Sesión 2

Manejo básico de la interfaz de programación Win32

Objetivos

Aprender los mecanismos básicos necesarios para utilizar la interfaz de programación Win32 en los programas C.

1 Conocimientos previos

Todos los servicios proporcionados por el sistema operativo Windows se llaman a través de una interfaz de programación estándar, que recibe el nombre de Interfaz Win32. Con objeto de poder utilizar dicho interfaz en los programas, Microsoft proporciona un sistema de desarrollo que recibe el nombre de *System Development Kit* (SDK). El SDK está formado por un conjunto de librerías que contienen las funciones necesarias para llamar a los servicios de la API Win32. Además de estas librerías, el SDK también proporciona una colección de ficheros de cabecera que, entre otras cosas, definen un conjunto de tipos de datos que se utilizan en la comunicación con el sistema operativo. Así el SDK contiene todos los elementos necesarios para desarrollar programas que llamen a los servicios de Windows.

La versión más actual del SDK puede descargarse desde el portal conocido como MSDN (*Microsoft Development Network*), que es un portal orientado a dar soporte a los programadores de aplicaciones para plataformas Windows. La dirección de este portal es http://msdn.microsoft.com. No obstante, el SDK forma parte también del entorno de desarrollo del Visual Studio. Por consiguiente, cuando desarrollamos con el Visual, tenemos el SDK a nuestra disposición y podremos generar programas que hagan uso de la API Win32.

En los programas en ensamblador que hiciste en las primeras prácticas de la asignatura, ya utilizaste funciones de la API Win32. Por ejemplo la función *ExitProcess()*, que debe colocarse al final de todo programa ensamblador para devolver el control al sistema operativo. Recordarás que *ExitProcess()* formaba parte de la librería `kernel32.lib`. Esta librería junto con la `user32.lib` y la `gdi32.lib` forman el núcleo básico de las librerías del SDK, aunque existen otras que contienen funciones de menor uso. Afortunadamente, cuado desarrollamos programas utilizando el entorno integrado del Visual Studio y estos programas usan funciones de la API Win32, no tenemos que preocuparnos de con qué librerías deben ser enlazados, ya que esto se encuentra automatizado en el entorno.

A continuación vamos a analizar qué pasos debemos seguir para utilizar funciones de la API Win32 en un programa C.

Desarrollo de la práctica

2 Primeros pasos

Vamos a ver los pasos que es necesario dar para utilizar una función de la API dentro de un programa. Para ello utilizaremos una función perfectamente conocida, que es la función *ExitProcess()*. Cuando dentro de un programa se llama a esta función, se termina el programa devolviéndose el control al sistema operativo. La función recibe como parámetro un número, que será el código de salida del programa. Este código se utiliza en técnicas relacionadas con el tratamiento de errores. Nosotros utilizaremos siempre el código 0, es decir, llamaremos a la función de la siguiente forma:

```
ExitProcess(0);
```

A continuación se muestra el listado de un programa en el que utilizaremos la función *ExitProcess()*. El programa utiliza dos funciones *printf()* para enviar dos mensajes a la pantalla (“Mensaje 1” y “Mensaje 2”). Sin embargo, como antes del *printf()* que envía el segundo mensaje se ejecuta la función *ExitProcess()*, en este punto el programa devuelve el control al sistema operativo y, por tanto, el segundo *printf()* no llegará a ejecutarse. A continuación se muestra el listado del programa, aunque en él faltan las inclusiones de los ficheros de cabecera necesarios.

```
// Fichero de cabecera para las funciones de la API
...

// Ficheros de cabecera para las funciones de C
...


main()
{
        printf(“Mensaje 1\n”);

        ExitProcess(0);

        printf(“Mensaje 2\n”);
}
```

H Crea un proyecto que se llame `2-2prog1` y agrégale el fichero `2-2prog1.c`. Entonces copia el listado del programa anterior en este fichero.

Ahora vamos a completar lo que le falta, la inclusión de los ficheros de cabecera. Como sabes, cada vez que utilizas una función de librería en un programa C, debes incluir en el programa el fichero de cabecera que se asocia a esa función. Como en este programa se utiliza la función *printf()*, debes incluir el fichero `stdio.h`.

H Incluye este fichero debajo del comentario *// Ficheros de cabecera para las funciones de c.*

Con las funciones de la API Win32 hay que hacer exactamente lo mismo. Sin embargo, en el caso de la API Win32 el problema es muy fácil de resolver, porque se usa el

mismo fichero de cabecera para todas las funciones. Este fichero se llama `windows.h`. Es decir, la regla es muy sencilla: **siempre que en un programa C se llame a una función de la API win32 hay que incluir el fichero de cabecera** `windows.h`.

H Incluye el fichero `windows.h` después del comentario *// Fichero de cabecera para las funciones de la API.*

En este punto, el programa ya está completo, así que vamos a generar el ejecutable y a probarlo.

H Compila y enlaza `2-2prog1.c` y ejecútalo desde CMD.EXE, comprobando que su comportamiento es el esperado.

H Finalmente, comenta la función *ExitProcess()*, compila y enlaza de nuevo el programa y ejecútalo desde CMD.EXE. Comprueba que ahora envía dos mensajes.

La API Win32 y las librerías del C

En el ejercicio anterior has visto cómo abandonar un programa llamando directamente al servicio del sistema operativo que tiene este cometido. Para ello has utilizado la función de la API Win32 *ExitProcess()*. Sin embargo, hay otras formas de abandonar un programa. Por ejemplo mediante la función *exit()* de la librería del C estándar. Esta función también recibe como parámetro el código de salida del programa. En el ejemplo que haremos a continuación utilizaremos el código 0. La función *exit()* necesita como fichero de cabecera `process.h`.

H En el programa `2-2prog1.c`, justo debajo de la sentencia que llama a *ExitProcess()* (que se encuentra comentada en este momento), escribe una nueva sentencia que llame a la función *exit()*, pasándole el parámetro 0. Ahora debajo de la inclusión del fichero `stdio.h`, incluye también `process.h`. Compila y enlaza de nuevo el programa y ejecútalo desde CMD.EXE comprobando que se comporta de la forma esperada.

¿Qué relación hay entre la función del C *exit()* y la función de la API Win32 *ExitProcess()*? La relación es total. Una parte importante de las funciones del C necesitan llamar a servicios del sistema operativo para llevar a cabo su cometido. Por ejemplo, en el código de la función *exit()* del C se llama al servicio *ExitProcess()*. Esto es así porque estamos utilizando una librería de C preparada para trabajar sobre una plataforma Windows. Si estuviéramos utilizando una librería de C preparada para trabajar en una plataforma Unix, la función *exit()* no se basaría en *ExitProcess()*, sino en el servicio proporcionado por Unix para abandonar un programa, que tendrá otro nombre diferente.

Como colusión importante debes tener en cuenta que **una parte importante de las funciones de librería del C estándar utilizan servicios del sistema operativo para llevar a cabo su cometido**.

Obtener ayuda sobre las funciones de la API win32

Para entrar en la ayuda debes hacer lo siguiente:

H Entra en el menú *Programas*, después en *Microsoft Developer Network* y dentro de él ejecuta la única opción disponible: *MSDN Library para Visual Studio 2005*. Esto te introducirá en la ayuda del entorno de desarrollo del Visual Studio. Ahí

encontrarás un océano de información sobre múltiples aspectos del desarrollo de aplicaciones para plataformas Windows. Uno de esos aspectos es la API Win32. Vamos a obtener información sobre ella.

H En la ventana de navegación que te aparece en la parte izquierda de la pantalla elige el panel *Contenido*. Este panel presenta el contenido de la ayuda ordenado por temas que se organizan de forma jerárquica. En el nivel superior de la jerarquía se muestran nueve temas, que van desde *Herramientas y lenguajes de programación* hasta *Ayuda sobre la ayuda*. Pulsando sobre un tema éste se expande en los temas jerárquicamente inferiores en los que dicho tema se encuentra organizado. En concreto, para buscar ayuda sobre las funciones de la API Win32 utilizaremos el tema *Windows API Reference*, al que puedes llegar siguiendo la ruta que se indica a continuación en la jerarquía de temas de la ayuda:

Desarrollo Win32 y COM **à** *Development Guides* **à** *Windows API* **à** *Windows API Reference*

A partir del tema *Windows API Reference*, vamos a buscar información sobre la función *MessageBox()*. Esta función pertenece al grupo de funciones que manejan cuadros de diálogo (*Dialog Box*).

H Abre el tema *Windows API Reference*. Para abrirlo no debes pulsar sobre el signo '+' (esto expande el tema en el panel *Contenido*), sino sobre el propio nombre del tema, lo cual hace que se abra una ficha con el contenido del tema. Ahora pulsa sobre el enlace *Functions by category*. Entonces se muestra una página en la que se indican las diferentes categorías de funciones en las que se organiza la API Win32 según su funcionalidad. La función *MessageBox()* pertenece a la categoría *Dialog Box*. Pulsa sobre este enlace, así obtendrás información general sobre el manejo de cuadros de diálogo, y en el apartado funciones, un listado de todas las funciones que los manejan. Busca la función *MessageBox()* y pulsa sobre ella. De esta forma habrás llegado al contenido de la ayuda sobre *MessageBox()*. Deja la ayuda de *MessageBox()* abierta, porque la vamos a utilizar en la siguiente sección.

3 Uso de las funciones de la API win32

Vamos a hacer algunos ejercicios en los que utilicemos funciones de la API win32. Nos basaremos en la ayuda para saber cómo utilizar las funciones. Empezaremos con la función *MessageBox()*.

H Céntrate ahora en la ayuda de *MessageBox()*. Lo primero que se muestra es una breve indicación acerca del cometido de la función. Se trata de una función cuyo objetivo es sacar una ventana con un mensaje y uno o varios botones. Esta ventana podría utilizarse, por ejemplo, para mostrar algún tipo de error ocurrido durante la ejecución de un programa. Un ejemplo de venta generado con *MessageBox()* se muestra a continuación:

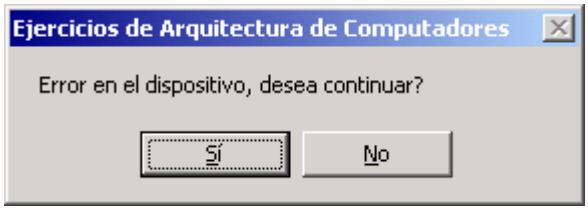

Figura 1: Ejemplo de ventana generada mediante la función MessageBox()

H Ahora vamos a analizar el prototipo de la función mostrado en la ayuda. Observarás que esta función requiere cuatro parámetros, que en el prototipo reciben los siguientes nombres: *hWnd*, *lpText*, *lpCaption*, *uType*. Vamos a olvidarnos del primer parámetro, al que daremos siempre el valor NULL, y nos concentraremos en los otros tres.

- *lpText* es un puntero a la cadena de caracteres que será mostrada dentro de la ventana. En el ejemplo anterior esta cadena es "*Error en el dispositivo, desea continuar?*".
- *lpCaption* es un puntero a la cadena de caracteres que aparece en la barra de título de la ventana: "*Ejercicios de Arquitectura de Computadores*" en el ejemplo anterior.
- *uType* especifica el contenido y el comportamiento de la ventana. Hay una serie de números diferentes que podemos colocar en este parámetro. Cada número determinará un comportamiento diferente de la ventana. Sin embargo, con objeto de hacer más agradable la programación, para indicar el valor de este parámetro no se utilizan directamente constantes numéricas, sino constantes definidas en los ficheros de cabecera del SDK. Mirando en la ayuda, indica a continuación qué botones tendría una ventana que en el parámetro *uType* recibiera la constante MB_RETRYCANCEL.

Antes de hacer tu primer programa utilizando la función *MesaggeBox()* hay que resaltar otro aspecto que antes hemos pasado por alto, los tipos de los parámetros recibidos por la función. Estos tipos, en principio, no se parecen a los que utilizamos habitualmente en C (char, int, char *, int *, etc.).

H Observando la ayuda de *MesaggeBox()*, escribe a continuación el tipo de los parámetros *lpText* y *lpCaption*.

H Indica el tipo del parámetro *uType*.

Tipos de datos proporcionados por el SDK

El SDK de Windows utiliza sus propios tipos de datos, aunque todos ellos están basados en los tipos que proporciona el C estándar. Lo que hace el SDK es definir sus nuevos tipos a partir de los del C estándar utilizando ficheros de cabecera. El objetivo de estos

nuevos tipos es dotar de mayor significado a los parámetros y valores de retorno manejados por las funciones de la API.

La referencia de todos los tipos de datos manejados por el SDK está disponible en el tema de ayuda *Windows Data Types*, que se encuentra en la misma ubicación que el resto de información de la API, es decir, en

Desarrollo Win32 y COM **à** *Development Guides* **à** *Windows API* **à** *Windows API Reference*

H Con objeto de no perder la información de *MessegeBox()*, que ya tienes abierta en una ficha, abre el menú *Ventana* y selecciona la opción *Nueva Ventana*. Esto te proporciona una nueva ficha que replica el contenido de la última ficha abierta, en nuestros caso, la información sobre la función *MessegeBox()*. Ahora en el panel *Contenido* expande el tema *Windows API Reference*. Bajo él observarás, entre otros, el tema *Windows Data Types*, ábrelo (pulsando sobre él y no sobre el singno '+'). Observarás entonces una tabla en la que se muestran todos los tipos de datos manejados por la API Win32. En este momento tienes dos fichas abiertas, una con información de la función *MessageBox()* y otra con la información de los *Windows Data Types*. Buscaremos ahora en esta ficha información sobre los tipos de los parámetros usados en *MessageBox()*.

Empezaremos con los parámetros *lpText* y *lpCaption*. Antes habrás contestado que el tipo de estos parámetros es LPCTSTR.

H Busca este tipo en la ayuda. La definición que encontrarás es la siguiente: "An LPCWSTR if UNICODE is defined, an LPCSTR otherwise".

UNICODE es una forma de representar caracteres en los programas, que nosotros no utilizamos en las prácticas. Por tanto, el tipo LPCTSTR es en realidad un LPCSTR.

H Busca LPCSTR en la ayuda. Encontrarás la siguiente información: "Pointer to a constant null-terminated string of 8-bit Windows (ANSI) characters". Resumiendo, puntero a una cadena de caracteres terminada con el carácter nulo, que son las cadenas estándar utilizadas en lenguaje C. Es decir, en algún lugar de los ficheros de cabecera el tipo "*LPCSTR*" está definido como un "*char **".

H Busca UINT en la ayuda. Encontrarás la siguiente información. "Unsigned INT", que se explica por sí misma.

UNICODE frente a ASCII

Cuando se van a utilizar funciones de la API Win32 que manejan caracteres, es necesario especificar qué tipo de codificación se usa para los caracteres manejados. Hasta hora, siempre has trabajado con programas en los que se codifican los caracteres siguiendo el estándar ASCII, que se engloba a su vez en el estándar ANSI. Sin embargo, los caracteres pueden codificarse también siguiendo el estándar UNICODE (que codifica los caracteres mediante códigos de 16 bits, en vez de 8 como el ASCII).

Debido a que los programas pueden diseñarse para que trabajen con caracteres codificados según uno u otro estándar, existe una versión doble de toda función de la API Win32 que trabaje con caracteres, una versión opera con caracteres ASCII y otra con caracteres UNICODE. Este es el caso de la función *MessageBox()*. El sistema de desarrollo elige automáticamente la versión de la función a utilizar según se encuentre configurado el proyecto.

Los programas C que escribimos en estas prácticas siguen el estándar de codificación ASCII. Esto es así por la forma en la que se definen las cadenas de caracteres, las funciones de librería que utilizamos (*printf()* y *scanf_s()* trabajan con cadenas ASCII), etc. Por consiguiente, cuando ahora utilicemos funciones de la API Win32, desearemos que el sistema de desarrollo utilice la versión ASCII de las mismas. Sin embargo, la configuración por defecto de un proyecto hace que el sistema de desarrollo utilice la versión UNICODE, lo que nos llevaría a generar programas que no manejan correctamente las cadenas de caracteres. Debido a esto será de crucial importancia en estas prácticas configurar los proyectos de modo que se utilicen las versiones ASCII de las funciones de la API Win32. Enseguida veremos cómo se lleva a cabo esta configuración.

Tras este análisis de diversos aspectos de la función *MessageBox()*, ya estás preparado para empezar a utilizarla.

Ejemplo simple de uso de *MessageBox()*

H Crea un proyecto que se llame `2-2prog2` de la forma habitual. Lo primero que harás ahora es configurarlo para que utilice las versiones ASCII de las funciones de la API Win32. Para ello, en el *Explorador de soluciones* pulsa con el botón derecho del ratón sobre el nombre del proyecto y en el menú que se abre elige *Propiedades*. En el árbol de propiedades elige *Propiedades de configuración* **à** *General*. En la parte derecha de esta ventana observarás la propiedad *Juego de caracteres*, que por defecto se encuentra configurada con el valor *Utilizar juego de caracteres unicode*. Debes de cambiar esto por el valor *Sin establecer*. Esto hará que el sistema de desarrollo elija al generar el programa las funciones de la API Win32 que trabajan con caracteres ASCII. Ahora agrega al proyecto el fichero `2-2prog2.c`. Entonces escribe en este fichero un programa que muestre una ventana exactamente igual a la indicada en la figura 1. Define en la función *main()* las cadenas de caracteres que necesites para *MessageBox()*. Recuerda que para definir una cadena puedes utilizar la siguiente sintaxis:

```
char cadena[] = “Hola mundo”;
```

Recuerda también que el primer parámetro que tienes que pasar a *MessageBox()* debe ser NULL.

H Una vez que hayas escrito el programa, compílalo y enlázalo, ejecútalo desde CMD.EXE y comprueba su correcto funcionamiento. Deberás observar una ventana como la mostrada en la figura 1. Pulsando sobre cualquiera de los botones el programa terminará.

H Par que observes la importancia de configurar adecuadamente el proyecto, vuelve a abrir la ventana de propiedades de éste y en la propiedad *Juego de caracteres*, vuelve a poner el valor *Utilizar juego de caracteres unicode*. Compila y enlaza de nuevo el programa. Se producirán avisos durante la compilación, ya que la versión de *MessageBox()* elegida por el sistema de desarrollo espera recibir punteros a cadenas UNICODE y las cadenas definidas en nuestro programa son ASCII. No obstante el programa se genera. Ejecútalo desde CMD.EXE. Observarás que la información mostrada en la ventana no es la esperada, ¿qué ocurre? Indícalo a continuación:

Combinación de constantes

El parámetro *uType* tiene un tamaño de 32 bits y se organiza en cuatro campos[1]. Cada campo es de un determinado número de bits, de forma que entre los cuatro campos deben sumar los 32 bits totales del parámetro. Cada campo puede recibir un valor independientemente de los demás campos. Cada valor diferente que puede ser asignado a un campo recibe el nombre de *flag*. Vamos a analizar esto detenidamente en la ayuda.

H Ubícate en la zona de la ayuda referente al parámetro *uType*. En el segundo párrafo pone lo siguiente: "*To indicate the buttons displayed in the message box, specify one of the following values.*" En este punto se está indicando el cometido del primer campo de *uType*. Este campo determina la combinación de botones que va a tener la ventana. Para dar un valor a este campo hay que cargarlo con un *flag*, que no es otra cosa que una constante numérica. No obstante, según se ha comentado previamente, para hacer más cómoda la programación se utilizan constantes definidas en los ficheros de cabecera del SDK, que son más legibles que las constantes numéricas. Según puedes observar en la ayuda, los *flags* diferentes con los que podemos cargar este campo son: MB_ABORTRETRYIGNORE, MB_CANCELTRYCONTINUE, MB_HELP, MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO y MB_YESNOCANCEL.

La siguiente característica relativa a la ventana mostrada por *MessageBox()* que se puede configurar con el parámetro *uType* es si se desea que muestre algún tipo de icono en la ventana. Esta característica se configura con el siguiente campo de *uType*. Los *flags* con los que podemos cargar este campo son MB_ICONEXCLAMATION, MB_ICONWARNING, MB_ICONINFORMATION, MB_ICONASTERISK y MB_ICONQUESTION.

Pero ¿cómo podemos asignar un valor a más de un campo del parámetro *uType*? Para esto se utiliza la técnica de combinar *flags* mediante el operador OR del lenguaje C, que se expresa mediante el símbolo '|'. Así por ejemplo, supón que queremos obtener una ventana con las siguientes características:

- Que contenga los botones YES, NO y CANCEL.
- Que muestre un icono de *stop*.

Esto requiere utilizar dos *flags*: MB_YESNOCANCEL para especificar los botones de la ventana y MB_ICONSTOP para indicar que se desea un icono de *stop* en la ventana. Los *flags* se combinan entonces mediante el operador '|'. Lo que se pasaría a la función en el parámetro *uType* sería: MB_YESNOCANCEL | MB_ICONSTOP.

H Para probar esto utilizaremos el proyecto anterior, es decir, `2-2prog2`. Primero vuelve a configurar la propiedad *Juego de caracteres* del proyecto con el valor *Sin establecer*, para que el sistema de desarrollo elija las funciones de la API que manejan caracteres ASCII. Después en el programa `2-2prog2.c` comenta la llamada a *MessageBox()* realizada anteriormente. Ahora escribe una nueva llamada a *MessageBox()* que muestre la misma ventana que en la versión anterior del programa, pero que además de mostrar los botones SI y NO, muestre también el icono de exclamación. Compila y enlaza el programa y ejecútalo desde CMD.EXE comprobando su correcto funcionamiento.

---

[1] El concepto de campo no aparece en la ayuda, pero es así como realmente se organiza el parámetro *uType*.

Valor de retorno

Las funciones de la API Win32 retornan habitualmente valores, que pueden ser usados en los programas que llaman a estas funciones. La ayuda de cada función proporciona información acerca de los valores retornados por la función en la sección *Return Value*.

H Utiliza la ayuda para conocer los valores que pueden ser retornados por la función *MessageBox()*.

H Crea un nuevo proyecto llamado `2-2prog3`. Primero configura la propiedad *Juego de caracteres* del proyecto con el valor *Sin establecer*, para que el sistema de desarrollo elija las funciones de la API que manejan caracteres ASCII. Entonces agrega al proyecto el fichero `2-2prog3.c`. Copia en este fichero el código del programa `2-2prog2.c`. Ahora modificaremos este programa para tratar las pulsaciones que haga el usuario sobre los botones de la ventana. Cuando se pulse SI, el programa debe mostrar en la consola el mensaje "Se ha pulsado SI". En el caso de que se pulse NO, mostrará "Se ha pulsado NO". Para hacer el programa ten en cuenta las siguientes indicaciones:

- Necesitarás capturar el valor retornado por *MessageBox()* (puedes usar una variable auxiliar para este cometido, que tendrás que definir al principio de la función *main()*). Después puedes comparar este valor con las constantes que definen los valores posibles retornados por la función y en función de estas comparaciones decidir qué mensaje envías a la consola.
- Para escribir en la consola utiliza *printf()*.

H Compila y enlaza el programa, ejecútalo desde CMD.EXE y comprueba que se comporta de la forma esperada.

4 Intercambio de información entre programas y sistema operativo

Los programas pasan información al sistema operativo en forma de parámetros en las llamadas a sus servicios. Asimismo, el sistema operativo puede retornar información al programa a través de los valores retornados por las funciones que llaman a los servicios. Sin embargo, hay ocasiones en las que un servicio tiene que proporcionar una gran cantidad de información al programa llamador, y toda esta información no se puede devolver en el valor retornado por el servicio, que siempre es un dato de tipo simple. Entonces ¿cómo el sistema operativo puede retornar a los programas estructuras complejas de información? Esto es precisamente lo que vamos a analizar en este apartado de la práctica. Para ello vamos a trabajar con el servicio *GetLocalTime()*, cuyo objetivo es proporcionar información acerca de la hora del sistema. Comenzaremos por obtener información de esta función.

H Abre la ayuda del Visual Studio y entonces abre la ficha:

Desarrollo Win32 y COM **à** *Development Guides* **à** *Windows API* **à** *Windows API Reference*

Aplicando un poco de intuición, busca la categoría a la que pertenece la función *GetLocalTime()*. Indica a continuación cuál es dicha categoría.

H Una vez localizada la función, abre la ficha de ayuda sobre ella.

Observarás que se trata de una función muy simple. No retorna ningún valor y recibe un solo parámetro, `lpSystemTime`, que es un puntero a una estructura de datos del tipo SYSTEMTIME. Se trata de un tipo de estructura definida en los ficheros de cabecera del SDK. Veamos ahora cómo es esa estructura.

H Pulsa sobre el enlace SYSTEMTIME para ver la información correspondiente a este tipo de estructura.

Observarás que la estructura está formada por una serie de campos de tipo WORD (que es lo mismo que un entero sin signo de 16 bits). El objetivo de cada campo es almacenar un tipo diferente de información sobre la fecha y hora del sistema. Así hay un campo para el año (`wYear`), otro para el mes (`wMonth`) y así sucesivamente. El sistema operativo utilizará una estructura de este tipo para proporcionar la fecha y hora al programa que se lo solicite.

Pero ¿cómo se establece la comunicación entre programa y sistema operativo? Primero explicaremos el proceso de forma genérica y luego lo programarás.

- Se define dentro del programa una estructura del tipo SYSTEMTIME. Es importante resaltar esto: la estructura de tipo SYSTEMTIME está dentro del programa.
- El programa transfiere el control al sistema operativo llamando a la función *GetLocalTime()* y le pasa un puntero (`lpSystemTime`) a su estructura SYSTEMTIME. Así el sistema operativo tiene una referencia a la estructura de datos SYSTEMTIME del programa.
- Utilizando el puntero `lpSystemTime`, el sistema operativo rellena la estructura SYSTEMTIME del programa y, después, retorna. En ese momento, el programa ya tiene toda la información sobre la fecha y hora del sistema en su estructura SYSTEMTIME.
- Finalmente, manejando de forma apropiada los campos de la estructura SYSTEMTIME, una vez que éstos han sido rellenados por el sistema, el programa podrá, por ejemplo, visualizar en pantalla la fecha y hora del sistema o llevar a cabo cualquier otra operación con esta información.

A continuación se proporciona la estructura de un programa cuyo objetivo es imprimir en pantalla la hora y minuto del sistema.

```
#include    <windows.h>
#include    <stdio.h>

main()
{
    // Definir una estructura llamada tiempo del tipo SYSTEMTIME
    ...

    // Poner a 0 todos los campos de la estructura
    tiempo.wYear=0;
    ...
```

```
    // Imprimir el contenido de los campos hora y minuto de la
    // estructura tiempo. Usar printf()
    ...

    // Llamar a la función GetLocalTime()
    ...

    // Volver a imprimir el contenido de los campos hora y minuto
    // de la estructura tiempo. Usar printf()
    ...
}
```

La salida que debe generar este programa es la siguiente:

```
Hora: 0

Minuto: 0

Hora: xx

Minuto: yy
```

Donde xx e yy son, respectivamente, la hora y minuto del sistema en el momento de la ejecución del programa.

H Crea un proyecto llamado `2-2prog4`. En este ejemplo no es necesario que modifiques la propiedad *Juego de caracteres* del proyecto, ya que la única función de la API que vamos a utilizar es *GetLocalTime()* y esta función no maneja cadenas de caracteres. Agrega al proyecto el fichero `2-2prog4.c`. Copia en este fichero el listado del programa anterior y completa las sentencias que faltan. Compila y enlaza el programa. Antes de ejecutarlo debes comprobar la hora del sistema, con objeto de contrastarla con la proporcionada por tu programa. Para ello puedes ejecutar en la consola de CMD.EXE el comando TIME. Este comando después de indicar la hora espera que el usuario introduzca una hora nueva. En este punto pulsa ENTER ya que no quiere modificar la hora. Después ejecuta tu programa desde CMD.EXE comprobando que funciona correctamente.

H Finalmente, crea un proyecto llamado `2-2prog5`. Haz en él un programa que imprima en la consola la fecha en el formato dd-mm-aaaa. Para hacer este programa debes conocer que en los especificadores de formato de *printf()* puede indicarse la longitud de los campos que se imprimen. Para ello se indica el tamaño del campo entre el % y la letra del especificador de formato. Así para imprimir un dato en decimal en un campo de 5 caracteres de ancho debe usarse el especificador '%5d', en vez de '%d'. Teniendo esto en cuenta, haz el programa indicado y comprueba su correcto funcionamiento.

El mecanismo de intercambio de información con los programas utilizado por *GetLocalTime()* (pasar información en una estructura) es el mecanismo estándar utilizado por otra muchas funciones de la API Win32 para proporcionar información a los programas.

En los ejercicios adicionales se propone practicar este mecanismo con otras dos funciones de la API Win32.

5 Ejercicios adicionales

E Realiza un programa que consulte al sistema operativo el usuario que se encuentra activo (*logged in*) en el sistema e imprima su identificador en la consola. En tu caso, se tratará del usuario *Alumno*. Para ello deberás utilizar la función de la API *GetUserName()*. Utiliza la ayuda para conocer el funcionamiento de esta función. Para encontrar la ayuda de la función sin falta de realizar una búsqueda puedes abrir la ficha

Desarrollo Win32 y COM **à** *Development Guides* **à** *Windows API* **à** *Windows API Reference*

Elige en ella el enlace *Functions in Alphabetical Order* y a partir de aquí podrás encontrar la información de la función sin mayor problema.

En la realización del programa debes utilizar la constante UNLEN. En la ayuda se indica el fichero de cabecera en el que se encuentra definida esta constante.

Realiza este programa en el proyecto `2-2prog6`.

E Realiza un programa que consulte al sistema operativo el nombre del equipo e imprima su identificador en la consola. Para ello deberás utilizar la función de la API *GetComputerName()*.Utiliza la ayuda para conocer el funcionamiento de esta función. En la realización del programa debes utilizar la constante MAX_COMPUTERNAME_LENGTH. En la ayuda se indica el fichero de cabecera en el que se encuentra definida esta constante. Para saber el nombre del equipo y así comprobar que tu programa funciona correctamente, pulsa con el botón derecho del ratón sobre *Mi PC* y elige *Propiedades*. Después elige la ficha *Identificación de red* y pulsa el botón *Propiedades*. Entonces se abre una ventana en la que aparece un campo con el nombre del equipo. Realiza este programa en el proyecto `2-2prog7`.

E ¿Comprendes el cometido del parámetro *nSize* en las funciones *GetUserName()* y *GetComputerName()*? Haz las modificaciones oportunas en los programas anteriores (`2-2prog6.c` y `2-2prog7.c`) para mostrar el valor tomado por la variable relativa a *nSize* después de la llamada a *GetUserName()* o *GetComputerName()*. ¿Comprendes el valor que toma? Si tienes dudas, pregúntale a tu profesor.