Sesión 3

Procesos

Objetivos

Comprender la información mostrada por el *Administrador de tareas* acerca de los procesos y aplicaciones que se encuentran en ejecución en el sistema en un momento dado.

Conocer algunos procesos de sistema fundamentales en la plataforma Windows.

Saber utilizar la función *CreateProcess()* para poner en ejecución un nuevo proceso.

1 Conocimientos previos

Conocimiento de los estados por los que puede pasar un proceso en un sistema multitarea, así como conocimientos básicos sobre la planificación de los procesos.

Desarrollo de la práctica

2 El administrador de tareas

Uno de los objetivos fundamentales del *Administrador de tareas* de Windows es proporcionar información acerca de los procesos que se encuentran en ejecución en el sistema. Vamos a centrarnos de momento en observar los procesos en ejecución.

H Asegúrate de que no tienes ninguna aplicación abierta en el sistema. Ahora abre el *Administrador de tareas* y elige la ficha *Procesos*. En este momento observarás todos los procesos que se están ejecutando actualmente en el sistema.

Para cada proceso se muestran varios campos de información. Lo normal es que los campos mostrados sean los siguientes[1]:

- **Nombre de imagen**: Es el nombre del programa ejecutable desde el que se cargó el proceso.
- **PID**: Es un número entero que identifica al proceso dentro del sistema. A cada proceso se le asigna un número diferente. PID es el acrónimo de *Process Identifier*.

[1] Puede que en tu sistema los campos que aparezcan no sean exactamente estos, ya que los campos mostrados son configurables. Luego haremos alguna prueba sobre esto.

- **CPU**: Indica el porcentaje de CPU consumida por un proceso.
- **Tiempo de CPU**: Es el tiempo total que un proceso ha estado ocupando la CPU desde que ha sido cargado en el sistema.
- **Uso de memoria**: Es la cantidad de memoria utilizada por el proceso.

Procesos de sistema

En las clases de teoría se indicó que el sistema operativo cuenta con un conjunto de programas, conocidos como programas de sistema, que se mantienen habitualmente en ejecución durante toda sesión de trabajo y que proporcionan servicios de diversos tipos a los usuarios y a otros programas. En concreto en este momento, todos los procesos que observas con el *Administrador de tareas* son procesos de sistema. Estos procesos han sido puestos en marcha automáticamente por el sistema durante el arranque.

Vamos a conocer algunos procesos de sistema muy importantes que se ejecutan en las plataformas Windows. Comenzaremos por el *Proceso inactivo del sistema*. Este proceso es el que se ejecuta cuando la CPU no tiene trabajo útil que llevar a cabo.

H Indica a continuación el identificador de este proceso y el porcentaje de CPU utilizado por él:

PID:
Porcentaje de uso de la CPU:

H ¿Qué conclusión sacas del porcentaje de CPU utilizado por este proceso? Escríbela a continuación.

Otros procesos críticos del sistema son el SMSS.EXE, el CSRSS.EXE y WINLOGON.EXE. El SMSS.EXE, conocido como *Session Manager*, es el que controla toda una sesión de trabajo, desde que el sistema se inicializa hasta que se apaga. El cometido del CSRSS.EXE es algo más complejo de explicar, pero para darte cuenta de su importancia te bastará saber que este proceso hace de intermediario entre los programas y el sistema operativo para dar servicio a las llamadas de la API Win32. Finalmente, WINLOGON.EXE es el que controla las sesiones de usuario. Así por ejemplo, este proceso controla la ventana en la que introducimos nuestro nombre de usuario y clave para comenzar una sesión en el sistema.

H Asegúrate de que has observado estos tres procesos en el *Administrador de tareas*. Quédate con sus nombres porque siempre estarán presentes en toda sesión de trabajo.

Vamos a ver ahora otro proceso fundamental que es el EXPLORER.EXE[2]. Este proceso es el que proporciona la funcionalidad de la interfaz gráfica con el usuario, así el EXPLORER.EXE es el principal medio de comunicación entre el usuario y el sistema. Vamos a eliminar este proceso para ver que ocurre.

[2] No confundir con el Internet Explorer: el Explorer y el Internet Explorer son programas diferentes

H Pulsa con el botón derecho sobre el proceso EXPLORER.EXE. Elige la opción *Terminar proceso*. ¿Qué ocurre? Observarás que desaparecen todos los iconos del *Escritorio*, así como la *Barra de tareas* y el *Menú de inicio*. Al eliminar el EXPLORER.EXE hemos cortado prácticamente toda posibilidad de comunicación con el usuario. En definitiva nos hemos "cargado" la sesión de usuario actual.

Lo único que podemos hacer ahora es eliminar la presente sesión de usuario (en la que ya no podemos hacer nada) y poner en marcha una nueva sesión. Esto podemos hacerlo sin problemas porque quien controla las sesiones de usuario es el proceso WINLOGON.EXE, que sigue activo. Este proceso se activa con la pulsación de `Ctrl-Alt-Supr`.

H Pulsa `Ctrl-Alt-Supr`. WINLOGON.EXE se activa mostrando la ventana *Seguridad de Windows*, que es la que usas para lanzar el *Administrador de tareas*, pero que también te permite cerrar la sesión de usuario. Elige *Cerrar sesión*. Ahora comienza como siempre una nueva sesión. Observa que todo vuelve a estar como estaba. WINLOGON ha dirigido todo este proceso. Al crear la nueva sesión de usuario ha cargado el EXPLORER.EXE y por tanto hay una interfaz con el usuario disponible para atender a nuestras peticiones.

H Abre el *Administrador de tareas* y comprueba que el EXPLORER.EXE está de nuevo en ejecución.

Ahora vas a hacer una prueba para diferenciar el EXPLORER.EXE del *Internet Explorer*.

H Mantén el *Administrador de tareas* abierto. Abre el Internet Explorer y observa el nuevo proceso que aparece en la ventana del *Administrador de tareas*. El nuevo proceso que se acaba de cargar es el Internet Explorer. Indica a continuación el nombre de este proceso.

Otro proceso de sistema que utilizamos habitualmente es el CMD.EXE, que proporciona una interfaz textual con el usuario. Una diferencia importante entre este proceso de sistema y los que hemos visto hasta ahora es que CMD.EXE no es arrancado automáticamente por el sistema, sino tras la solicitud realizada por el usuario. Vamos a utilizar CMD.EXE para diferenciar entre el concepto de programa (que es un conjunto de instrucciones y datos cargados en un fichero ejecutable) del de proceso (que es un programa puesto en ejecución). Para ello lo que vamos a hacer es generar varios procesos a partir de un mismo programa.

H En primer lugar vas a localizar el programa ejecutable. CMD.EXE se ubica en la carpeta *\windows\system32*. Abre esta carpeta con el explorador de Windows y busca en ella CMD.EXE. Ahora haciendo doble clic sobre el programa CMD.EXE ponlo en ejecución. Observarás en el *Administrador de tareas* el proceso CMD.EXE. Anota a continuación su PID.

PID =

Haz de nuevo doble clic sobre el programa CMD.EXE. Observarás un nuevo proceso CMD.EXE en ejecución. Anota el PID de este nuevo proceso.

PID =

A partir de un único programa hemos generado dos procesos diferentes. El sistema operativo los diferencia porque ha asignado un PID distinto a cada uno de ellos.

Porcentaje de uso de la CPU

Es habitual que los procesadores actuales dispongan de más de un núcleo (*core*) de procesamiento. En concreto, los procesadores de los ordenadores del Laboratorio disponen de dos núcleos. Cada núcleo representa una CPU completa y, por tanto, nuestros ordenadores cuentan con dos CPUs. El número de núcleos activos en un sistema puede observarse mediante el *Administrador de tareas*.

H Abre el Administrador de tareas. Elige la ficha *Rendimiento*. En la zona de esta ficha identificada como *Historial de uso de la CPU* se muestran tantos recuadros como núcleos activos hay en el sistema. En este momento debes observar dos recuadros, que corresponden a los dos núcleos del procesador.

Para comprender la planificación de procesos en la CPU es mejor empezar experimentando con un sistema que tenga una sola CPU. Esto podemos conseguirlo haciendo que nuestro sistema funcione con un solo núcleo activo. Para ello, tenemos que modificar un fichero del sistema operativo en el que se especifican diversos aspectos relativos a la carga e inicialización del sistema. Se trata del fichero `boot.ini`. En nuestro sistema, este fichero se encuentra en la carpeta raíz de la unidad `C:`. De momento no lo ves, porque es un fichero oculto del sistema.

H Para poder manipular este archivo debes configurar el explorador de Windows para que permita ver archivos protegidos del sistema. Para ello debes abrir una ventana de exploración de archivos y en el menú *Herramientas*, elegir *Opciones de carpeta*. Posteriormente elige la ficha *Ver* y en ella deberás configurar adecuadamente dos opciones para que se muestren los archivos ocultos del sistema.

H Una vez que observes `boot.ini` en la carpeta raíz de la unidad `C:` debes eliminar el atributo *Sólo lectura*, para que se pueda modificar.

H Abre el fichero con el *Bloc de notas*.

Este fichero contiene dos secciones: *boot loader* y *operating systems*. Cada línea de la sección *operating systems* corresponde a un sistema operativo instalado en el sistema. En nuestro caso observarás una línea correspondiente al Windows Server 2003 instalado en nuestro sistema. En la parte final de esta línea se indican las opciones de arranque. Estas se separan mediante un espacio en blanco y van precedidas del carácter '/'. Para indicar el número de núcleos con los que va a trabajar el sistema se utiliza la opción '/numproc='.

H Añade la opción '/numproc=1' a `boot.ini`. Recuerda que debe estar separada de las otras opciones mediante un espacio. Usando esta opción ordenamos al sistema que funcione con un solo núcleo.

H Arranca de nuevo el sistema. Abre el *Administrador de tareas*, ficha *Rendimiento*. Entonces debes observar que en la zona *Historial de uso de la CPU* solo hay un recuadro, que corresponderá al único núcleo activo.

Ahora realizaremos una serie de experimentos con esta configuración del sistema y luego dejaremos las cosas como estaban inicialmente.

En los experimentos realizados anteriormente en esta práctica, pudiste observar que la CPU suele estar muy descargada, porque los programas que normalmente ejecutamos, basados en mucha interacción con el usuario, consumen muy poca CPU. Exactamente lo mismo ocurre con los programas de sistema. Por ejemplo, piensa en WINLOGON.EXE. Prácticamente sólo trabaja en el inicio y finalización de una sesión de usuario, el resto del tiempo está bloqueado esperando una señal proveniente del teclado (pulsación de `Ctrl-Alt-Supr`). Para cambiar esta situación, vamos a introducir en el sistema un proceso que consuma CPU intensivamente.

H Crea un proyecto que se llame `2-3prog1` y agrégale el fichero `2-3prog1.c`. Utilizando este fichero, escribe un programa que contenga dos bucles `for` anidados. El bucle interior no ejecutará ninguna sentencia. El bucle exterior sólo ejecutará el bucle interior. Cada bucle incrementa una variable entera desde 0 hasta 1000000000 (mil millones). Esto asegura que el programa tarde mucho tiempo en ejecutarse. Como puedes observar, no hay ninguna operación de E/S en el programa, por tanto éste intentará usar la CPU tanto como le sea posible. Compila y enlaza el programa para obtener su ejecutable.

H Ahora vas a observar la ejecución de este programa con el *Administrador de tareas*. Abre el *Administrador de tareas*. Observa que en este momento la CPU está siendo ocupada, casi al cien por cien, por el *Proceso inactivo*. Abre la carpeta *Debug* del proyecto `2-3prog1` y ejecuta `2-3prog1.exe` haciendo doble clic sobre él. ¿Qué ocurre? Indica a continuación el porcentaje de CPU que utilizan en este momento el *Proceso inactivo* y `2-3prog1.exe`.

% de CPU proceso inactivo:
% de CPU proceso 2-3prog1.exe:

H En el *Administrador de tareas*, pulsando con el botón derecho del ratón sobre el proceso `2-3prog1.exe`, termina la ejecución de ese proceso, comprobando cómo la CPU pasa de nuevo a ser ocupada por el *Proceso inactivo*.

Ahora vamos a comprobar cómo los procesos compiten por la CPU. Para ello vamos a lanzar varias ejecuciones del programa `2-3prog1.exe`.

H Abre la carpeta *Debug* en la que tienes almacenado el programa `2-3prog1.exe`. Abre el *Administrador de tareas*. Ahora vas a ir lanzando sucesivas ejecuciones del programa `2-3prog1.exe` (haciendo doble clic sobre él) y vas a anotar a continuación el porcentaje de CPU que se asigna a cada proceso correspondiente a este programa cuando hay una, dos, tres y cuatro imágenes[3] de él en ejecución. Es posible que tras lanzar cada imagen tengas que esperar un poco de tiempo hasta que se estabilice la planificación, e incluso que tengas que minimizar y restaurar la ventana del *Administrador de tareas*. Para ver todas las imágenes en ejecución juntas, puedes pulsar con el ratón en la cabecera de la columna del *Administrador de tareas* denominada *Nombre de imagen*. Esto ordena todos los procesos en ejecución por su nombre. Deberás observar que la CPU se reparte equitativamente entre las diferentes imágenes en ejecución.

Una imagen en ejecución. % de CPU asociado a la imagen:

[3] El término *imagen* (o *imagen binaria*) es equivalente al de fichero ejecutable.

Dos imágenes en ejecución. % de CPU asociado a cada imagen:

Tres imágenes en ejecución. % de CPU asociado a cada imagen:

Cuatro imágenes en ejecución. % de CPU asociado a cada imagen:

H Para terminar estos procesos, en el *Administrador de tareas* pulsa con el botón derecho del ratón sobre cada imagen de `2-3prog1.exe` y en el menú contextual elige la opción *Terminar proceso*.

H ¿Qué mecanismo utiliza el sistema operativo para que un programa como `2-3prog1.exe` no monopolice totalmente la CPU cuando se ejecuta?

Finalmente modificaremos de nuevo el fichero `boot.ini` para que el sistema funcione con todos su núcleos activos, que es la situación habitual.

H Abre el fichero `boot.ini` con el *Bloc de notas*. Elimina en él la opción '/numproc=1'. Asegúrate de no dejar espacios en blanco al final de la línea correspondiente. Salva el fichero. Ahora arranca de nuevo el sistema. Abre el *Administrador de tareas* y usando la ficha *Rendimiento* comprueba que los dos núcleos vuelven a estar activos en el sistema. Para ello debes observar que en la zona *Historial de uso de la CPU* hay dos recuadros, correspondiendo cada uno de ellos a un núcleo del sistema.

Haremos ahora una última prueba de planificación con los dos núcleos activos.

H Abre el *Administrador de tareas*, ficha *Rendimiento*. Ejecuta el programa `2-3prog1.exe`. Observarás que el uso de la CPU no sube hasta el 100%, sino sólo hasta el 50%.

La explicación es que el uso de CPU mostrado por el *Administrador de tareas* indica la carga del procesador globalmente, es decir, teniendo en cuenta los dos núcleos. O sea, el uso de CPU es la media de la carga de ambos núcleos. En nuestro caso, solo tenemos un proceso que quiere hacer uso de la CPU constantemente. Sin embargo, el sistema tiene dos núcleos. En cada quantum de ejecución el proceso se da a un núcleo, quedando el otro núcleo descargado. La media de carga de ambos núcleos será del 50% y por tanto la media de carga del procesador como recurso global será también del 50%.

H Finalmente vuelve a marcar el archivo `boot.ini` como de solo lectura y configura el explorador de Windows para que los archivos ocultos del sistema no se muestren. Comprueba que has realizado esto correctamente, observando que el fichero `boot.ini` ya no se muestra.

Seleccionar la información proporcionada por el *Administrador de tareas*

Hasta ahora sólo nos hemos fijado en unos pocos datos entre todos los que puede mostrar el *Administrador de tareas*. Sin ánimo de ser exhaustivos, sólo se trata ahora de ver cómo un usuario puede elegir el tipo de información mostrada por el *Administrador de tareas*.

H Abre el menú *Ver* del *Administrador de tareas* y elige la opción *Seleccionar columnas*. Se abre la ventana del mismo nombre en la que se muestran todos los campos de información que se pueden mostrar para cada proceso. A modo de prueba puedes seleccionar (además de los campos que están ya seleccionados) el campo

Bytes de lectura de E/S. De esta forma se mostrará para cada proceso la cantidad de bytes leídos de los dispositivos de E/S durante su ejecución. Pulsa *Aceptar* para cerrar la ventana *Seleccionar columnas*, y observa ahora en la ficha *Procesos* (quizás tengas que hacer *Scroll* a la derecha) cómo aparece el nuevo campo *Bytes de lectura de E/S*.

Vamos a realizar un pequeño experimento, para que observes el funcionamiento de este campo. Para ello nos centraremos en el proceso EXPLORER.EXE. Como este proceso es el que proporciona la interfaz con el usuario, cualquier operación realizada por el usuario con el sistema de ficheros (crear, borrar, abrir y cerrar, tanto carpetas como ficheros) será gestionada por este proceso. Por otra parte, las operaciones realizadas sobre el sistema de ficheros se traducen en operaciones realizadas con el disco que es un dispositivo de E/S y, por tanto, cuando hagamos operaciones de lectura con el sistema de ficheros, tales como abrir una carpeta o abrir un fichero, eso se traducirá en bytes de E/S leídos por el proceso EXPLORER.EXE. Probemos esto.

H Manteniendo abierta la ficha *Procesos* del *Administrador de tareas*, observa el proceso EXPLORER.EXE. Vamos entonces a realizar algunas operaciones que hagan que este proceso lea bytes de los dispositivos de E/S. Para ello, prueba a abrir carpetas y ficheros de diversos tipos mientras observas el campo *Bytes de lectura de E/S*. Deberás observar cómo se incrementa el valor de este campo según llevas a cabo las operaciones de apertura.

H Vuelve a abrir la ventana *Seleccionar columnas* y desactiva el campo *Bytes de lectura de E/S* para dejar la configuración de la ficha *Procesos* como estaba al comienzo de la sesión de trabajo.

H Cierra todas las aplicaciones que tengas abiertas así como el *Administrador de tareas*.

La ficha aplicaciones

¿Qué contiene la ficha aplicaciones? En la jerga informática se entiende por aplicación a todo aquello que se encuentra entre un programa simple y un sistema software complejo, que puede estar formado por múltiples programas, ficheros, bases de datos, etc. Es decir, en su versión más simple una aplicación es un programa que al ponerlo en ejecución se convierte en un proceso. Sin embargo lo que se representa en la ficha *Aplicaciones* del *Administrador de tareas* no tiene nada que ver con esto. Por ello, siendo rigurosos, el nombre de esta ficha no es muy afortunado.

En realidad lo que muestra la ficha *Aplicaciones* es otro tipo de objetos diferentes a los procesos: muestra ventanas. Muchas aplicaciones de Windows utilizan ventanas para comunicarse con los usuarios. Pues bien, la ficha *Aplicaciones* muestra las ventanas abiertas por los procesos que se encuentran en ejecución en el sistema. Piensa que hay procesos que no usan ninguna ventana, como por ejemplo WINLOGON.EXE. Por ello, estos procesos nunca aparecen en la ficha *Aplicaciones*. Otros procesos utilizan una sola ventana que permanece siempre activa. Es el caso de CMD.EXE, que genera la ventana conocida como consola Win32. Finalmente hay otros procesos que pueden abrir y cerrar ventanas dinámicamente: este es el caso de EXPLORER.EXE.

H Abre el *Administrador de tareas* y selecciona la ficha *Aplicaciones*. En este momento no hay ningún proceso en ejecución que esté utilizando ventanas, por lo

que la ficha *Aplicaciones* se encuentra vacía[4]. Abre una consola. Como el proceso CMD.EXE utiliza una ventana, ésta se muestra en la ficha *Aplicaciones*. Abre a partir de *Mi PC* la unidad *C:*. Después vuelve a abrir *Mi PC*. Con esto has conseguido abrir dos nuevas ventanas que deben mostrarse en la ficha *Aplicaciones*. Ahora para saber a qué proceso pertenece cada ventana, pulsa con el botón derecho del ratón sobre cualquier entrada de la ficha *Aplicaciones* y elige la opción *Ir al proceso*. Esto provoca el paso a la ficha *Procesos*, mostrándose seleccionado el proceso propietario de la ventana. Comprueba que la ventana de la consola pertenece al proceso CMD.EXE y que las ventanas *C:* y *Mi PC* pertenecen a EXPLORER.EXE.

3 Creación de procesos

Vamos a trabajar ahora en la creación de procesos utilizando la función *CreateProcess()*. El objetivo de esta función es lanzar como proceso un programa ejecutable (`.exe`), que le pasamos como parámetro. Esta función tiene el siguiente prototipo:

```
BOOL CreateProcess(
   LPCTSTR  lpApplicationName,
   LPTSTR   lpCommandLine,
   LPSECURITY_ATTRIBUTES  lpProcessAttributes,
   LPSECURITY_ATTRIBUTES  lpThreadAttributes,
   BOOL     bInheritHandles,
   DWORD    dwCreatioFlags,
   LPVOID   lpEnvironment,
   LPCTSTR  lpCurrentDirectory,
   LPSTARTUPINFO   lpStartupInfo,
   LPROCESS_INFORMATION  lpProcessInformation);
```

Como puedes observar en el prototipo, se trata de una función que recibe muchos parámetros, los cuales determinan múltiples aspectos de comportamiento de los procesos que se crean. Sin embargo, en los casos más simples (que son los que nosotros manejaremos en esta práctica), muchos de estos parámetros reciben el valor NULL. A continuación se describen los parámetros de esta función:

- **lpApplicationName**: En aplicaciones Win32 (que es nuestro caso) se pasa NULL en este parámetro.
- **lpCommandLine**: Especifica la línea de comando completa que será utilizada por la función para crear el nuevo proceso. La línea de comando es una cadena de caracteres que puede estar formada por varios elementos, separados mediante espacios. El primer elemento siempre debe ser el nombre del programa ejecutable que será lanzado como proceso. Los elementos siguientes son parámetros que se pasan al ejecutable.

  Por ejemplo, para ejecutar un programa llamado `ejemplo.exe`, que no recibe parámetros y que se encuentra en el directorio activo, la línea de comando sería *"ejemplo.exe"*. También se puede indicar la ruta completa en la que se encuentra el ejecutable. Por ejemplo, si `ejemplo.exe` se encontrase en el directorio

[4] En realidad sí se está ejecutando un proceso que utiliza una ventana. Se trata del propio *Administrador de tareas*. Sin embargo, el *Administrador de tareas* nunca se muestra en la ficha *Aplicaciones*.

\temp\pruebas, se podría pasar como línea de comandos a *CreateProcess()*, *"\temp\pruebas\ejemplo.exe"*.

- **lpProcessAttributes y lpThreadAttributes**: Parámetro relacionado con la seguridad del proceso que se crea. Se pasa NULL en este parámetro para especificar los descriptores de seguridad por defecto. (En nuestro caso se pasa NULL en este parámetro.)
- **bInheritHandles**: Determina si el proceso hijo hereda los recursos (ventanas, ficheros, etc.) del proceso padre. Si "bInheritHandles == TRUE" se produce la herencia. (En nuestro caso se pasa TRUE en este parámetro.)
- **dwCreationFlags**: Determina diversos aspectos de comportamiento del proceso que se crea. Este campo se rellena con *flags* (constantes), que se pueden combinar mediante el operador '|'. Existen 13 *flags*. De ellos, solo utilizaremos uno, el CREATE_NEW_CONSOLE. Cuando se especifica este *flag*, se genera una nueva ventana de tipo consola para el proceso que se crea. Si no se especifica, el proceso creado hereda la ventana consola de su proceso padre, es decir, de CMD.EXE. (Fíjate que los programas que ejecutas desde CMD.EXE no crean una ventana nueva, sino que utilizan la ventana de CMD.EXE.)
- **lpEnvironment**: Puntero a un bloque de memoria que contiene las variables de entorno para el proceso. Se trata de unas variables (que no estudiaremos), que definen diversos aspectos de comportamiento del proceso. Cuando este parámetro recibe el valor NULL, el proceso que se crea hereda las variables de entorno de su proceso padre. (En nuestro caso se pasa NULL en este parámetro.)
- **lpCurrentDirectory**: Permite definir la ruta que será el directorio activo del proceso que se crea. Cuando este parámetro recibe el valor NULL, el directorio activo del proceso hijo será el mismo que el del proceso padre. (En nuestro caso se pasa NULL en este parámetro.)
- **lpStartupInfo**: Puntero a una estructura del tipo STARTUPINFO. Los campos de esta estructura definen diversos aspectos de creación del proceso, como por ejemplo, las características de la ventana que se asocia al proceso. Habitualmente, los campos de esta estructura se rellenan con sus valores por defecto, que son 0 o NULL.
- **lpProcessInformation**: Puntero a una estructura del tipo PROCESS_INFORMATION. El sistema operativo rellena esta estructura con información relativa al proceso que se crea, como por ejemplo, el PID que se asocia al proceso.

Con relación al valor retornado por la función, indicar que éste es de tipo BOOL. Si la función tiene éxito en la creación del nuevo proceso, ésta retorna TRUE, y si falla, FALSE.

Para practicar sobre el funcionamiento de la función *CreateProcess()*, vamos a realizar un programa, cuyo objetivo es ejecutar otros programas. El programa con el que vamos a trabajar tiene estructura en bucle infinito. En cada iteración del bucle pide por pantalla el nombre del programa ejecutable (`.exe`) que se desea ejecutar. Entonces trata de ejecutarlo mediante la función *CreateProcess()*. Finalmente muestra un mensaje indicando si ha habido éxito en la ejecución del programa, o por el contrario si la ejecución ha fallado. Terminada una iteración se pasa a la siguiente hasta que el usuario rompa la ejecución del programa pulsando `Ctrl-C`.

Escribir este programa resulta un tanto tedioso, porque hay que manejar muchos datos relativos a la función *CreateProcess()*. Debido a ello es preferible que trabajes sobre una versión ya casi completa del programa, que se encuentra en la capeta de la asignatura con el nombre `2-3prog2(incompleto).c`.

H Crea el proyecto `2-3prog2` de la forma habitual. Ahora tienes que configurar el proyecto para que el Visual elija la versión de *CreateProcess()* que trabaja con cadenas ASCII (fíjate que algunos campos de *CreateProcess()* son punteros a cadenas de caracteres y por eso, debe haber dos versiones de esta función). 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*. Elige para esta propiedad el valor *Sin establecer*. Entonces agrega al proyecto el fichero `2-3prog2.c` y copia en él el contenido de `2-3prog2(incompleto).c`.

En el archivo `2-3prog2.c` falta por completar algunas sentencias, que se marcan con el símbolo `(...)`. No obstante, antes de que empieces a completar este programa vamos a comentar algunos aspectos importantes del mismo.

En primer lugar hablaremos de las estructuras STARTUPINFO y PROCESS_INFORMACION utilizadas en este programa. Se trata de dos tipos de estructuras definidas en los ficheros de cabecera del sistema de desarrollo. Siempre que se desee utilizar la función *CreateProcess()*, hay que crear en el programa que va a llamar a *CreateProcess()* una estructura del tipo STARTUPINFO y otra del tipo PROCESS_INFORMATION. En el caso de `2-3prog2.c`, esto se hace en las dos primeras declaraciones de la función *main()* del programa.

```
STARTUPINFO si;
PROCESS_INFORMATION pi;
```

Gracias a estas declaraciones tenemos en el programa una estructura llamada `si` del tipo STARTUPINFO y otra llamada `pi`, del tipo PROCESS_INFORMATION.

Los campos de la estructura del tipo STARTUPINFO hay que rellenarlos con ciertos valores. En `2-3prog2.c` esto se hace en *main()* justo después de la declaración de variables, a continuación del comentario

```
\\ Inicialización de la estructura STARTUPINFO
```

La información almacenada en esta estructura será utilizada por la función *CreateProcess()* para determinar ciertos aspectos de comportamiento del proceso que se crea. Más adelante en esta práctica probaremos el cometido de alguno de los campos de esta estructura. De momento, observa que casi todos ellos se rellenan con los valores NULL ó cero, que son los valores por defecto. Una excepción a esto es el primer campo, llamado *cb*, que debe rellenarse siempre con el tamaño de la estructura del tipo STARTUPINFO. Para ello se usa el operador *sizeof()*.

La estructura del tipo PROCESS_INFORMATION no la rellena el programa (observa que no hay ninguna sentencia en `2-3prog2.c` que rellene los campos de esta estructura), sino el sistema operativo cuando ejecuta la función *CreateProcess()*. En ella el sistema operativo pone información relativa al proceso que se crea.

Volviendo al funcionamiento de `2-3prog2.c`, a continuación se muestra la salida que debe realizar este programa cuando se le pide que ejecute, en dos iteraciones de su

bucle, los programas `pepe.exe` y `2-3prog1.exe`. Deberás tener en cuenta esta salida por pantalla cuando completes el programa.

```
Nombre programa: pepe.exe
Fallo en la apertura del proceso

Nombre programa: 2-3prog1.exe
Apertura del proceso OK
```

`pepe.exe` es un programa inexistente y por eso se indica fallo en la apertura. `2-3prog1.exe` sí es un programa disponible (es el programa que has hecho antes en esta sesión) y será ejecutado con éxito.

Un aspecto muy importante del programa `2-3prog2.c` es que los procesos que se crean a partir de él crean su propia ventana de tipo consola. Esto se consigue pasando el *flag* CREATE_NEW_CONSOLE en el campo *fdwCreate* de *CreateProcess()*. Por tanto, cuando un proceso se abra con éxito, se abrirá una nueva consola Win32 en la que el proceso realizará sus operaciones de E/S, y cuando el proceso termine, su consola Win32 se destruirá automáticamente. Es muy importante que observes este comportamiento para comprobar que las cosas están funcionando correctamente.

H Completa el programa `2-3prog2.c` y obtén su correspondiente ejecutable.

Ahora vamos a probarlo lanzando desde él diversos programas, pero para esto necesitamos tener diversos programas disponibles. Lo más cómodo es tener dichos programas en la misma carpeta en la que se encuentra `2-3prog2.exe`, así cuando éste nos pida por pantalla el programa que queremos ejecutar, sólo será necesario indicar su nombre. Si el programa que queremos ejecutar está en otra carpeta, tendríamos que indicar la ruta completa en la que se encuentra su correspondiente fichero.

Ahora mismo, `2-3prog2.exe` se encuentra en la carpeta *Debug* del proyecto `2-3prog2`. Entonces vamos a colocar en esta misma carpeta otros programas ejecutables para que sean lanzados por `2-3prog2.exe`.

H Anteriormente has generado el programa `2-3prog1.exe`, que ejecuta dos bucles anidados durante mucho tiempo. Copia este ejecutable en la carpeta *Debug* del proyecto `2-3prog2`.

H En la carpeta de la asignatura en *Archivos-de-Practicas*, encontrarás dos programas ejecutables, llamados `2-3pruba1.exe` y `2-3prueba2.exe`. Copia estos dos programas a la carpeta *Debug* del proyecto `2-3prog2`.

El cometido de estos programas no es en realidad importante para esta práctica, ya que lo que único que nos interesa es disponer de varios ejecutables, que nos sirvan para probar nuestro programa `2-3prog2.exe`, cuyo objetivo es precisamente lanzar otros ejecutables. No obstante, analizaremos primero estos ejecutables durante un instante, antes de usarlos como elementos de prueba de `2-3prog2.exe`.

H `2-3prueba1.exe` pide por consola un número entero y escribe en la consola el día de la semana correspondiente a dicho número. Los números asignados a los días de la semana están entre 1 y 7. Si se le pasa al programa otro número, éste imprime en la consola un mensaje de error. Este programa se ejecuta en bucle infinito, pero se puede romper su ejecución pulsando `Ctrl-C`. Si quieres mirar también su

código fuente, lo tienes también en la carpeta de la asignatura. Ejecuta este programa haciendo doble clic sobre su icono y comprueba su funcionamiento. Entonces termínalo pulsando `Ctrl-C`.

H `2-3prueba2.exe` pide por consola dos números enteros y escribe en la consola la división entera de éstos. Al igual que el programa anterior, `2-3prueba2.exe` se ejecuta en bucle infinito, que puede romperse pulsando `Ctrl-C`. Su código fuente está también disponible en la carpeta de la asignatura. Ejecuta este programa haciendo doble clic sobre su icono y comprueba su funcionamiento. Entonces termínalo pulsando `Ctrl-C`.

Ahora mismo, en la carpeta *Debug* del proyecto `2-3prog2` tienes disponible (además de `2-3prog2.exe`, que es el programa que queremos probar) `2-3prog1.exe`, `2-3prueba1.exe` y `2-3prueba2.exe`.

H Ejecuta `2-3prog2.exe`, puedes hacerlo haciendo doble clic sobre su icono. Piensa que esto significa que EXPLORER.EXE está lanzando tu programa. Ahora la idea es usar `2-3prog2.exe`, en vez de EXPLORER.EXE, para poner otros programas en ejecución. Primero vas a probar cómo falla al intentar ejecutar un programa inexistente. Pídele que ejecute el programa `pepe.exe`. Comprueba que falla. Ahora pídele que ejecute `2-3prog1.exe`. Observa cómo se abre una nueva consola Win32 asociada a este programa. También verás que no hay E/S en esa consola, porque `2-3prog1.exe` no hace ninguna operación de E/S. Si esto ha funcionado, ¡en hora buena! Acabas de lanzar tu primer proceso en Windows. Ahora tienes simultáneamente en ejecución `2-3prog1.exe` y `2-3prog2.exe`. Para elegir con cuál quieres interaccionar, tienes que pulsar sobre la consola Win32 asociada a uno de ellos. Pulsa sobre la consola de `2-3prog2.exe` para seguir lanzando nuevos procesos. Ahora pídele que ejecute `2-3prueba1.exe`. Comprueba que se genera una nueva consola Win32. Vuelve a seleccionar la consola de `2-3prog2.exe` y lanza el programa `2-3prueba2.exe`. Ahora conmutando entre las diferentes consolas puedes trabajar con el programa que quieras. Haz esto, vete cambiando de consola y trabaja con cada programa, pero no finalices ninguno de ellos. La CPU se está repartiendo entre ellos para que todos se puedan ejecutar. Abre el *Administrador de tareas* y comprueba que todos se encuentran en ejecución. ¿Cuál de ellos está consumiendo más CPU?

¿Cuánta CPU consumen los otros programas que tienes en ejecución (`2-3prog2.exe`, `2-3prueba1.exe` y `2-3prueba2.exe`)?

Aunque, a tenor de lo observado en el *Administrador de tareas*, tu respuesta haya sido 0, el consumo real no será exactamente 0, pero será tan pequeño que puede considerarse despreciable. La conclusión es que los programas que hacen mucha E/S a través de la consola, consumen muy poca CPU.

Finalmente, termina los diversos programas que tienes en ejecución. Observa cómo al terminarlos desaparecen las consolas asociadas a los procesos.

La estructura STARTUPINFO

Mediante la estructura STARTUPINFO podemos controlar diversos aspectos acerca de cómo se inicia un proceso. Vamos a probar dos de estos aspectos: el tamaño de la consola Win32 asociada al proceso y el color del fondo y de la letra utilizados en dicha consola.

H Crea un nuevo proyecto llamado `2-3prog3`. Configúralo para que el Visual elija la versión de *CreateProcess()* que trabaja con cadenas ASCII. Agrégale el fichero `2-3prog3.c` y copia en él el contenido del programa `2-3prog2.c`. Realizaremos las modificaciones necesarias en el nuevo fichero.

La primera modificación a realizar es el tamaño de la ventana asociada al proceso que se crea. Hasta ahora hemos creado ventanas con el tamaño por defecto. Para indicar un tamaño determinado de ventana hay que utilizar los miembros *dwXSize* y *dwYSize* de la estructura STARTUPINFO. Los tamaños se indican en *pixels*. Supongamos que en el sistema en el que estás trabajando la resolución de pantalla es de 1024x768. Entonces podemos crear una consola de tamaño 900x600, que será por tanto ligeramente más pequeña que la pantalla, pero bastante mayor que la consola estándar. Antes de hacer ninguna modificación en el programa `2-3prog3.c`, debes buscar ayuda sobre los campos *dwXSize* y *dwYSize* de la estructura STARTUPINFO, y asegurarte que comprendes bien su cometido. **Al leer la ayuda te darás cuenta de que también tendrás que hacer alguna modificación en el campo *dwFlags***.

H Abre la ayuda. Recuerda que se encuentra en el menú *Programas*, después *Microsoft Developer Network* y, finalmente, *MSDN Library para Visual Studio 2005*. Ahora en la ventana de navegación que te aparece en la parte izquierda de la pantalla elige el panel *Contenido*. A partir de él, vas a localizar la información de la ayuda sobre la función *CreateProcess()*. En ella hay un enlace a la página de la ayuda en la que se explica STARTUPINFO. Para alcanzar el punto de la ayuda en el que se encuentra toda la información sobre las funciones de la API Win32 deberás seguir en la jerarquía de temas de ayuda la ruta que se indica a continuación:

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

Ahora puedes elegir el enlace *Functions in alphabetical order*, y en la página que se abre elegir el enlace *C Functions*. Ahora busca *CreateProcess* y ábrela. Entonces posiciónate en la zona de la ayuda en la que se explica el parámetro *lpStartupInfo*. Allí encontrarás un enlace a la estructura STARTUPINFO. Pulsando sobre él llegarás finalmente a la página informativa sobre esta estructura.

H Una vez que tengas claro el uso de los parámetros *dwXSize* y *dwYSize*, haz las modificaciones necesarias para que los procesos lanzados por `2-3prog3.c` se abran con una consola de tamaño 900x600. Obtén el ejecutable de este programa.

H Para probar `2-3prog3.exe`, utilizaremos como elemento de prueba el programa `2-3prueba1.exe`. Para ello, copia este programa en la carpeta *Debug* del proyecto `2-3prog3`.

H Ejecuta `2-3prog3.exe` (por ejemplo, haciendo doble clic sobre su icono) y lanza desde él el programa `2-3prueba1.exe` y comprueba que el tamaño de la consola que se asocia a este programa es el esperado. Entonces finaliza ambos procesos.

La segunda modificación a realizar es el color de fondo (BACKGROUND) y de los caracteres (FOREGROUND) de la ventana. Para ello tendrás que modificar el campo *dwFillAttribute* de la estructura STARTUPINFO.

H Busca en la ayuda la información correspondiente al campo *dwFillAttribute*. Modifica el programa `2-3prog3.c` para que, además de definir una ventana de 900x600, el fondo de la ventana sea de color rojo y los caracteres impresos en la misma, blancos (el blanco lo obtienes combinando mediante el operador '|' los tres colores básicos). Compila y enlaza el programa y ejecútalo. Lanza mediante él el programa `2-3prueba1.exe` y comprueba que el funcionamiento es el esperado. Entonces finaliza ambos procesos.

4 Ejercicios adicionales

E En este ejercicio se propone realizar una modificación simple en el programa `2-3prog3.c`, para trabajar con la estructura PROCESS_INFORMATION. En esta estructura, el sistema operativo escribe información acerca del proceso que se crea, como por ejemplo el identificador de proceso (PID). Busca en la ayuda información sobre la estructura PROCESS_INFORMATION y, en concreto, sobre su miembro *dwProcessId*. Realiza las modificaciones necesarias en el programa `2-3prog3.c` para que en el caso de que la función *CreateProcess()* tenga éxito, muestre el PID del proceso que se acaba de crear. Prueba el programa contrastando los PIDs mostrados por él con los indicados en el *Administrador de tareas*.

E En algunas ocasiones resulta necesario llevar a cabo acciones temporizadamente. Por ejemplo, imagina que quieres enviar un mensaje a la consola cada segundo. Para esto, típicamente se utiliza la función *Sleep()* de la API Win32. El objetivo de esta función es "dormir" (sacar de ejecución) al proceso durante un cierto tiempo, que se le pasa como parámetro a la función. En este ejercicio se propone realizar un programa, llamado `2-3prog4.c`, que envíe a su consola sucesivos mensajes "x Hola mundo...", donde 'x' representa el número de mensaje. Cada mensaje debe imprimirse en una línea diferente y el ritmo de impresión debe ser un mensaje por segundo. El programa debe estructurarse mediante un bucle controlado por la función *_kbhit()*. Esta función retorna TRUE si se ha pulsado una tecla y FALSE, en el caso contrario. El fichero ce cabecera asociado a esta función es `conio.h`. El bucle del programa debe seguir la siguiente estructura:

```
while( ¡_kbhit() )
{
   ... Imprimir mensaje cada segundo
}
```

Siguiendo esta estructura el programa terminará cuando se pulse una tecla.

Antes de escribir el programa lee en la ayuda la información relativa a la función *Sleep()*. Para buscar ayuda sobre esta función, ten en cuenta que pertenece a la categoría de funciones *Process and Threads*.

E Haz una nueva versión del programa anterior en la que sea el usuario el que indique por consola el ritmo de impresión de mensajes. Prueba tu programa con diferentes valores.