Enseñanza de programación multihilo y controladores de dispositivo en entornos Windows para alumnos de electrónica

A. Da Silva, V. Hernández y J.F. Martínez
Departamento de Ingeniería y Arquitecturas Telemáticas. Escuela Universitaria de Ingeniería Técnica de Telecomunicación. Universidad Politécnica de Madrid. España.
{adasilva, vhernandez, jfmartin}@diatel.upm.es

Este artículo describe la experiencia y el entorno desarrollado para la enseñanza de aplicaciones multihilo y controladores de dispositivo en entornos Windows. La docencia de estos temas suele quedar fuera de los programas académicos de las titulaciones de electrónica por los conocimientos previos de sistemas operativos necesarios para abordar su estudio y la probabilidad de desconfiguración del entorno de laboratorio al ejecutar código de las prácticas en modo núcleo. Sin embargo es muy importante para los estudiantes de las titulaciones de electrónica ya que introduce un factor adicional a tener en cuenta en todo diseño electrónico cuyo destino sea formar parte de un sistema basado en el uso de la plataforma PC como hardware de proceso. Se propone un entorno de prácticas completo que cubre todos los niveles de desarrollo, desde la lógica de aplicación hasta el controlador de dispositivo sin el uso de hardware real, lo que facilita la libertad de horarios del alumno al poder realizar la práctica en su ordenador personal.

1. Introducción

En los actuales sistemas de control complejos es completamente habitual el uso de un sistema operativo. Actualmente el uso de plataformas PC ejecutando alguna de las versiones de sistemas operativos Windows es una opción para el desarrollo de sistemas de control para sistemas empotrados complejos. Esta circunstancia facilita la tarea del programador ya que le permite centrarse en la lógica de aplicación y despreocuparse de temas como la gestión de memoria e interrupciones, además de disponer de soporte para la programación multihilo y otras facilidades. Sin embargo, al mismo tiempo, el sistema operativo limita e incluso prohíbe el acceso directo por parte de las aplicaciones al hardware. Los elementos de la interfaz hardware/software que se encargan de trasladar las peticiones de entrada/salida al hardware son los controladores de dispositivo que cumplen una doble misión:

- Ocultar al programador de aplicaciones las particularidades del hardware.
- Ofrecer una interfaz estándar independiente del dispositivo.

Es el fabricante quien debe proporcionar el controlador de su dispositivo si desea que pueda ser usado con los sistemas operativos más populares. Es evidente que programar un controlador requiere un conocimiento total del hardware que se desea controlar. Por ello cualquier desarrollo de una placa de expansión para PCI, debe ir acompañado del correspondiente controlador de dispositivo de forma que el hardware funcione con los sistemas operativos más comunes, en otro caso no podrá ser usado.

2. Estructura software de un controlador de dispositivo

Tal como se muestra en la figura 1, un controlador de dispositivo es un modulo software que se ejecuta en modo núcleo, esto le proporciona un acceso total a todos los recursos del sistema. Para cada una de las operaciones genéricas que la aplicación de control puede invocar: apertura, lectura, escritura y control, el controlador proporciona una rutina que la realiza. Dicha rutina será la acceda al hardware consiguiéndose los dos objetivos antes mencionados: ofrecer una interfaz uniforme y única al programador de aplicaciones y ocultar los detalles concretos del hardware.

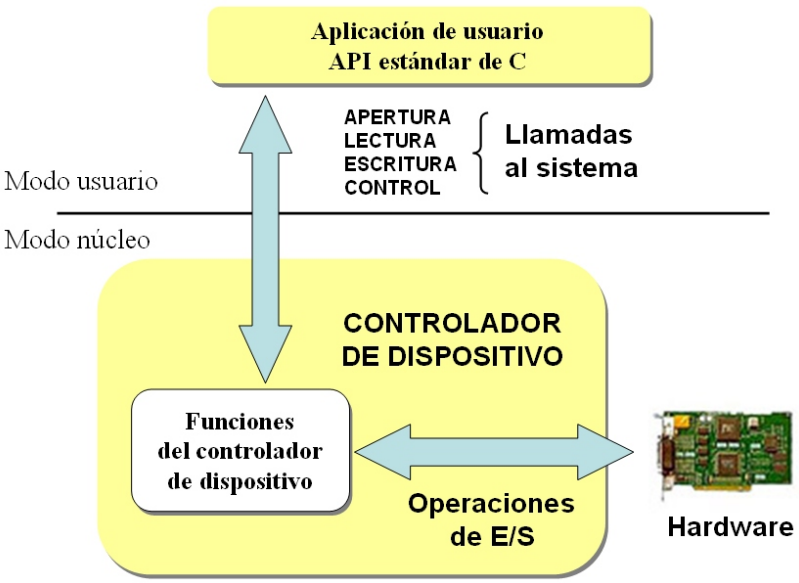

Figura 1. Estructura general

Tal como se puede leer en [1], desde el punto de la programación, un controlador de dispositivo es un conjunto de rutinas que responden a las peticiones genéricas realizadas por una aplicación de control ejecutada en modo usuario.

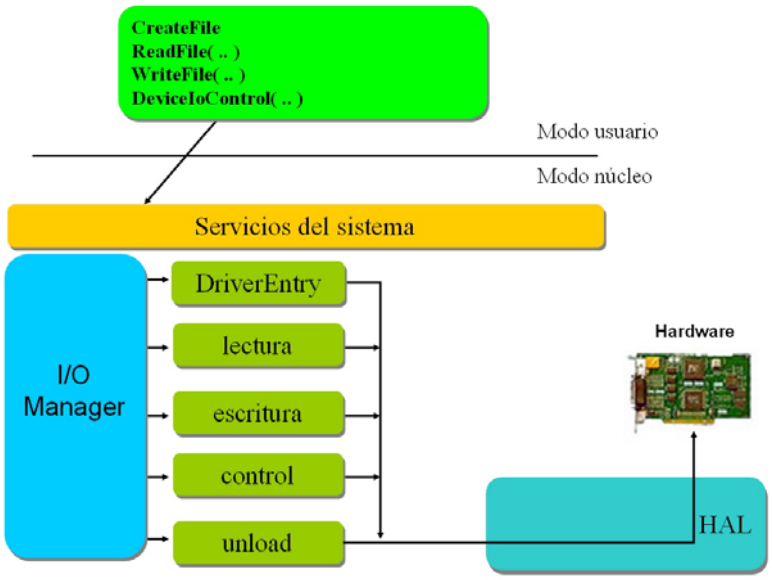

Figura 2. Estructura software de un controlador en entornos Windows

Además de estas rutinas de procesamiento existen otras de gran importancia para la correcta configuración del sistema. Concretamente la rutina ***DriverEntry*** es invocada por el sistema operativo durante su arranque y realiza dos tareas fundamentales:

- Da nombre al dispositivo para que este quede perfectamente identificado en el sistema y pueda ser usado por parte de una aplicación de control mediante la correspondiente operación de apertura.
- Asocia a cada petición genérica de E/S: lectura, escritura y control, una rutina de procesamiento.

A continuación se muestra de forma esquemática el código de configuración de las operaciones que está presente en ***DriverEntry*** y los prototipos de las rutinas de procesamiento y descarga del controlador.

```
/*
  Funciones de procesamiento del dispositivo
*/

MajorFunction[IRP_MJ_READ]           = lectura;
MajorFunction[IRP_MJ_WRITE]          = escritura;
MajorFunction[IRP_MJ_DEVICE_CONTROL] = control;

/* Función de descarga */

DriverObject->DriverUnload = unload;
```

El programador del controlador de dispositivo debe proporcionar el código de las rutinas de procesamiento. La rutina de descarga se invoca al descargar el controlador del sistema y debe encargarse de eliminar todos los recursos extraordinarios que hayan sido solicitados durante el arranque y configuración, así como de borrar el nombre del dispositivo del sistema para que no pueda ser referenciado.

3. Entorno desarrollado. Hardware Virtual

El desarrollo de un controlador no es una tarea sencilla aunque la funcionalidad del sistema sea trivial. El estilo de programación, la codificación incremental y prueba de cualquier pequeña modificación es imprescindible. La depuración es compleja y el uso de técnicas habituales en el desarrollo de aplicaciones como la ejecución paso a paso o el uso de puntos de ruptura se ve dificultada por la ejecución de código en modo núcleo. Si la estructura misma de un controlador es compleja y la programación en modo núcleo delicada para un profesional es de imaginar el impacto negativo que puede causar un código incorrecto ejecutado modo kernel. Un error de programación en modo usuario causa daños en el entorno de la aplicación que generalmente solo afectan a esta, pero ese mismo error en modo núcleo puede corromper el sistema operativo, pudiendo ser necesario, no solo rearrancar la maquina, sino incluso proceder a una reinstalación completa del sistema operativo. Si además de todo lo expuesto, es necesario enfrentarse a un hardware real con toda su problemática, el objetivo docente de configurar una práctica que puede servir de introducción a la programación de controladores de dispositivo puede ser ardua y no existe abundancia de descripciones de experiencias similares. En [2] se presenta una experiencia previa mediante el uso de un hardware virtual diferente y en [3] se presenta una experiencia similar para sistemas de tiempo real en entornos VxWorks y Linux.

Como escenario de aplicación, ver figura 3, se propone el desarrollo de una aplicación que gestione adecuadamente la temperatura de 4 zonas diferentes de un edificio. En cada zona hay una sistema de adquisición que permite la medida de la temperatura, un elemento ventilador y un elemento calefactor. Todos esos elementos están conectados a un sistema informático con sistema operativo Windows que tiene conectado el hardware necesario para la realización del control propuesto.

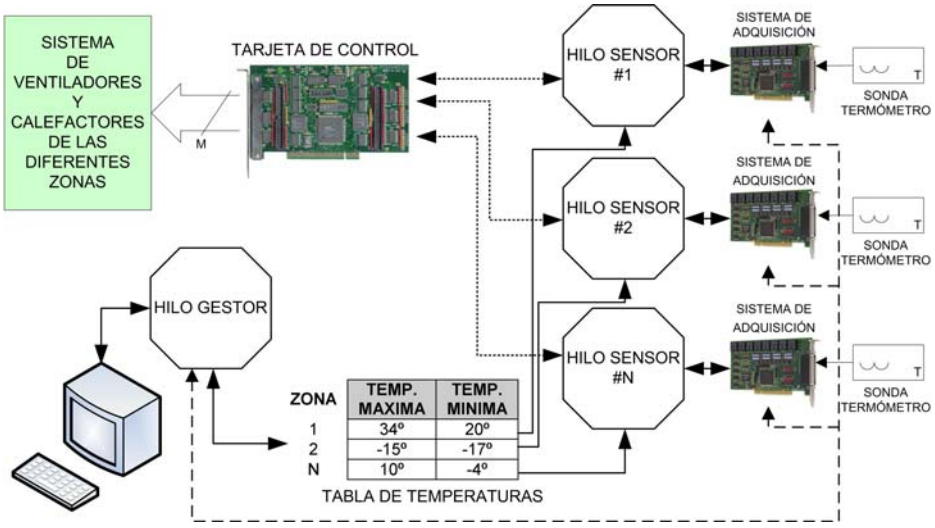

Figura 3. Escenario de aplicación

Por ello se ha desarrollado un entorno compuesto por dos elementos, figura 4:

- Un servicio Windows que simula una interfaz Hardware compuesta por una serie de registros que controlan una serie de sensores que obtienen la temperatura de varios locales y unos actuadores que gobiernan el encendido/apagado de unos ventiladores y calefactores con los que se pretende realizar la regulación de temperatura en un rango definido, figura 5.
- Una aplicación gráfica que actúa como visor del hardware permitiendo ver y modificar el contenido de los registros que se estime oportuno.

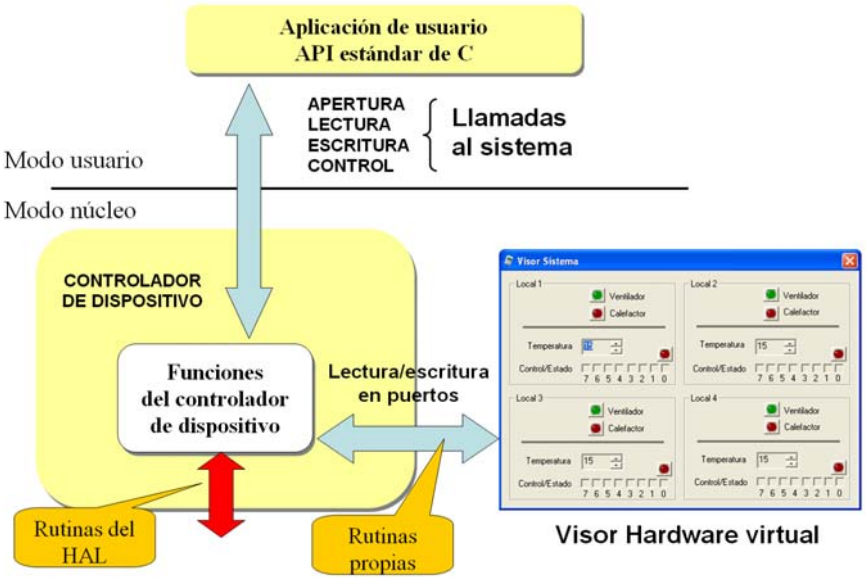

Figura 4. Hardware Virtual

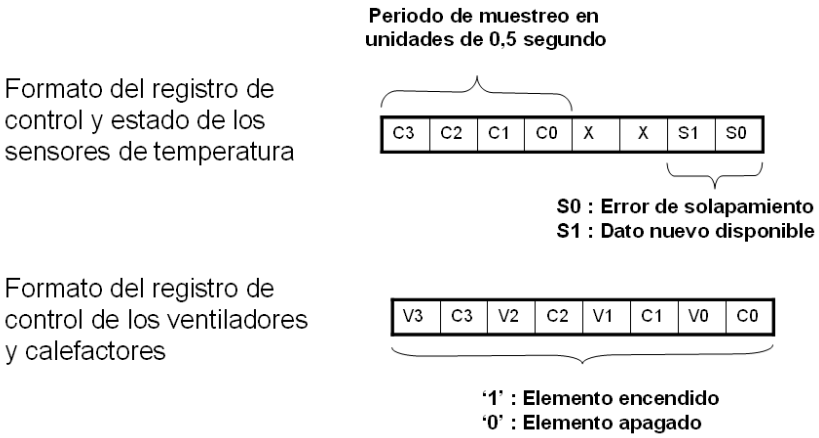

Figura 5. Formato de los registros de control

En las prácticas propuestas, figura 4, se propone al alumno la realización de la regulación de temperatura del local descrito previamente. Para ello se ha divido el escenario completo en tres partes:

- Aplicación de control: Código multitarea con la lógica de la aplicación y los mecanismos de sincronización entre hilos necesarios para la regulación del local. Para ello usará una librería de aplicación que ofrece llamadas específicas como ***obtenerTemp***, ***gestVentilador***, ***gestCalefactor***, etc.
- Librería de acceso a los dispositivos: Implementación de la librería de usuario mediante la API Windows genérica para el acceso a dispositivos.
- Controladores de dispositivos: Módulos del kernel que ofrecen los servicios básicos de lectura y escritura en los registros que controlan el hardware.

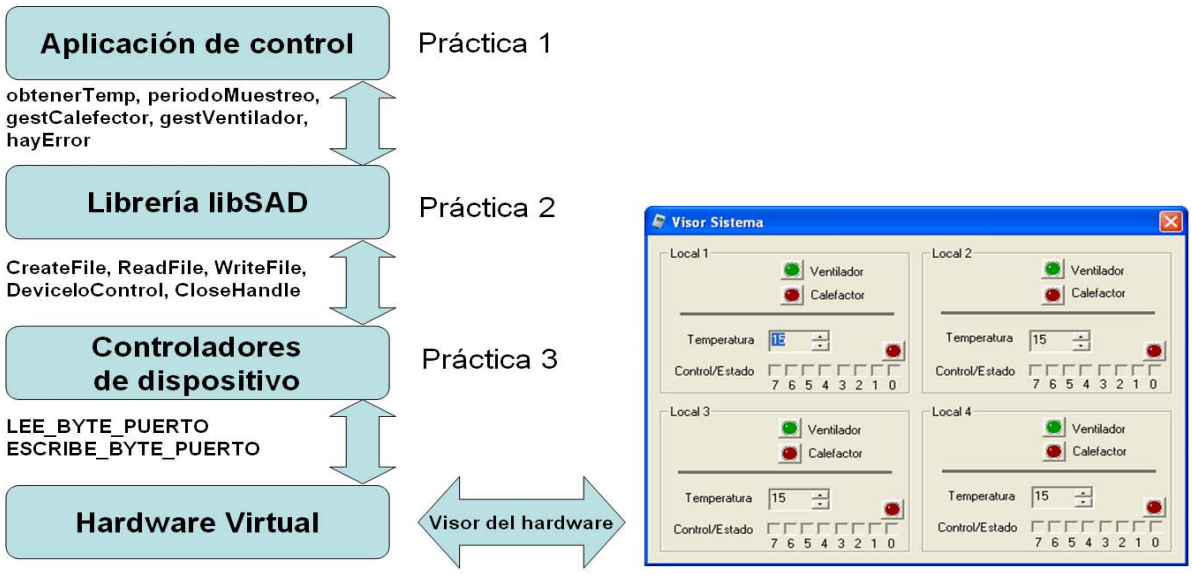

Figura 4. Prácticas desarrolladas

Las prácticas propuestas se desarrollan en secuencia En una primera fase se proporciona al estudiante una versión ya compilada e instalada de los controladores y de las librerías necesarias para el acceso al hardware de forma que el alumno se centra en la codificación de la aplicación de control multihilo y de los mecanismos de sincronización necesarios. En una segunda fase se procede al desarrollo de las librerías de acceso al controlador mediante el uso de la API estándar de Windows para el acceso a dispositivos. En este punto el alumno se abstrae de los problemas de la aplicación y se centra en la codificación de las rutinas y de los servicios que estas proporcionan. El alumno podrá probar su librería con su versión de la aplicación desarrollada en la práctica anterior o usar la versión de aplicación ya compilada proporcionada por el profesor. Finalmente se procede como última práctica a la codificación del controlador de dispositivo propiamente dicho. El desarrollo de este controlador es prácticamente idéntico al seguido para un hardware real. El único punto donde difiere es en el acceso a los puertos donde deben usarse dos rutinas que se proporcionan en forma de librería en lugar de las rutinas equivalentes del HAL (Hardware Abstraction Layer).

- unsigned char LEE_BYTE_PUERTO(puerto);
- void ESCRIBE_BYTE_PUERTO(puerto, dato);

La prueba del controlador codificado por el alumno es el momento más crítico para la integridad del sistema. En las dos primeras prácticas se prueba el código escrito por el alumno en modo usuario por lo que los errores de programación como punteros erroneos, índices fuera de rango, etc, causan en el peor de los casos un malfuncionamiento de la aplicación. En esta última fase se sustituye el controlador original adoptándose las siguientes precauciones para evitar la desconfiguración del sistema:

- Solo el controlador de dispositivo de la práctica tiene permisos de modificación para todos los usuarios, evitándose así que puedan borrarse/modificarse accidentalmente otros drivers del sistema almacenados en el directorio %Windows%\system32\drivers.
- La rutina crítica durante el arranque del sistema es ***DriverEntry***, un error de programación en ella impediría el arranque. Para evitar esta circunstancia se proporciona al alumno un esqueleto de práctica con la rutina ya codificada de forma que el alumno solo tiene que proporcionar el código de las operaciones de lectura, escritura y control. Estas se ejecutan bajo demanda cuando se ejecute la aplicación de control y no impiden el rearranque del sistema en caso de error.

4. Entorno de desarrollo

Como entorno de compilación se usa *MinGW*[4] ("Minimalistic GNU for Windows") versión para Windows del compilador GCC. Este compilador puede ser no solamente para generar aplicaciones gráficas y de consola, sino también librerías estáticas y dinámicas y controladores de dispositivo.

Al alumno se le proporciona un árbol de directorios, que contiene los esqueletos de todas las prácticas a realizar. En la figura 5 se muestra el árbol de directorios y el contenido del directorio de la fase 1 de la primera práctica. El directorio ***include*** incluye los ficheros header con las interfaces de uso de los módulos que permiten el acceso al hardware virtual mientras que el directorio ***lib*** contiene una versión ya compilada de dichos módulos.

El alumno debe editar el fichero fuente añadiendo el código que estime oportuno para cumplir las especificaciones descritas para cada una de las prácticas. El fichero ***Makefile*** contiene las opciones de compilación con los path de búsqueda de los ficheros header y de las librerías. Para compilar y generar el correspondiente ejecutable será suficiente con situarse en el directorio apropiado y ejecutar la orden make. Una vez generado el ejecutable, arrancará el visor del hardware junto con su aplicación y comprobará si la regulación se realiza de forma satisfactoria.

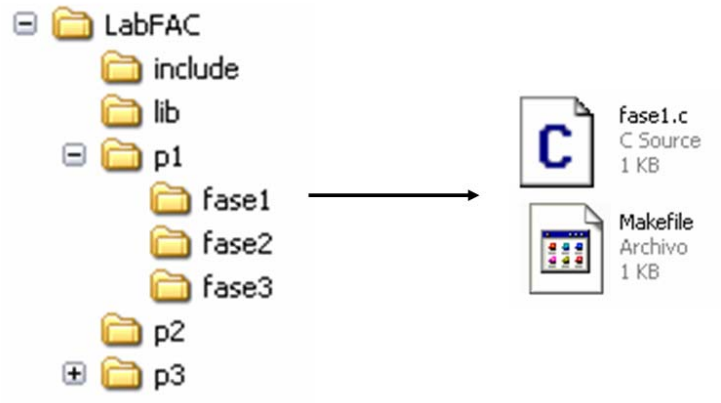

Figura 5. Esqueleto de prácticas

5. Conclusiones

El entorno propuesto se ha mostrado eficaz a la hora de ejercitar los conocimientos teóricos descritos en clase, básicamente programación concurrente, herramientas de sincronización entre hilos y los fundamentos de la codificación de controladores de dispositivo en entornos Windows. El estudiante podría a partir de este momento enfrentarse a retos más grandes como el acceso a un hardware real al conocer la interfaz software del controlador con un mínimo de profundidad.

Aunque las prácticas propuestas forman un todo, es posible abordar y probar de forma independiente cada una de ellas al proporcionarse versiones ya compiladas con las que enlazar el código del alumno. De esta forma es posible centrarse en la resolución de la problemática que se la plantea en cada nivel.

No es necesario un hardware real y se usan entornos de desarrollo GNU lo que facilita la realización de las prácticas en los ordenadores particulares de los alumnos dando mayor libertad de horarios a los mismos.

Referencias

[1] Microsoft Drivers Development Kit. http://www.microsoft.com/whdc/devtools/ddk/

[2] A. Da Silva y V. Hernandez, “Device Drivers teaching experience on Windows environments”, TELEC’04 International Conference, Santiago de Cuba, 2004

[3] A. Kornecki, H. Wojcicki, L. Peltier, J. Zalewski, N. Kruszynska, “Teaching Device Drivers Technology in a Real-Time Systems Currículum”

[4] http://www.mingw.org