de sistema del buffer de usuario. A pesar de ser una dirección de kernel, el buffer se encuentra en memoria de usuario, por lo que dicho buffer se bloquea haciéndolo no paginable y por tanto garantizando su acceso.

Extender la interfaz mediante llamadas Ioctl

Las llamadas Ioctl permiten cualquier tipo de operaciones especificas en el controlador sin las restricciones de la abstracción de las llamadas de lectura y escritura.

Existen dos tipos de llamadas Ioctl:

- `IRP_MJ_DEVICE_CONTROL`: La función asociada a este código permite invocaciones procedentes de la llamada Win32 de usuario `DeviceIoControl`.
- `IRP_MJ_INTERNAL_DEVICE_CONTROL`: La rutina asociada a este código permite llamadas sólo desde el modo kernel. Su principal uso es recoger llamadas procedentes de otros controladores.

Estas rutinas cuentan en sus parámetros con un código de control `IoControlCode` que permite discernir entre unas acciones a tomar u otras.

Estos códigos son definidos por el programador de la función Ioctl, pero deben seguir unas reglas respecto al significado de los bits de que se componen. En los 32 bits disponibles se definen los siguientes campos: tipo de dispositivo, acceso requerido, código de control y tipo de transferencia.

Es necesario permitir al usuario el acceso a los códigos definidos sobre estos campos mediante un fichero de cabecera para que pueda hacer uso de ellos.

Al igual que ocurría con las llamadas de lectura y escritura, las llamadas Ioctl también hacen uso de buffers. En todos los casos, la dirección del buffer desde el Ioctl se obtiene del campo del IRP `AssociatedIrp.SystemBuffer`. Existen tres tipos de acceso:

- **METHOD_BUFFERED:** Los datos se manejan en la memoria del kernel y el intercambio con el usuario se hace mediante copia.
- **METHOD_IN_DIRECT:** Buffer de entrada localizado en la memoria del usuario. El buffer se bloquea para permitir el acceso desde el modo kernel.

- **METHOD_OUT_DIRECT:** Buffer de salida localizado en la memoria del usuario. El buffer se bloquea para permitir el acceso desde el modo kernel.

7. E/S por interrupciones

Algunos dispositivos tienen un patrón de funcionamiento irregular que hace que permanezcan ociosos durante amplios periodos de tiempo. Estos dispositivos son uno de los grupos de dispositivos susceptibles de poseer la habilidad de interrumpir al procesador en el momento en que un dato esté disponible, en lo que se conoce como E/S por interrupciones o programada.

Funcionamiento de la E/S programada

En la E/S programada, la CPU transfiere cada unidad de datos desde o hacia el dispositivo en respuesta a una interrupción. La secuencia de eventos es la que sigue:

1. Un paquete IRP (normalmente `IRP_MJ_READ` o `IRP_MJ_WRITE`) determina que es necesaria la interacción con el dispositivo para ser completado. La rutina encola el IRP para que sea atendido por la rutina Start I/O.

2. La rutina Start I/O inicializa el dispositivo, normalmente escribiendo o leyendo un dato.

3. Finalmente, el dispositivo genera una interrupción que el kernel pasa a la Rutina de Tratamiento de Interrupción (ISR) del controlador.

4. Si es necesario transmitir más datos, la ISR comienza una nueva transferencia. Los pasos 3 y 4 se repiten hasta que se termine la transferencia completa de los datos.

5. En este momento, la ISR encola una petición para lanzar la rutina del controlador DpcForIsr. Como se comentó en capítulos anteriores, las rutinas DPC corren a un nivel de prioridad inferior, pero los datos ya están transmitidos y sólo queda finalizar la IRP por lo que no es necesaria mayor prioridad.

6. Finalmente se ejecuta la rutina DpcForIsr, que marca el IRP como completado, informando al I/O Manager de que un nuevo IRP puede ser procesado.

Cambios para adaptar las rutinas de inicialización y limpieza

Es necesario llevar a cabo inicialización extra durante la rutina `DriverEntry`/AddDevice, exportando la rutina Start I/O, inicializando la rutina DpcForIsr mediante la llamada `IoInitializeDpcRequest` y conectando con la interrupción asociada al dispositivo mediante `IoConnectInterrupt`.

De forma similar, la rutina de Unload debe liberar los nuevos recursos empleados.

Escribir una rutina Start I/O

El I/O Manager invoca a la rutina del controlador Start I/O cuando se llama a `IoStartPacket` o `IoStartNextPacket`.

Sus responsabilidades son las siguientes:

1. Llamar a `IoGetCurrentStackLocation` para obtener un puntero a la pila IRP.
2. Si el dispositivo soporta más de una función `IRP_MJ_XXX`, examinar en la pila IRP el campo `MajorFunction` para determinar la operación actual.
3. Hacer copias del puntero al buffer del sistema y su longitud en la estructura de Extensión de Dispositivo.
4. Marcar en la Extensión de Dispositivo que se espera una interrupción.
5. Empezar la operación con el dispositivo.

Escribir una rutina de tratamiento de interrupción (ISR)

Una vez el dispositivo está operativo, la transferencia de datos viene de la llegada de interrupciones hardware.

La ISR debe entonces seguir estos pasos:

1. Determinar si la interrupción pertenece al controlador. En caso negativo, devolver `FALSE`.
2. Llevar a cabo las operaciones necesarias en el dispositivo para dar por recibida la interrupción y que deje de mantenerla.
3. Determinar si quedan por transmitir más datos, en cuyo caso se debe planificar la siguiente transferencia que desembocará seguramente en una nueva interrupción.

4. Si todos los datos han sido transmitidos o ha ocurrido un error, encolar una petición DPC mediante `IoRequestDpc`.

5. Retornar `TRUE`.

Cualquier trabajo que no sea absolutamente esencial debe ser transferido a una rutina DPC, ya que es imprescindible que los manejadores de interrupciones se mantengan ligeros.

Escribir una rutina DpcForIsr

La rutina del controlador DpcForIsr es responsable de determinar el estado final de la petición actual, completando el IRP o comenzando uno nuevo.

La rutina DpcForIsr se planifica tras invocarse `IoRequestDpc`.

Dado que la mayor parte del trabajo ya se ha realizado durante el procesamiento de la interrupción, la rutina DpcForIsr no tiene mucho que hacer:

1. Escribir los campos del IRP `Status` e `Information` que indican el estado del IRP y en número de bytes transmitidos respectivamente.

2. Llamar a `IoCompleteRequest` para completar el IRP.

3. Llamar a `IoStartNextPacket` para enviar el siguiente IRP a la rutina Start I/O.

8. Inicialización del hardware

Un controlador moderno no debe presuponer que los recursos a emplear vayan a estar localizados en una dirección de memoria concreta o en una línea de interrupción fija. Tampoco debe obligar al usuario instalador de un dispositivo a configurar jumpers, interruptores o cualquier otro elemento de estas características.

Introducción a la arquitectura Plug and Play (PnP)

Con la llegada de la tecnología Plug and Play (PnP) se consigue dar soporte a la instalación y desinstalación de dispositivos en el sistema sin las dificultades derivadas de los problemas ya citados cuando se tiene que configurar el hardware.

Objetivos de Plug and Play

Se resumen en los siguientes puntos:

- Detección automática del hardware instalado o retirado.
- Los dispositivos deben ser configurables mediante software. Ya no son necesarios switches y jumpers.
- Los controladores necesarios para el nuevo hardware deben ser automáticamente instalados tras ser requeridos por el sistema operativo.
- Si el bus de conexión del dispositivo lo permite, se debe soportar la conexión en caliente del dispositivo.

Componentes de Plug and Play

Los sistemas operativos Windows NT implementan PnP con varios componentes software:

- **Gestor Plug and Play:** Consta de dos partes, una trabaja en modo kernel y otra en modo usuario. El modo kernel interactúa con el hardware y otros componentes del kernel para gestionar la correcta detección y configuración del hardware. El modo usuario interacciona con los componentes de la interfaz de usuario que permiten consultar y alterar la configuración del software PnP instalado.
- **Gestor de energía:** Gestiona el manejo de la energía a los dispositivos. Dependiendo de su naturaleza, es posible temporalmente eliminar la corriente de dispositivos que no están siendo utilizados.
- **Registro:** El Registro de Windows mantiene una base de datos del hardware y software instalado para los dispositivos PnP. Permite a los controladores y a otros componentes identificar y localizar los recursos empleados por un dispositivo.
- **Ficheros INF:** Cada dispositivo debe ser descrito por completo mediante un fichero que será empleado para la instalación del controlador del dispositivo.
- **Controladores Plug and Play:** Los controladores para dispositivos PnP pueden formar parte de dos categorías: WDM y NT. Los controladores NT PnP son controladores legados que emplean ciertas características de la arquitectura PnP pero no soportan por completo el modelo WDM. Por ejemplo, se aprovechan de PnP para obtener información de configuración pero no soportan los mensajes IRP de PnP.

Los controladores WDM son, por definición, totalmente compatibles con PnP.

Detección de dispositivos en PnP

El modelo WDM es una extensión del modelo de controladores NT visto hasta el momento. Por ello, `DriverEntry` sigue sirviendo como punto de entrada, pero sus responsabilidades se ven reducidas. Únicamente se limita al anuncio de otros puntos de entrada del controlador.

Una nueva función AddDevice se encarga de las tareas antes asignadas a `DriverEntry` como son la creación del objeto de dispositivo.

Las capas de un controlador en PnP

El modelo de controladores WDM se basa en capas estructuradas de Objetos Físicos de Dispositivo (PDOs) y Objetos Funcionales de Dispositivo (FDOs):

- **PDO:** Generalmente existe un PDO por cada pieza física del hardware que se encuentra unida al bus. El PDO asume la responsabilidad de control de bajo nivel del dispositivo.
- **FDO:** Existen por cada función lógica o abstracta presente en el software de alto nivel.

El proceso de carga de controladores de Windows NT es el siguiente:

1. Durante la instalación del SO, se enumeran todos los buses del Registro de sistema y su topología.
2. Durante el arranque (boot), un controlador para cada bus es cargado.
3. Una de las primeras responsabilidades de un controlador de bus es enumerar todos los dispositivos conectados. Se crea un PDO para cada dispositivo encontrado.
4. Por cada dispositivo descubierto, se obtiene la clase de dispositivo que permite obtener su controlador FDO.
5. Si el FDO no está aún cargado se invoca la `DriverEntry`.
6. Se invoca AddDevice para cada FDO. Tras crear en esta rutina el dispositivo, se invoca `IoAttachDeviceToDeviceStack` para apilarlo.

Nuevos IRPs en WDM

Existen una serie de IRPs propios de los controladores WDM que son enviados por el gestor PnP cuando eventos tales como inicialización o parada de dispositivo ocurren [3].

Se debe por tanto crear un manejador para gestionar los IRPs. Algunos son obligatorios en función del dispositivo y otros son opcionales.

En el Cuadro 2 aparecen los IRPs que deben ser soportados por todos los dispositivos.

Cuadro 2: Códigos de los mensajes IRP de Plug and Play y su significado.

Código PnP IRP	**Significado**
`IRP_MN_START_DEVICE`	(Re)Inicializa el dispositivo.
`IRP_MN_QUERY_STOP_DEVICE`	¿Puede ser detenido el dispositivo para una posible reasignación de recursos?.
`IRP_MN_STOP_DEVICE`	Detiene el dispositivo. Posible reinicio o retirada del dispositivo.
`IRP_MN_CANCEL_STOP_DEVICE`	Notifica que el QUERY_STOP no será llevado a cabo.
`IRP_MN_QUERY_REMOVE_DEVICE`	¿Puede ser retirado el dispositivo de forma segura?
`IRP_MN_REMOVE_DEVICE`	Deshace el trabajo de AddDevice.
`IRP_MN_CANCEL_REMOVE_DEVICE`	Notifica que la QUERY_REMOVE es desechada.
`IRP_MN_SURPRISE_REMOVAL`	Notifica que el dispositivo ha sido retirado sin previo aviso.

El PnP Manager comunica el IRP al controlador de más alto nivel en la pila de dispositivos. Es común que el controlador funcional (FDO) deje al controlador físico (PDO) la implementación de muchas peticiones PnP. Para legar estos IRPs a niveles inferiores de la pila se usa la combinación de `IoCopyCurrentStackLocationToNext` y `IoCallDriver`.

Enumeración de dispositivos

El Gestor de Configuración PnP es el encargado de enumerar el hardware descubierto en el sistema y asignarle los recursos.

Cuando el controlador recibe el IRP `IRP_MN_START_DEVICE` enviado por el PnP Manager tras descubrir el dispositivo, un campo del IRP enumera la lista de recursos asignados al dispositivo: `Parameters.StartDevice-.AllocatedResourcesTranslated`. Es en este punto cuando se debe anotar cualquier dato necesario para acceder a los recursos asignados que se deseen

utilizar. Por ejemplo, anotar la dirección base y longitud de un puerto o el canal y puerto para DMA en la Extensión de Dispositivo.

9. Instalación del controlador

La instalación automática del controlador se lleva a cabo mediante un fichero de texto con la extensión INF. Este fichero permite una instalación automatizada basada en diálogo.

La instalación provoca los siguientes cambios en el sistema:

- Se crean nuevas entradas en el Registro que describen el controlador, el modo de carga y cualquier otro dato de configuración.
- Los ficheros del controlador son copiados a su correspondiente directorio de sistema.

Estructura y campos del fichero INF

Los ficheros INF son ficheros de texto divididos en secciones donde cada sección está indicada mediante un identificador entre corchetes []. Algunos de los nombres de secciones son requeridos, mientras que otros se especifican según las necesidades del controlador.

Las secciones cuentan en su interior con entradas. Las entradas tienen el siguiente formato:

```
entry = value [, value...]
```

en donde `entry` es la directiva y `value` el atributo aplicado a la `entry`.

A continuación se enumeran algunas de las secciones más relevantes:

- `Version`: Actúa como cabecera y firma del fichero INF.
- `Manufacturers`: Cada entrada lista los dispositivos y controladores que son instalados por el fichero INF.
- `Models`: Presenta descripciones de los dispositivos junto con el identificador PnP devuelto por el hardware durante su anuncio en un bus PnP compatible. Este identificador es el que permite asociar dispositivos y controladores. La estructura del identificador puede variar dependiendo del hardware, pero normalmente tiene la siguiente forma:
  `<enumerator>\<enumerator-specific-device-ID>`
  Ejemplo: `PCI\VEN_1000&DEV_0001&SUBSYS_00000000&REV_02`

- `DDInstall`: Agrupa en su interior otras secciones como `CopyFiles` y `AddReg`, que se encargan de copiar ficheros y añadir entradas del Registro respectivamente.
- `DDInstall.Services`: Notifica al Gestor de Control de Servicios (SCM) la existencia del controlador.
- `DestinationDirs`: Especifica el directorio donde instalar los ficheros del controlador.

DDK incluye una herramienta llamada CHKINF que permite detectar muchos de los errores presentes en los ficheros INF con el objeto de corregir y validar su correcta sintaxis.

Utilización del fichero INF en la instalación

Se puede llevar a cabo una instalación manual del controlador haciendo click con el botón derecho del ratón y seleccionando Instalar. Sin embargo, este método a veces no es soportado por el sistema. No obstante, lo más habitual es llevar a cabo la instalación automática, que se activa cuando se inserta un dispositivo.

Los pasos de la instalación son los que siguen:

1. Cuando un dispositivo es insertado, el hardware alerta al bus de la presencia del nuevo dispositivo.
2. El PnP Manager notifica al usuario de la presencia de un nuevo dispositivo.
3. El PnP Manager construye una lista de controladores posibles para el nuevo dispositivo. El directorio INF del sistema es rastreado en busca de la clase y modelo que correspondan con el dispositivo.
4. Si no se encuentra un fichero INF adecuado, se pregunta al usuario por la localización del controlador.
5. Una vez se encuentra un fichero INF adecuado, se lleva a cabo la instalación de acuerdo a las instrucciones contenidas en el fichero.
6. Se instalan todos los controladores necesarios. Por último, se notifica al controlador de nivel superior una IRP de tipo `IRM_MN_START_DEVICE`.

Firmado digital del controlador

Microsoft permite someter a los controladores a un proceso de firmado digital que verifica su autoría con el objetivo de no instalar software fraudulento o que pueda comprometer la estabilidad del sistema [4].

En los sistemas operativos de 64 bits el firmado de controladores es obligatorio.

10. Creación de un primer controlador de prueba

Preparación del entorno

- Instalar Windows Driver Kit (WDK) [5].
  - Incluye las herramientas y documentación necesarias para desarrollar controladores.
  - Contiene los ficheros de cabecera .h con las definiciones de tipos y cabeceras que componen la API de desarrollo de controladores. Existen dos versiones:
    - Windows Driver Kit 8: Desarrollo de controladores para Windows 8, Windows 7 y Windows Vista. Integrado en el IDE Visual Studio. Provee las herramientas para desarrollar, construir, empaquetar, probar y depurar controladores [6].
    - Windows Driver Kit 7.1.0: Desarrollo de controladores para Windows 7, Windows Vista y Windows XP. No está integrado en Visual Studio, por lo que es necesario el uso de herramientas complementarias para escribir, probar y depurar el controlador.
- Instalar DebugView [7]: Permite monitorizar la salida de depuración de un sistema local o remoto vía TCP/IP. Muestra el resultado de llamadas a la API de depuración como `DbgPrint` (una especie de `printf` de los controladores). Permite filtrar los mensajes según procedan del modo kernel o Win32.
- Para la edición de los ficheros del controlador no es necesario nada más que un editor de ficheros de texto plano. No obstante, se aconseja contar con un editor que resalte el código escrito en C/C++.

Desarrollo

1. Lo primero es crear una carpeta para alojar el futuro controlador. Por comodidad y dado que se deben hacer referencias relativas a los di-

rectorios de WDK, se puede crear la carpeta en el directorio \src de WDK.

2. Después, comienza el trabajo más complejo: codificar el controlador ideado. Uno o varios ficheros .c y .h compondrán el código basado en las ideas expuestas hasta este punto.

3. A continuación, se crea el fichero Sources:

```
TARGETNAME=<nombre controlador>
TARGETTYPE=DRIVER

TARGETPATH=.
INCLUDES= $(BASEDIR)\inc;.

SOURCES=<lista de ficheros fuente separados por espacios>
```

4. Creación del Makefile (cuidado de no incluir tabulaciones al comienzo de la línea):

```
!INCLUDE $(NTMAKEENV)\makefile.def
```

Compilación

El controlador debe ser generado con un compilador especial de WDK. El compilador se encuentra en Inicio → Windows Driver Kits → WDK → Build Enviroments. En este punto, se debe escoger el sistema operativo objetivo y su versión (32 o 64 bits). Si se desea hacer un controlador para distintas versiones de Windows, se debe compilar una versión para cada uno de ellos y dar las alternativas correspondientes en el fichero INF.

A su vez, se puede optar por una compilación *Checked* (depuración activada) o *Free* (depuración no activada).

Una vez con la herramienta de compilación arrancada, acudir al directorio donde se encuentren los fuentes del controlador y ejecutar `build`. Si no hay errores, se obtendrá un fichero `.sys` con el controlador ya compilado.

Instalación

Existen varias formas de instalar un controlador. Como ya se ha comentado anteriormente, si el controlador es PnP la instalación se hace en base a un fichero INF de instalación que se encarga de realizar todo el trabajo. En el caso de que no se disponga de este fichero, una instalación manual puede ser realizada en algunos casos.

Para ello, se debe copiar el fichero .sys del controlador en el directorio del sistema Windows\System32\drivers.

A continuación se debe registrar el controlador como un servicio del sistema. Podemos crear nuestro propio cargador/eliminador de servicios empleando las funciones Win32 `CreateService` y `DeleteService`. Sino, se puede utilizar alguna herramienta que ya lo haga por nosotros, como el OSR Driver Loader [8].

Prueba

1. Arrancar la herramienta DebugView en modo administrador.
   - Configurarla para captura en modo kernel.
2. Lanzar el controlador:
   - Mediante un programa propio al estilo del cargador de instalación (de nuevo se puede usar el OSR Driver Loader).
   - O acudiendo al administrador de dispositivos.
     - Si el controlador no aparece:
       - Es probable que sea porque no está recogido en ninguno de los tipos indicados en la lista.
       - En el menú superior *Ver* seleccionar los distintos tipos de dispositivos hasta que aparezca el controlador en la lista.
       - A veces hay que reiniciar el sistema varias veces hasta que aparezca.
     - Doble clic sobre el controlador.
     - Pulsar los botones Start y Stop para lanzar y parar el controlador.
       - Cada botón ejecuta una de las funciones que se definieron al programar el driver Start (`CreateDevice`) y Stop (`DriverUnload`).
     - Ver el resultado en DebugView si se incluyeron mensajes de depuración.

Referencias

[1] Jerry Lozano Art Baker. *The Windows 2000 Device Driver Book: A Guide For Programmers*. Prentice Hall, Inc., Upper Saddle River, New Jersey, 2001.

[2] Win32 Error Codes. `http://msdn.microsoft.com/en-us/library/cc231199.aspx`. Consultado el: 17-04-2013.

[3] Plug and Play Minor IRPs. `http://msdn.microsoft.com/en-us/library/windows/hardware/ff558807%28v=vs.85%29.aspx`. Consultado el: 17-04-2013.

[4] Kernel-Mode Code Signing Walkthrough. `http://msdn.microsoft.com/en-us/windows/hardware/gg487328`. Consultado el: 17-04-2013.

[5] Windows Driver Kit (WDK). `http://msdn.microsoft.com/es-es/library/windows/hardware/gg487428.aspx`. Consultado el: 17-04-2013.

[6] Writing your first driver. `http://msdn.microsoft.com/es-es/library/windows/hardware/ff554811%28v=vs.85%29.aspx`. Consultado el: 17-04-2013.

[7] Windows Sysinternals DebugView. `http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx`. Consultado el: 17-04-2013.

[8] OSR Driver Loader. `http://www.osronline.com/article.cfm?article=157`. Consultado el: 17-04-2013.