Tutoriales MASM 32 bits + RAD ASM del 1 al 35 en Español by iczelion´s

Escrito por: iczelion´s

Traductor al idioma español Weeken

TUTORIAL 1: CONCEPTOS BÁSICOS

Este tutorial asume que el lector sepa cómo utilizar MASM. Si usted no está familiarizado con MASM, descarga win32asm.exe y estudiar el texto dentro del paquete antes de seguir con el tutorial. Buena. Ya está listo. ¡Vamos!

TEORÍA:

Programas de Win32 se ejecutan en modo protegido, que está disponible desde 80.286. Sin embargo, 80.286 ya es historia. Así que sólo tenemos que ocuparnos de 80386 y sus descendientes. Windows se ejecuta cada programa de Win32 en el espacio virtual de separarse. Esto significa que cada programa de Win32 tendrá su propio espacio de direcciones de 4 GB. Sin embargo, esto no significa que cada programa de Win32 tiene 4 GB de memoria física, sólo que el programa puede encargarse de cualquier dirección en ese rango. Windows hará todo lo necesario para hacer que la memoria las referencias del programa válido. Por supuesto, el programa deben cumplir con las normas establecidas por Windows, cosa que hará que el fallo de protección general temido. Cada programa está solo en su espacio de direcciones. Esto está en contraste con la situación en Win16. * Todos los programas Win16 pueden ver unos a otros *. No es así en Win32. Esta característica ayuda a reducir la posibilidad de escribir un programa sobre otro código de programa del / de datos.

Modelo de la memoria también es drásticamente diferente de los viejos tiempos del mundo de 16 bits. Bajo Win32, no tenemos que estar preocupados con el modelo de memoria o los segmentos más! Sólo hay un modelo de memoria: el modelo de memoria plana. No hay más segmentos de 64K. La memoria es un gran espacio continuo de 4 GB. Esto también significa que no tiene que jugar con los registros de segmento. Se puede utilizar cualquier registro de segmento para hacer frente a cualquier punto en el espacio de memoria. Eso es una gran ayuda para los programadores. Esto es lo que hace que la programación de Win32 de montaje es tan fácil como C.

Cuando se programa bajo Win32, usted debe saber algunas reglas importantes. Una de ellas es que, Windows utiliza ESI, EDI, EBP y ebx internamente y no esperar que los valores en los registros para cambiar. Así que recuerde esta primera regla: si se utiliza cualquiera de los cuatro registros en su función de callback, nunca te olvides de restauración antes de devolver el control a Windows. Una función de devolución de llamada es de su propia función que se llama por Windows. El ejemplo obvio es el procedimiento de Windows. Esto no quiere decir que usted no puede utilizar los registros de cuatro, se puede. Sólo asegúrese de restaurar de nuevo antes de pasar el control a Windows.

CONTENIDO:

Aquí está el programa esqueleto. Si usted no entiende alguno de los códigos, no se asuste. Voy a explicar cada uno de ellos más tarde.

```
0.386
. MODELO plana, STDCALL
. DATOS
<Su <datos Inicializado
......
. DATOS?
<Su <datos Sin inicializar
......
Const.
<Su Constants>
......
. CÓDIGO
<label>
<Su <código
.....
finales <label>
```

Eso es todo! Vamos a analizar este programa esqueleto.

0.386

Se trata de una directiva de ensamblador, diciéndole al ensamblador que utiliza el conjunto de instrucciones 80386. También puede utilizar 0.486, 0.586, pero la apuesta más segura es pegarse a 0.386. En realidad, hay dos formas casi idénticas para cada modelo de CPU. .386/.386p, .486/.486p. Esos "p" versiones sólo son necesarios cuando el programa utiliza instrucciones privilegiadas. Instrucciones privilegiadas son las instrucciones reservadas por el sistema de la CPU / operativo en el modo protegido. Sólo pueden ser utilizados por el código privilegiado, tales como los controladores de dispositivo virtual. La mayoría de las veces, el programa funcionará en modo no privilegiado por lo que es seguro de usar versiones no-p.

. MODELO PLANO, STDCALL

. Modelo es una directiva de ensamblador que especifica el modelo de memoria de su programa. Bajo Win32, hay sólo en el modelo, Modelo PLANA.

STDCALL MASM dice acerca de convenciones paso de parámetros. Convención de paso de parámetros especifica el orden de paso de parámetros, de izquierda a derecha o de derecha a izquierda, y también que se va a equilibrar el marco de pila después de la llamada a la función.

Bajo Win16, hay dos tipos de convención de llamada, C y Pascal

C convención de llamada pasa parámetros de derecha a izquierda, es decir, se empuja el parámetro más a la derecha primero. La persona que llama es el responsable de equilibrar el marco de la pila después de la llamada. Por ejemplo, con el fin de llamar a una función llamada foo (int first_param, int second_param, int third_param) en C la convención de los códigos en ensamblador se verá así:

pulse el botón [third_param]; Empuje el tercer parámetro
pulse el botón [second_param], seguido por el segundo
pulse el botón [first_param], y el primer
llamar a foo
añadir sp, 12; La persona que llama equilibre el marco de pila

PASCAL convención de llamada es a la inversa de C convención de llamada. Se pasa parámetros de izquierda a derecha y el destinatario es responsable de la pila de equilibrio después de la llamada.
Win16 adopta PASCAL convenciones, ya que produce códigos más pequeños. C convención es útil cuando no sabes cuántos parámetros serán pasados a la función como en el caso de wsprintf (). En el caso de wsprintf (), la función no tiene manera de determinar de antemano cuántos parámetros se inserta en la pila, por lo que no puede hacer el balance de la pila.
STDCALL es el híbrido de C y de la convención PASCAL. Pasa parámetros de derecha a izquierda, pero el destinatario es responsable para el equilibrio de la pila después de que el uso de la plataforma call.Win32 STDCALL exclusivamente. Excepto en un caso: wsprintf (). Debe utilizar la convención de llamada de C., con wsprintf ().

. DATOS
. DATOS?
Const.
. CÓDIGO

Las cuatro directivas son lo que se llama la sección. Usted no tiene segmentos en Win32, ¿recuerdas? Sin embargo, usted puede dividir su espacio de direcciones completa en secciones lógicas. El inicio de una sección denota el final de la sección anterior. There'are dos grupos de la sección: los datos y el código. Secciones de datos se dividen en 3 categorías:

- **. DATOS Esta sección contiene datos inicializados de su programa.**
- **. DATOS? Esta sección contiene datos no inicializados de su programa. A veces sólo quiero asignar previamente parte de la memoria, pero no quiere que lo inicie. Esta sección es para ese propósito. La ventaja de los datos sin inicializar es: que no tiene espacio en el archivo ejecutable. Por ejemplo, si asigna 10.000 bytes en el archivo. DATOS? sección, el ejecutable no está hinchado hasta 10.000 bytes. Su tamaño se mantiene prácticamente la misma. Sólo decirle al ensamblador cuánto espacio usted necesita cuando el programa se carga en memoria, eso es todo.**
- **Const. Esta sección contiene la declaración de constantes utilizadas por el programa. Constantes en esta sección no se puede modificar en su programa. Son simplemente * constante *.**

Usted no tiene que usar las tres secciones en su programa. Declare sólo la sección (s) que desea utilizar.

Sólo hay una sección de código:. CÓDIGO. Aquí es donde residen sus códigos.
<label>
finales <label>

donde <label> es cualquier etiqueta arbitraria se utiliza para especificar el alcance de su código. Ambas etiquetas deben ser idénticos. Todos los códigos deben residir entre <label> y <label> final

---

TUTORIAL 2: MESSAGEBOX

En este tutorial, vamos a crear un programa de Windows completamente funcional que muestra un cuadro de mensaje que dice "Win32 conjunto es genial!".

.

TEORÍA:

Windows se prepara una gran cantidad de recursos para los programas de Windows. Para ello, es la API de Windows (Application Programming Interface). API de Windows es una colección enorme de funciones muy útiles que residen en el propio Windows, listo para su uso por cualquier programa de Windows. Estas funciones se almacenan en varios dinámicos vinculados a las bibliotecas (DLL) como kernel32.dll, user32.dll y gdi32.dll. Kernel32.dll contiene funciones de la API que tienen que ver con la memoria y la gestión de procesos. User32.dll controla los aspectos de la interfaz de usuario de su programa. Gdi32.dll es responsable de las operaciones de gráficos. Con excepción de "los tres principales", existen otros archivos DLL que el programa puede utilizar, siempre que disponga de suficiente información sobre las funciones de la API deseados.

Los programas de Windows de forma dinámica vincular estos archivos DLL, es decir. los códigos de funciones de la API no están incluidos en el archivo ejecutable de Windows del programa. Para que su programa para saber dónde encontrar las funciones de la API en tiempo de ejecución deseados, usted tiene que incorporar esa información en el archivo ejecutable. La información se encuentra en las bibliotecas de importación. Usted debe vincular sus programas con las bibliotecas de importación correctas o no será capaz de localizar las funciones de la API.

Cuando un programa de Windows se carga en memoria, Windows lee la información almacenada en el programa. Esta información incluye los nombres de las funciones que el programa utiliza los archivos DLL y las funciones residen pulg Cuando Windows encuentra información como en el programa, que va a cargar los archivos DLL y realizar composturas de función de dirección en el programa por lo que las llamadas se transfiere el control a la derecha función.

Hay dos funciones de la API categoriesof: Uno para el ANSI y el otro para Unicode. Los nombres de las funciones de la API de ANSI se postfixed con "A", por ejemplo. MessageBoxA. Aquellos para Unicode se postfixed con "W" (por Wide Char, creo). Windows 95 soporta de forma nativa ANSI y Unicode de Windows NT.

Generalmente estamos familiarizados con las cadenas ANSI, que son arreglos de caracteres terminados en NULL. Carácter ANSI es de 1 byte de tamaño. Mientras que el código ANSI es suficiente para los idiomas europeos, que no puede manejar varios idiomas orientales que tienen varios miles de caracteres únicos. Es por eso que entra en juego UNICODE Un carácter Unicode es de 2 bytes de tamaño, por lo que es posible tener 65536 caracteres únicos en las cuerdas.

Pero la mayoría de las veces, se utiliza un archivo de inclusión que puede determinar y seleccionar las funciones de la API apropiado para su plataforma. Sólo se refieren a nombres de las funciones del API, sin el sufijo.

EJEMPLO:

Voy a presentar el esqueleto del programa desnuda a continuación. Vamos a la carne que más tarde.

```
0.386
. Modelo plano, stdcall
. Datos
. Código
empezar:
poner fin a empezar a
```

La ejecución se inicia desde la primera instrucción inmediatamente debajo de la etiqueta especificada después de la directiva de final. En el esqueleto de lo anterior, la ejecución se iniciará en la primera instrucción inmediatamente por debajo de empezar a etiqueta. La ejecución continuará la instrucción por instrucción hasta algunas instrucciones de control de flujo, tales como jmp, jne, je, etc ret se encuentra. Esas instrucciones redirigir el flujo de ejecución de otras instrucciones. Cuando el programa necesita para salir a Windows, se debe llamar a una función de la API, ExitProcess.

ExitProcess proto uExitCode: DWORD

La línea de arriba se llama un prototipo de función. Un prototipo de la función define los atributos de una función para el ensamblador / enlazador para que pueda hacer el tipo de comprobación para usted. El formato de un prototipo de función es la siguiente:

FunctionName PROTO [ParameterName]: Tipo de datos, [ParameterName]: Tipo de datos, ...

En resumen, el nombre de la función seguido de la palabra clave PROTO y luego por la lista de tipos de datos de los parámetros, separados por comas. En el ejemplo ExitProcess anterior, se define como una función ExitProcess que tiene sólo un parámetro de tipo DWORD. Prototipos de funciones son muy útiles cuando se utiliza la sintaxis de llamada de alto nivel, invocar. Usted puede pensar en invocar como una simple llamada con el tipo de control. Por ejemplo, si usted lo hace:

llamada ExitProcess

sin forzar un valor DWORD en la pila, el enlazador ensamblador / no será capaz de atrapar ese error para ti. Usted lo notará más adelante cuando se bloquea el programa. Pero si se utiliza:

invocar ExitProcess

El enlazador le informará que usted se olvidó de apretar un dword en la pila por lo tanto evitar el error. Le recomiendo que use en lugar de invocar simple llamada. La sintaxis de invocar es como sigue:

INVOKE expresión [, argumentos]

expresión puede ser el nombre de una función o puede ser un puntero de función. Los parámetros de la función se separan con comas.

La mayoría de los prototipos de función para funciones de la API se mantienen en los archivos de inclusión. Si utiliza MASM32 Hutch, que estará en la carpeta MASM32/include. El incluir los archivos tienen la extensión. Inc y los prototipos de las funciones para las funciones en una

DLL se almacena en el archivo. Inc, con el mismo nombre que el archivo DLL. Por ejemplo, ExitProcess es exportado por kernel32.lib por lo que el prototipo de función para ExitProcess se almacena en kernel32.inc.
También puede crear prototipos de las funciones de sus propias funciones.
A lo largo de mis ejemplos, voy a utilizar windows.inc cabina, que se puede descargar desde http://win32asm.cjb.net

Ahora, de vuelta a ExitProcess, el parámetro uExitCode es el valor que desea que el programa para volver a Windows después de que el programa termina. Usted puede llamar a ExitProcess de esta manera:

invocar ExitProcess, 0

Ponga esa línea inmediatamente por debajo de empezar a etiqueta, usted recibirá un programa win32 que inmediatamente sale a Windows, pero es un programa válido, no obstante.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib
. Datos
. Código
empezar:
invocar ExitProcess, 0
poner fin a empezar a
```

casemap opción: no le dice a MASM para hacer etiquetas de mayúsculas y minúsculas para ExitProcess y ExitProcess son diferentes Tenga en cuenta una nueva directiva, se incluyen.. Esta directiva es seguido por el nombre de un archivo que desea insertar en el lugar de la directiva es. En el ejemplo anterior, cuando MASM procesa la línea include \ masm32 \ include \ windows.inc, se abrirá windows.inc que se encuentra en \ MASM32 \ include carpeta y procesar el contenido de windows.inc como si pegar el contenido de las ventanas . inc allí. windows.inc cabina contiene definiciones de constantes y estructuras necesarias en la programación de Win32. No contiene ningún prototipo de función. windows.inc no es en absoluto exhaustiva. Hutch y yo trato de poner como constantes y de estructuras dentro de ella como sea posible, pero todavía hay muchos dejaron de ser incluidos. Va a ser constantemente actualizado. Echa un vistazo a la conejera y mi página de inicio para las actualizaciones.
Desde windows.inc, el programa tiene las definiciones de constantes y de estructuras. Ahora, para prototipos de funciones, es necesario incluir otros archivos de inclusión. Ellos se almacenan en \ masm32 carpeta \ include.

En nuestro ejemplo anterior, llamamos a una función exportada por kernel32.dll, así que tenemos que incluir los prototipos de funciones profesionales de kernel32.dll. Ese archivo es kernel32.inc. Si lo abres con un editor de texto, verás que está lleno de prototipos de las funciones de kernel32.dll. Si usted no incluye kernel32.inc, puede llamar a ExitProcess, pero sólo con una sintaxis simple llamada. Usted no será capaz de invocar la función. El punto aquí es que: con el fin de invocar una función, tienes que poner de su prototipo de función en alguna parte del código fuente. En el ejemplo anterior, si no se incluye kernel32.inc, se puede definir el prototipo de función para ExitProcess en cualquier parte del código fuente de arriba el comando invocar y va a trabajar. Los archivos incluyen están ahí para salvar el trabajo de escribir los prototipos de sí mismo para usarlos cada vez que pueda.
Ahora nos encontramos con una nueva directiva, includelib. includelib no funciona como se incluyen. Es 's sólo una manera de saber lo que el ensamblador biblioteca de importación el programa utiliza. Cuando el ensamblador considera que una directiva includelib, pone un comando de enlace en el archivo de objeto de modo que el enlazador sabe lo que las bibliotecas de importación de su programa tiene que enlazar con. Usted no está obligado a

utilizar includelib sin embargo. Usted puede especificar los nombres de las bibliotecas de importación en la línea de comandos del vinculador, pero créanme, es tedioso y la línea de comandos puede contener sólo 128 caracteres.

Ahora, guarde el ejemplo bajo el nombre de msgbox.asm. Suponiendo que ml.exe está en su ruta, montar msgbox.asm con:

ml / c / coff / Cp msgbox.asm

- / C dice MASM para ensamblar solamente. No invoque link.exe. La mayoría de las veces, usted no quiere llamar a link.exe de forma automática, ya que puede que tenga que realizar algunas otras tareas antes de llamar a link.exe.
  / Coff dice MASM para crear. Obj en formato COFF. MASM utiliza una variación de COFF (formato de archivo de objeto común) que se utiliza en Unix como su propio objeto y el formato de archivo ejecutable.
  / Cp dice MASM para preservar caso de los identificadores de usuario. Si usa el paquete MASM32 cabina, es posible poner "casemap opción: ninguno" a la cabeza de su código fuente, justo debajo de modelo de directiva para lograr el mismo efecto..

Después de reunirse con éxito msgbox.asm, obtendrá msgbox.obj. msgbox.obj es un archivo objeto. Un archivo de objeto es sólo un paso de un archivo ejecutable. Contiene las instrucciones o datos en forma binaria. Lo que falta es algunas composturas de los discursos pronunciados por el enlazador.

A continuación, seguir con el enlace:

link / SUBSYSTEM: WINDOWS / LIBPATH: c: \ masm32 \ lib msgbox.obj

/ SUBSYSTEM: WINDOWS Enlace informa qué tipo de ejecutable de este programa es
/ LIBPATH: <ruta importar library> le dice a Link en las bibliotecas de importación. Si utiliza MASM32, que estará en MASM32 \ lib carpeta.

Enlace lee en el archivo de objeto y lo arregla con las direcciones de las bibliotecas de importación. Cuando termine el proceso se obtiene msgbox.exe.

Ahora usted consigue msgbox.exe. Vamos, lo ejecute. Usted encontrará que no hace nada. Bueno, no hemos puesto nada interesante en ella todavía. Pero es un programa de Windows, no obstante. Y mira a su tamaño! En mi PC, que es 1.536 bytes.

A continuación vamos a poner en un cuadro de mensaje. Su prototipo de función es:

MessageBox PROTO hwnd: DWORD, lpText: DWORD, lpCaption: DWORD, uType: DWORD

hwnd es el handle a la ventana principal. Usted puede pensar en un mango como un número que representa la ventana que está a la referrring. Su valor no es importante para usted. Sólo recuerde que representa la ventana. Cuando usted quiere hacer cualquier cosa con la ventana, usted debe referirse a ella por el mango.
lpText es un puntero al texto que desea mostrar en el área de cliente del cuadro de mensaje. Un puntero es realmente una dirección de algo. Un puntero a la cadena de texto == La dirección de esa cadena.
lpCaption es un puntero a la leyenda del cuadro de mensaje
uType especifica en el icono de los botones y el número y el tipo de en el cuadro de mensaje

Vamos a modificar msgbox.asm para incluir el cuadro de mensaje.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ user32.lib

. Datos
MsgBoxCaption db "Iczelion Tutorial N º 2", 0
MsgBoxText db "Win32 Asamblea es genial!", 0

. Código
empezar:
invocar el cuadro de mensaje, NULL, MsgBoxText addr, addr MsgBoxCaption, MB_OK
invocar ExitProcess, NULL
poner fin a empezar a
```

Ensamble y ejecutarlo. Usted verá un cuadro de mensaje muestra el texto "Asamblea de Win32 es genial!".

Vamos a ver de nuevo en el código fuente.
Se definen dos cadenas terminadas en cero en. Sección de datos. Recuerde que cada cadena ANSI en Windows debe ser resuelto por NULL (0 hexadecimal).
Utilizamos dos constantes, nula y MB_OK. Estas constantes se documentan en windows.inc. Así que usted puede hacer referencia a ellos por su nombre en lugar de los valores. Esto mejora la legibilidad del código fuente.
El operador addr se utiliza para pasar la dirección de una etiqueta a la función. Es válido sólo en el contexto de invocar a la Directiva. No se puede utilizar para asignar la dirección de una etiqueta a un registro / variable, por ejemplo. Se puede utilizar en lugar de compensar addr en el ejemplo anterior. Sin embargo, hay algunas diferencias entre los dos:

1. addr no puede manejar referencia adelantada, mientras que compensar puede. Por ejemplo, si la etiqueta se define en algún lugar aún más en el código fuente de la línea de invocar, addr no funcionará.

   ```
   invocar el cuadro de mensaje, NULL, MsgBoxText addr, addr MsgBoxCaption, MB_OK
   .....
   MsgBoxCaption db "Iczelion Tutorial N º 2", 0
   MsgBoxText db "Win32 Asamblea es genial!", 0
   ```

   MASM informe de errores. Si se utiliza compensado en lugar de addr en el fragmento de código anterior, MASM se reúnen felizmente.

2. addr puede manejar las variables locales al mismo tiempo compensado no pueden. Una variable local es sólo un espacio reservado en la pila. Sólo se conoce su dirección durante el tiempo de ejecución. Desplazamiento se interpreta durante el tiempo de montaje por el ensamblador. Así que es natural que el offset no va a funcionar para las variables locales. Addr es capaz de manejar las variables locales por el hecho de que el ensamblador comprueba en primer lugar si la variable referida por addr es global o local. Si se trata de una variable global, que pone la dirección de esa variable en el archivo objeto. En este sentido, funciona como compensación. Si se trata de una variable local, que genera una secuencia de instrucción como esta antes de que realmente llama a la función:

```
lea eax, LocalVar
push eax
```

Dado que lea puede determinar la dirección de una etiqueta en tiempo de ejecución, esto funciona muy bien.

TUTORIAL 3: UNA SIMPLE VENTANA

En este tutorial, vamos a construir un programa de Windows que muestra una ventana completamente funcional en el escritorio.

TEORÍA:

Los programas de Windows dependen en gran medida las funciones de la API para su interfaz gráfica de usuario. Este enfoque beneficia a los usuarios y programadores. Para los usuarios, no tienen que aprender a navegar por la interfaz gráfica de usuario de cada uno de los nuevos programas, la interfaz gráfica de usuario de programas de Windows son iguales. Para los programadores, los códigos de interfaz gráfica de usuario ya están allí, probados y listos para su uso. La desventaja para los programadores es el aumento de la complejidad involucrada. Con el fin de crear o manipular cualquier objeto de interfaz gráfica de usuario, tales como ventanas, menús o iconos, los programadores deben seguir una receta estricta. Pero que se puede superar mediante la programación modular o paradigma de programación orientada a objetos.
Voy a esbozar los pasos necesarios para crear una ventana en el escritorio a continuación:

1. Obtener el identificador de instancia de su programa (requerido)
2. Conseguir la línea de comandos (no es necesario a menos que su programa quiere procesar una línea de comandos)
3. Registrarse clase de ventana (se requiere, a menos que use tipos predefinidos de la ventana, por ejemplo. MessageBox o un cuadro de diálogo)
4. Crear la ventana (se requiere)
5. Mostrar la ventana en el escritorio (requiere a menos que usted no desea que aparezca la ventana de inmediato)
6. Actualizar el área de cliente de la ventana
7. Escriba un bucle infinito, comprobar si hay mensajes de Windows
8. Si llegan los mensajes, que son procesados por una función especializada que se encarga de la ventana
9. Salir del programa si el usuario cierra la ventana

Como puede ver, la estructura de un programa de Windows es bastante complejo en comparación con un programa de DOS. Pero el mundo de Windows es radicalmente diferente del mundo de la DOS. Los programas de Windows debe ser capaz de coexistir pacíficamente con los demás. Deben cumplir con normas más estrictas. Usted, como programador, también debe ser más estricta con su estilo de programación y el hábito.

CONTENIDO:

A continuación se muestra el código fuente de nuestro programa simple ventana. Antes de saltar en los detalles escabrosos de la programación de Win32 ASM, voy a señalar algunos puntos finos que facilitan su programación.

- Usted debe poner todas las constantes de Windows, las estructuras y los prototipos de funciones en un archivo de inclusión e incluirlo en el comienzo de su archivo. Asm. Le ahorrará un montón de esfuerzo y error tipográfico. En la actualidad, el más completo archivo de inclusión para MASM es windows.inc cabina, que se puede descargar desde su página o en mi página. También puede definir sus propias constantes y definiciones de la estructura, pero usted debe ponerlos en un archivo separado de inclusión.
- Utilice Directiva includelib para especificar la biblioteca de importación utilizados en su programa. Por ejemplo, si su programa llama de mensajes, usted debe poner la línea:

  user32.lib includelib

  al comienzo de su archivo. asm. Esta directiva indica a MASM que su programa hará uso de las funciones en la biblioteca de importación. Si su programa llama a funciones en más de una biblioteca, basta con añadir un includelib para cada biblioteca que utiliza. Uso de la Directiva IncludeLib, usted no tiene que preocuparse acerca de las bibliotecas de importación en tiempo de enlace. Usted puede utilizar / LIBPATH interruptor enlazador decir Enlace en todas las librerías son.

- Al declarar prototipos de funciones API, las estructuras, o constantes en el archivo de inclusión, trate de cumplir con los nombres originales utilizados en Windows incluyen los archivos, incluyendo el caso. Esto le ahorrará muchos dolores de cabeza cuando se esté buscando algún artículo en referencia de la API de Win32.
- Utilice makefile para automatizar su proceso de montaje. Esto le ahorrará un montón de escribir.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ user32.lib; las llamadas a funciones en user32.lib y kernel32.lib
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

Los datos, datos inicializados
ClassName db "SimpleWinClass", 0; el nombre de nuestra clase de ventana
AppName db "Nuestra Primera Ventana", 0; el nombre de nuestra ventana

Los datos, ¿los datos no inicializados
HINSTANCE hInstance;? identificador de instancia de nuestro programa
LPSTR de línea de comandos?
. Código; Aquí comienza nuestro código
empezar:
invocar GetModuleHandle, NULL; obtener el identificador de instancia de nuestro programa.
                                        ; Bajo Win32, hModule == hinstance hInstance mov, eax
mov hInstance, eax
invocar GetCommandLine; obtener la línea de comandos. Usted no tiene que llamar a esta
función SI
```

```
                                                , Su programa no procesa la línea de comandos.
 CommandLine mov, eax
 WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT, llame a la
función principal
 invocar ExitProcess, eax; salir de nuestro programa.  El código de salida se devuelve en eax de
WinMain.

 WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
 LOCAL wc: WNDCLASSEX, crear variables locales en la pila
 Msg LOCAL: MSG
 LOCAL hwnd: HWND

 mov wc.cbSize, sizeof WNDCLASSEX; rellenar los valores de los miembros de wc
 mov wc.style, CS_HREDRAW o CS_VREDRAW
 mov wc.lpfnWndProc, OFFSET WndProc
 mov wc.cbClsExtra, NULL
 mov wc.cbWndExtra, NULL
 impulsar hInstance
 pop wc.hInstance
 mov wc.hbrBackground, COLOR_WINDOW un
 mov wc.lpszMenuName, NULL
 wc.lpszClassName mov, ClassName OFFSET
 invocar LoadIcon, NULL, IDI_APPLICATION
 mov wc.hIcon, eax
 mov wc.hIconSm, eax
 invocar LoadCursor, NULL, IDC_ARROW
 wc.hCursor mov, eax
 invocar RegisterClassEx, addr wc, registrar nuestra clase de ventana
 invocar CreateWindowEx, NULL, \
 ADDR ClassName, \
 ADDR AppName, \
 WS_OVERLAPPEDWINDOW, \
 CW_USEDEFAULT, \
 CW_USEDEFAULT, \
 CW_USEDEFAULT, \
 CW_USEDEFAULT, \
 NULL, \
 NULL, \
 hInst, \
 NULL
 mov hwnd, eax
 invocar ShowWindow, hwnd, CmdShow, mostrar nuestra ventana en el escritorio
 invocar UpdateWindow, hwnd; actualización del área de cliente

 Al tiempo que TRUE; Introduzca bucle de mensajes
 invocar GetMessage, msg ADDR, NULL, 0,0
 . INTER. If (! Eax)
 invocar TranslateMessage, ADDR msg
 invocar DispatchMessage, ADDR msg
 . ENDW
 mov eax, msg.wParam; código de retorno de salida en eax
 ret
 WinMain endp

 WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
 . SI uMsg == WM_DESTROY, si el usuario cierra la ventana
 invocar PostQuitMessage, NULL; salir de nuestra aplicación
 . MÁS
 invocar DefWindowProc, hWnd, uMsg, wParam, lParam; procesamiento de mensajes por defecto
```

ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

poner fin a empezar a

ANÁLISIS:

Usted puede ser sorprendido de que un simple programa de Windows requiere una codificación tanto. Pero la mayoría de esos códigos son sólo * Plantilla * Los códigos que se pueden copiar de un archivo de código fuente a otra. O si lo prefiere, puede reunir algunos de estos códigos en una biblioteca para ser utilizado como prólogo y epílogo códigos. Usted puede escribir sólo los códigos de la función WinMain. De hecho, esto es lo que hacen los compiladores de C. Ellos le permiten escribir códigos WinMain sin preocuparse por las demás labores domésticas. El único inconveniente es que debe tener una función llamada WinMain demás compiladores de C no será capaz de combinar los códigos con el prólogo y el epílogo. Usted no tiene restricción de este tipo con el lenguaje ensamblador. Puede utilizar cualquier nombre de función en lugar de WinMain o no funciona en absoluto.
Prepárese. Este va a ser un tutorial mucho, mucho tiempo. Vamos a analizar este programa hasta la muerte!

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
```

Las tres primeras líneas son "necesidades". 0.386 MASM dice que va a utilizar 80.386 conjunto de instrucciones en este programa. . Modelo plano, stdcall dice MASM que nuestro programa utiliza la memoria de direccionamiento plano modelo. También vamos a utilizar el parámetro stdcall pasando convención como el predeterminado en nuestro programa.
El siguiente es el prototipo de función para WinMain. Dado que vamos a llamar a WinMain más tarde, tenemos que definir su prototipo de función primero, para que podamos invocarlo.
Debemos incluir windows.inc al comienzo del código fuente. Contiene estructuras importantes y constantes que se utilizan en nuestro programa. El archivo de inclusión, windows.inc, es sólo un archivo de texto. Usted puede abrir con cualquier editor de texto. Tenga en cuenta que windows.inc no contiene todas las estructuras y constantes (todavía). Hutch y yo estamos trabajando en ello. Usted puede agregar nuevos elementos en el caso de que no se encuentran en el archivo.
Nuestro programa llama a funciones de la API que residen en user32.dll (CreateWindowEx, RegisterWindowClassEx, por ejemplo) y kernel32.dll (ExitProcess), por lo que debemos vincular nuestro programa a las dos bibliotecas de importación. La siguiente pregunta: ¿cómo puedo saber qué biblioteca de importación debe estar vinculado a mi programa? La respuesta: Usted debe saber dónde están las funciones de la API llamados por su programa de residencia. Por ejemplo, si usted llama a una función de la API en gdi32.dll, debe enlazar con gdi32.lib.

Este es el enfoque de MASM. Manera de TASM @ s de biblioteca de importación que une es mucho más sencillo: basta con enlace a uno y sólo un archivo: import32.lib.

```
. DATOS
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0

. DATOS?
HINSTANCE hInstance?
LPSTR de línea de comandos?
```

A continuación se muestran las secciones "DATA".
En DATOS., Declaramos terminada en cero dos cadenas (ASCIIZ): classname que es el nombre de nuestra clase de ventana y AppName que es el nombre de nuestra ventana. Tenga en cuenta que las dos variables se inicializan.
.? En DATOS, se declaran dos variables: hInstance (identificador de instancia de nuestro programa) y de línea de comandos (línea de comandos de nuestro programa). Los tipos de datos desconocidos, hInstance y LPSTR, son realmente nombres de nuevas versiones de DWORD. Usted puede buscar en windows.inc. Tenga en cuenta que todas las variables. Datos? sección no se inicializan, es decir, que no tiene que tener cualquier valor específico en el arranque, pero queremos reservar el espacio para uso futuro.

```
. CÓDIGO
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax
.....
poner fin a empezar a
```

. CÓDIGO contiene todas sus instrucciones. Los códigos deben residir entre <Arranque label>: y label> <Arranque final. El nombre de la etiqueta no es importante. Puede que sea lo que quiera con tal de que es única y no viola la convención de nombres de MASM.
Nuestra primera instrucción es la llamada a GetModuleHandle para recuperar el identificador de instancia de nuestro programa. Bajo Win32, identificador de instancia y el identificador de módulo son una y la misma. Usted puede pensar en identificador de instancia como el ID de su programa. Se utiliza como parámetro para varias funciones de la API debe llamar a nuestro programa, por lo que es generalmente una buena idea para recuperarlo al comienzo de nuestro programa.
Nota: En realidad, bajo Win32, identificador de instancia es la dirección lineal de su programa en la memoria.
A su regreso de una función de Win32, el valor devuelto por la función, en su caso, se puede encontrar en eax. Todos los demás valores se devuelven a través de variables pasadas en la lista de parámetros de función que se define por la llamada.
Una de las funciones de Win32 que se llama casi siempre preservar los registros de segmento y de los registros EBX, EDI, ESI y EBP. Por el contrario, ecx y edx se consideran registros de rasca y gana y siempre están sin definir a su regreso de una función de Win32.
Nota: No espere que los valores de EAX, ECX, EDX que se preserva a través de llamadas a funciones API.

La conclusión es que: cuando se llama a una función de la API, espera que el valor de retorno en eax. Si cualquier parte de su función debe ser llamada por Windows, también debe jugar por la regla: preservar y restaurar los valores de los registros de segmento, ebx, edi, esi y ebp a su regreso función de otra cosa que su programa se colgará en breve, esto incluye su procedimiento de ventana y ventanas de las funciones de devolución de llamada.

La llamada GetCommandLine no es necesario si el programa no procesa una línea de comandos. En este ejemplo, yo te mostraré cómo lo llaman en caso de que necesite en su programa.

El siguiente es el llamado WinMain. Aquí recibe cuatro parámetros: el identificador de instancia de nuestro programa, el identificador de instancia de la instancia anterior de nuestro programa, la línea de comandos y el estado de la ventana en la primera comparecencia. Bajo Win32, no hay ninguna instancia anterior. Cada programa está solo en su espacio de direcciones, por lo que el valor de hPrevInst es siempre 0. Se trata de un residuo desde el día de Win16 cuando todas las instancias de un programa que se ejecuta en el mismo espacio de direcciones y la instancia de uno quiere saber si es la primera instancia. Bajo Win16, si hPrevInst es NULL, este caso es el primero.

Nota: Usted no tiene que declarar el nombre de función como WinMain. De hecho, usted tiene total libertad en este sentido. Usted no tiene que utilizar cualquier función WinMain equivalente en absoluto. Puede pegar los códigos dentro de la función WinMain junto a GetCommandLine y el programa seguirá siendo capaz de funcionar a la perfección.

A su regreso de WinMain, eax está lleno de código de salida. Pasamos de que el código de salida como el parámetro a ExitProcess que termina nuestra aplicación.

WinMain proc Inst.: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow: DWORD

La línea anterior es la declaración de una función WinMain. Tenga en cuenta el parámetro: pares de tipos que siguen directiva PROC. Son parámetros que WinMain recibe de la persona que llama. Puede hacer referencia a estos parámetros por su nombre en lugar de manipulación de la pila. Además, MASM genera el prólogo y el epílogo de códigos para la función. Así que no tenemos que ocuparnos de marco de pila en la función de entrar y salir.

LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND

La directiva LOCAL asigna la memoria de la pila para las variables locales utilizadas en la función. El grupo de directivas locales deben estar inmediatamente debajo de la directiva PROC. La directiva LOCAL es inmediatamente seguido por el nombre de <el variable> local: <tipo variable>. Así LOCAL wc: WNDCLASSEX dice MASM para asignar la memoria de la pila del tamaño de la estructura WNDCLASSEX de la variable llamada wc. Podemos referirnos a la CC en nuestros códigos sin ningún tipo de dificultad en la manipulación de la pila. Eso es realmente un regalo de Dios, creo. La desventaja es que las variables locales no se pueden utilizar fuera de la función que están creados y se destruyen automáticamente cuando la función retorna a la persona que llama. Otro inconveniente es que no se puede inicializar las variables locales de forma automática, ya que son sólo memoria de pila asignado dinámicamente cuando la función se escribe. Usted tiene que asignar manualmente con los valores deseados después de directivas locales.

mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInstance
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL

wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc

Las líneas anteriores inimidating son realmente simples en su concepto. Sólo hace falta varias líneas de instrucción para llevar a cabo. El concepto detrás de todas estas líneas es *la clase de ventana.* Una clase de ventana no es más que un proyecto o una especificación de una ventana. En él se definen varias características importantes de una ventana, como su icono, su cursor, la función de responsable de la misma, su color, etc Para crear una ventana de una clase de ventana. Esta es una especie de concepto orientado a objetos. Si desea crear más de una ventana con las mismas características, es razonable pensar que para almacenar todas estas características en un solo lugar y se refieren a ellos cuando sea necesario. Este esquema le ahorrará gran cantidad de memoria, evitando la duplicación de información. Recuerde que Windows está diseñado en el pasado cuando los chips de memoria son prohibitivos y la mayoría de equipos tienen 1 MB de memoria. Windows debe ser muy eficiente en el uso de los recursos de memoria escasa. El punto es: si se define su propia ventana, usted debe llenar las características deseadas de la ventana en un WNDCLASS o estructura WNDCLASSEX y llame RegisterClass o RegisterClassEx antes de que puedas crear tu ventana. Usted sólo tiene que registrar la clase de ventana de una vez para cada tipo de ventana que desea crear una ventana de.
Windows tiene varias clases de ventanas predefinidas, como botón y un cuadro de edición. Por estas ventanas (o controles), usted no tiene el registro de una clase de ventana, simplemente llame a CreateWindowEx con el nombre de la clase predefinida.
El miembro más importante en el WNDCLASSEX es lpfnWndProc. lpfn representa puntero largo a la función. Bajo Win32, no hay ningún puntero "cerca" o "mucho", sólo el puntero por el modelo de memoria plano nuevo. Pero esto es otra vez un resto del día de Win16. Cada clase de ventana debe estar asociado con una función llamada procedimiento de ventana. El procedimiento de ventana es responsable de la gestión de mensajes de todas las ventanas creadas a partir de la clase de ventana asociada. Windows envía mensajes al procedimiento de ventana que le informe de los acontecimientos importantes en relación con la que las ventanas 's responsables, como el teclado de usuario o la entrada del ratón. Es hasta el procedimiento de ventana para responder inteligentemente a cada mensaje de la ventana que recibe. Usted pasará la mayor parte de su tiempo a escribir controladores de eventos en el procedimiento de ventana.
Describo a cada miembro de WNDCLASSEX a continuación:

WNDCLASSEX STRUCT DWORD
cbSize DWORD?
DWORD estilo?
lpfnWndProc DWORD?
cbClsExtra DWORD?
cbWndExtra DWORD?
hInstance DWORD?
hIcon DWORD?
DWORD hCursor?
hbrBackground DWORD?
lpszMenuName DWORD?
DWORD lpszClassName?
hIconSm DWORD?
WNDCLASSEX TERMINA

cbSize: El tamaño de la estructura WNDCLASSEX en bytes. Podemos usar el operador sizeof a obtener el valor.
Estilo: El estilo de las ventanas creadas a partir de esta clase. Se pueden combinar varios estilos, junto con "o" del operador.
lpfnWndProc: La dirección del procedimiento de ventana responsable de las ventanas creadas a partir de esta clase.
cbClsExtra: Especifica el número de bytes extra se reservarán a raíz de la estructura de la clase. El sistema operativo inicializa los bytes a cero. Puede almacenar las ventanas específicas de la clase de

datos aquí.
cbWndExtra: Especifica el número de bytes extra se reservarán a raíz de la instancia de la ventana. El sistema operativo inicializa los bytes a cero. Si una aplicación utiliza la estructura WNDCLASS para registrar un cuadro de diálogo creado mediante la directiva CLASS en el archivo de recursos, tiene que establezca este miembro en DLGWINDOWEXTRA.
hInstance: identificador de instancia del módulo.
hIcon: identifica el icono. Se puede obtener del llamado LoadIcon.
hCursor: identifica el cursor. Se puede obtener del llamado LoadCursor.
hbrBackground: Color de fondo de las ventanas creadas a partir de la clase.
lpszMenuName: mango por defecto del menú para las ventanas creadas a partir de la clase.
lpszClassName: El nombre de esta clase de ventana.
hIconSm: Controlar a un pequeño icono que se asocia con la clase de ventana. Si este miembro es NULL, el sistema busca el recurso de icono especificado por el miembro hIcon un icono del tamaño adecuado para usarlo como icono pequeño.

```
    invocar CreateWindowEx, NULL, \
ADDR ClassName, \
ADDR AppName, \
WS_OVERLAPPEDWINDOW, \
CW_USEDEFAULT, \
CW_USEDEFAULT, \
CW_USEDEFAULT, \
CW_USEDEFAULT, \
NULL, \
NULL, \
hInst, \
NULL
```

Después de registrar la clase de ventana, podemos llamar a CreateWindowEx para crear nuestra ventana sobre la base de la clase de ventana presentado. Tenga en cuenta que hay 12 parámetros a esta función.

```
CreateWindowExA proto dwExStyle: DWORD, \
lpClassName: DWORD, \
lpWindowName: DWORD, \
dwStyle: DWORD, \
X: DWORD, \
Y: DWORD, \
nWidth: DWORD, \
nHeight: DWORD, \
hwndParent: DWORD, \
hMenu: DWORD, \
hInstance: DWORD, \
lpParam: DWORD
```

Vamos a ver una descripción detallada de cada parámetro:
dwExStyle: estilos adicionales de la ventana. Este es el nuevo parámetro que se agrega a la antigua CreateWindow. Usted puede poner nuevos estilos de ventanas para Windows 95 y NT here.You puede especificar el estilo de ventana normal en dwStyle pero si quieres algunos estilos especiales, tales como ventana de nivel superior, debe especificar aquí. Usted puede utilizar NULL si no desea estilos adicionales de la ventana.
lpClassName: (Obligatorio). Dirección de la cadena ASCIIZ con el nombre de clase de ventana que desea utilizar como plantilla para esta ventana. La clase puede ser su propia clase social o de clase de ventana predefinida. Como se indicó anteriormente, cada ventana que ha creado se debe basar en una clase de ventana.
lpWindowName: Dirección de la cadena ASCIIZ con el nombre de la ventana. Va a ser que aparece en la barra de título de la ventana. Si este parámetro es NULL, la barra de título de la ventana estará en blanco.
dwStyle: Estilos de la ventana. Usted puede especificar el aspecto de la ventana de aquí. Pasar NULL está bien, pero la ventana no tendrá ningún cuadro de menú del sistema, no hay botones de minimizar, maximizar, y no hay botón de cerrar ventana. La ventana no sería de mucha utilidad en absoluto. Usted

tendrá que pulsar Alt + F4 para cerrarla. El estilo de ventana más común es WS_OVERLAPPEDWINDOW. Un estilo de ventana es sólo un indicador de bits. Así, usted puede combinar varios estilos de la ventana por la "u" operador para lograr el aspecto deseado de la ventana. Estilo WS_OVERLAPPEDWINDOW es en realidad una combinación de los estilos de ventana más comunes por este método.

X, Y: La coordenada de la esquina superior izquierda de la ventana. Normalmente, estos valores deben ser CW_USEDEFAULT, es decir, que desea que Windows para decidir por dónde debe poner la ventana en el escritorio.

nWidth, nHeight: La anchura y la altura de la ventana en píxeles. También puede utilizar CW_USEDEFAULT para permitir que Windows elija la anchura y altura apropiada para usted.

hwndParent: un identificador de ventana principal de la ventana (si existe). Este parámetro indica a Windows si la ventana es un niño (subordinada) de alguna otra ventana y, si es que la ventana es el padre. Tenga en cuenta que esta no es la relación padre-hijo de la interfaz de múltiples documentos (MDI). Ventanas de los niños no están obligados al área de cliente de la ventana principal. Esta relación es específicamente para el uso interno de Windows. Si la ventana padre se destruye, todas las ventanas secundarias serán destruidos de forma automática. Es realmente así de simple. Dado que en nuestro ejemplo, sólo hay una ventana, se especifica este parámetro como NULL.

hMenu: un identificador de menú de la ventana. NULL si el menú de la clase se va a utilizar. Mirar hacia atrás en el miembro de la estructura de un WNDCLASSEX, lpszMenuName. lpszMenuName especifica * menú por defecto * para la clase de ventana. Cada ventana creada a partir de esta clase de ventana tendrá el mismo menú por defecto. A menos que especifique un menú * * imperiosas de una ventana específica a través de su parámetro hMenu. hMenu es en realidad un parámetro de doble propósito. En el caso de la ventana que desea crear es de un tipo de ventana predefinida (es decir, el control), dicho control no puede ser dueño de un menú. hMenu se utiliza como identificador de que el control del lugar. Windows puede decidir si hMenu es realmente un mango de menú o un identificador de control al mirar en el parámetro lpClassName. Si es el nombre de una clase de ventana predefinida, hMenu es un identificador de control. Si no es así, entonces es un identificador para el menú de la ventana.

hInstance: El mango ejemplo para el módulo de programa crear la ventana.

lpParam: puntero opcional a una estructura de datos pasa a la ventana. Esto es usado por la ventana MDI para pasar los datos CLIENTCREATESTRUCT. Normalmente, este valor se establece en NULL, que significa que no se pasan los datos a través de CreateWindow (). La ventana se puede recuperar el valor de este parámetro por la llamada a la función GetWindowLong.

```
  mov hwnd, eax
invocar ShowWindow, hwnd, CmdShow
invocar UpdateWindow, hwnd
```

El retorno exitoso de CreateWindowEx, el identificador de ventana se devuelve en eax. Debemos mantener este valor para su uso futuro. La ventana que acabamos de crear, no se muestra automáticamente. Usted debe llamar a ShowWindow con la manija de la ventana y la pantalla deseada * * estado de la ventana para que aparezca en la pantalla. A continuación, puede llamar a UpdateWindow para ordenar la ventana para volver a pintar su área cliente. Esta función es útil cuando se desea actualizar el contenido del área de cliente. Puede omitir esta convocatoria sin embargo.

```
  . MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
```

En este momento, la ventana es en la pantalla. Pero no puede recibir las aportaciones del mundo. Así que tenemos que * Informar * que de los hechos relevantes. Esto lo logramos con un bucle de mensajes. Sólo hay un bucle de mensajes para cada módulo. Este bucle de mensajes comprueba continuamente los mensajes de Windows con llamada GetMessage. GetMessage pasa un puntero a una estructura MSG para Windows. Esta estructura MSG estará lleno de información sobre el mensaje que desea enviar de Windows a una ventana en el módulo. La función GetMessage no regresará hasta que hay un mensaje de una ventana en el módulo. Durante ese tiempo, Windows se puede dar el control a otros programas. Esto es lo que constituye el sistema de multitarea cooperativa de la plataforma Win16. GetMessage

devuelve FALSE si el mensaje se recibe WM_QUIT que, en el bucle de mensajes, se terminará el bucle y salir del programa.
TranslateMessage es una función de utilidad que toma la entrada del teclado en bruto y genera un nuevo mensaje (WM_CHAR) que se coloca en la cola de mensajes. El mensaje con WM_CHAR contiene el valor ASCII de la tecla pulsada, que es más fácil de tratar que los códigos de rastreo de teclado primas. Puede omitir esta convocatoria si su programa no procesa las pulsaciones de teclado.
DispatchMessage envía los datos del mensaje al procedimiento de ventana responsable de la ventana específica es el mensaje.

```
    mov eax, msg.wParam
ret
WinMain endp
```

Si termina el bucle de mensajes, el código de salida se almacena en el miembro wParam de la estructura MSG. Puede guardar este código de salida en eax para volver a Windows. En la actualidad, Windows no hace uso del valor de retorno, pero es mejor estar en el lado seguro y juega por la regla.

```
WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
```

Este es nuestro procedimiento de ventana. Usted no tiene que nombrar que WndProc. El primer parámetro, hWnd, es el identificador de ventana de la ventana que el mensaje está destinado a. uMsg es el mensaje. Tenga en cuenta que uMsg no es una estructura MSG. Es sólo un número, de verdad. Windows define cientos de mensajes, la mayoría de los cuales sus programas no se interese Windows enviará el mensaje correspondiente a una ventana en caso de que algo relevante a esa ventana que sucede. El procedimiento de ventana recibe el mensaje y reacciona a ella de forma inteligente. wParam y lParam son sólo parámetros adicionales para el uso de algunos mensajes. Algunos mensajes se envían datos adjuntos, en adición al propio mensaje. Esos datos se pasan al procedimiento de ventana a través de lParam y wParam.

```
. SI uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
```

Aquí viene la parte crucial. Aquí es donde la mayor parte de la inteligencia de su programa reside. Los códigos que responden a cada mensaje de Windows se encuentran en el procedimiento de ventana. El código debe comprobar el mensaje de Windows para ver si es un mensaje que está interesado pulg Si es así, hacer lo que quieras hacer en respuesta a ese mensaje y luego regresar con el cero en eax. Si no es así, usted debe llamar a DefWindowProc, pasando por todos los parámetros que ha recibido para la transformación por defecto .. Este DefWindowProc es una función API que procesa los mensajes que su programa no está interesado pulg
El único mensaje que usted debe responder es WM_DESTROY. Este mensaje es enviado a su procedimiento de ventana cada vez que la ventana está cerrada. En el momento de su procedimiento de ventana recibe este mensaje, la ventana ya está retirado de la pantalla. Esto es sólo una notificación de que su ventana estaba destruida, usted debe prepararse para volver a Windows. En respuesta a esto, usted puede realizar la limpieza antes de regresar a Windows. Usted no tiene más remedio que dejar de fumar cuando se trata de este estado. Si usted quiere tener una oportunidad para detener el usuario cierre la ventana, se debe procesar el mensaje WM_CLOSE. Ahora de nuevo a WM_DESTROY, después de realizar las tareas de limpieza, se debe llamar PostQuitMessage que publicará WM_QUIT de nuevo a su módulo. WM_QUIT hará devolución GetMessage con valor cero en eax, que a su vez, termina el bucle de mensajes y se cierra a Windows. Puede enviar un mensaje WM_DESTROY a su propio procedimiento de ventana llamando a la función DestroyWindow.

TUTORIAL 4: PINTANDO CON TEXTO

En este tutorial, vamos a aprender a "pintar" el texto en el área de cliente de una ventana. También vamos a aprender acerca de contexto de dispositivo.

TEORÍA:

Texto en Windows es un tipo de objeto GUI. Cada personaje se compone de numerosos píxeles (puntos) que se agrupan en un patrón definido. Es por eso que se llama "pintura" en lugar de "por escrito". Normalmente, se pinta el texto en su propia área cliente (en realidad, usted puede pintar fuera del área de cliente, pero eso es otra historia). Poner texto en la pantalla en Windows es drásticamente diferente de DOS. En DOS, se puede pensar de la pantalla en la dimensión de 80x25. Sin embargo, en Windows, la pantalla son compartidos por varios programas. Algunas reglas deben aplicarse para evitar los programas de escritura de uno sobre el otro de la pantalla. Windows garantiza esto, limite el área de pintura de cada ventana a su propia área de cliente única. El tamaño del área cliente de una ventana no es también constante. El usuario puede cambiar el tamaño en cualquier momento. Así que debe determinar las dimensiones de su área de cliente propia dinámica.

Antes de poder pintar algo en el área de cliente, debe pedir permiso a Windows. Así es, usted no tiene el control absoluto de la pantalla a medida que se encontraban en DOS. Usted debe pedir a Windows permiso para pintar su área cliente. Windows determinar el tamaño de su área de cliente, tipo de letra, colores y otros atributos de GDI y envía un identificador de contexto de dispositivo de regreso a su programa. A continuación, puede utilizar el contexto de dispositivo como un pasaporte a la pintura en su área de cliente.

¿Qué es un contexto de dispositivo? Es sólo una estructura de datos mantenida internamente por Windows. Un contexto de dispositivo está asociado con un dispositivo particular, tal como una pantalla de la impresora o vídeo. Para una pantalla de vídeo, un contexto de dispositivo se asocia generalmente con una ventana en particular en la pantalla.

Algunos de los valores en el contexto de dispositivo son los atributos gráficos como los colores de fuente, etc Estos son los valores por defecto que se puede cambiar a voluntad. Ellos existen para ayudar a reducir la carga de tener que especificar estos atributos en las llamadas a funciones GDI cada.

Usted puede pensar en un contexto de dispositivo como entorno por defecto preparado para usted por Windows. Se puede reemplazar algunos valores por defecto más adelante si así lo desea.

Cuando un programa necesita para pintar, debe obtener un identificador para un contexto de dispositivo. Normalmente, hay varias maneras de lograr esto.

llamar a BeginPaint en respuesta al mensaje WM_PAINT.
llamar a GetDC en respuesta a otros mensajes.
llame CreateDC para crear su propio contexto de dispositivo

Una cosa que usted debe recordar, una vez has terminado con el identificador de contexto de dispositivo, debe liberarlo durante la tramitación de un solo mensaje. No obtener el mango en respuesta a un mensaje y lo liberan en respuesta a otra.

De Windows mensajes WM_PAINT mensajes a una ventana para notificar que ya es hora de volver a pintar su área cliente. Windows no guardar el contenido del área de cliente de una ventana. En cambio, cuando ocurre una situación que merece un repinte del área de cliente (por ejemplo, cuando una ventana estaba cubierta por otro y se descubre solo), Windows pone mensaje WM_PAINT en la cola de mensaje de que la ventana. Es la responsabilidad de esa

ventana para volver a pintar su área cliente. Usted debe reunir toda la información acerca de cómo volver a pintar su área cliente en la sección WM_PAINT de tu procedimiento de ventana, por lo que la ventana se puede volver a pintar procudure el área de cliente cuando el mensaje llega WM_PAINT.

Otro concepto que debe llegar a un acuerdo con el rectángulo inválido. Windows define un rectángulo inválido como el área rectangular más pequeña en el área de cliente que necesita ser repintada. Cuando Windows detecta un rectángulo inválido en el área de cliente de una ventana, envía mensajes WM_PAINT a esa ventana. En respuesta al mensaje WM_PAINT, la ventana se puede obtener una estructura PAINTSTRUCT que contiene, entre otros, la coordenada del rectángulo inválido. Usted llama a BeginPaint en respuesta al mensaje WM_PAINT para validar el rectángulo inválido. Si no procesar el mensaje WM_PAINT, por lo menos debe llamar a DefWindowProc o ValidateRect para validar el rectángulo inválido cosa de Windows en repetidas ocasiones le enviará mensaje WM_PAINT.

A continuación se muestran los pasos que debe realizar en respuesta a un mensaje WM_PAINT:

Obtener un manejador al contexto de dispositivo con BeginPaint.
Pinte el área de cliente.
Suelte el manejador del contexto de dispositivo con EndPaint

Tenga en cuenta que no tiene que validar explícitamente el rectángulo inválido. Se realiza automáticamente por la llamada BeginPaint. Entre BeginPaint-EndPaint par, usted puede llamar a cualquiera de las funciones GDI para pintar tu área de cliente. Casi todos ellos requieren el manejador del contexto de dispositivo como un parámetro.

Contenido:

Vamos a escribir un programa que muestra una cadena de texto "Win32 conjunto es grande y fácil!" en el centro del área de cliente.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ user32.lib
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib

. DATOS
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0
OurText db "Win32 conjunto es grande y fácil!", 0

. DATOS?
HINSTANCE hInstance?
LPSTR de línea de comandos?

. CÓDIGO
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
```

```
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL hdc: HDC
LOCAL ps: PAINTSTRUCT
LOCAL rect: RECT
. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_PAINT
invocar BeginPaint, hWnd, ADDR ps
mov hdc, eax
invocar GetClientRect, hWnd, ADDR rect
invocar DrawText, hdc, ADDR OurText, -1, ADDR rect, \
DT_SINGLELINE o DT_CENTER o DT_VCENTER
invocar EndPaint, hWnd, ADDR ps
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
```

```
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

La mayoría del código es el mismo que el ejemplo de la guía 3. Voy a explicar sólo los cambios importantes.

```
  LOCAL hdc: HDC
LOCAL ps: PAINTSTRUCT
LOCAL rect: RECT
```

Estas son las variables locales que son utilizados por las funciones GDI en nuestra sección WM_PAINT. HDC se utiliza para almacenar el identificador de contexto de dispositivo de regresar de llamada BeginPaint. ps es una estructura PAINTSTRUCT. Normalmente, usted no utiliza los valores de PS. Ha pasado a la función BeginPaint y Windows se llena con los valores apropiados. A continuación, pasar a ps para EndPaint función cuando termine de pintar el área cliente. rect es una estructura RECT definen como sigue:

```
RECT Struct
larga por la izquierda?
la parte superior TIEMPO?
derecho TIEMPO?
inferior TIEMPO?
RECT termina
```

Izquierda y de arriba son las coordenadas de la esquina superior izquierda de un rectángulo de la derecha y de abajo son las coordenadas de la esquina inferior derecha. Una cosa para recordar: El origen de los ejes xy está en la esquina superior izquierda del área de cliente. Así que el punto y = 10 está por debajo del punto y = 0.

```
  invocar BeginPaint, hWnd, ADDR ps
mov hdc, eax
invocar GetClientRect, hWnd, ADDR rect
invocar DrawText, hdc, ADDR OurText, -1, ADDR rect, \
DT_SINGLELINE o DT_CENTER o DT_VCENTER
invocar EndPaint, hWnd, ADDR ps
```

En respuesta al mensaje WM_PAINT, se llama a BeginPaint con la manija de la ventana que quieres pintar y una estructura no inicializada PAINTSTRUCT como parámetros. Después de una llamada exitosa, eax contiene el manejador del contexto de dispositivo. A continuación, llamar a GetClientRect para recuperar la dimensión del área de cliente. La dimensión se devuelve en la variable rect, que se pasa a DrawText como uno de sus parámetros. La sintaxis DrawText es la siguiente:

```
DrawText proto hdc: HDC, lpString: DWORD, nCount: DWORD, lpRect: DWORD,
uFormat: DWORD
```

DrawText es un alto nivel de salida de texto función de la API. Se ocupa de algunos detalles morbosos, tales como el ajuste de línea, centrado, etc por lo que podría concentrarse en la cadena que desea pintar. Su bajo nivel hermano, TextOut, será examinado en el siguiente tutorial. Formatos DrawText una cadena de texto para que quepa dentro de los límites de un rectángulo. Utiliza la fuente seleccionada, el color y el fondo (en el contexto de dispositivo) para dibujar las text.Lines se envuelven para encajar dentro de los límites del rectángulo. Se

devuelve la altura del texto de salida en unidades de dispositivo, en nuestro caso, los píxeles. Veamos sus parámetros:

hdc manejador del contexto de dispositivo de
lpString El puntero a la cadena que quieres dibujar en el rectángulo. La cadena debe ser terminada en cero otra cosa que tendría que especificar su longitud en el siguiente parámetro, nCount.
nCount El número de caracteres a la salida. Si la cadena es terminada en cero, nCount debe ser -1. De lo contrario nCount debe contener el número de caracteres en la cadena que desea llamar.
lpRect El puntero al rectángulo (una estructura de tipo RECT) que desea señalar a la cadena de in Tenga en cuenta que este rectángulo es también un rectángulo de recorte, es decir, que no podía sacar la cadena fuera de este rectángulo.
uFormat El valor que especifica cómo la cadena se muestra en el rectángulo. Usamos tres valores combinados por "o" del operador:

- **DT_SINGLELINE** especifica una sola línea de texto
- **DT_CENTER** centra el texto horizontalmente.
- **DT_VCENTER** centra el texto verticalmente. Se debe utilizar con DT_SINGLELINE.

Después de terminar de pintar el área cliente, debe llamar a la función EndPaint para liberar el identificador de contexto de dispositivo.
Eso es todo. Podemos resumir los puntos más importantes aquí:

- Usted llama a BeginPaint-EndPaint par en respuesta al mensaje WM_PAINT.
- Haz lo que quieras con el área de cliente entre las llamadas a BeginPaint y EndPaint.
- Si desea pintar su área cliente en respuesta a otros mensajes, usted tiene dos opciones:
  - Utilice GetDC-ReleaseDC pareja y hacer su pintura entre estas llamadas
  - Llame o InvalidateRect UpdateWindow para invalidar todo el área cliente, obligando a Windows a poner el mensaje WM_PAINT en la cola de mensajes de la ventana y hacer su pintura en la sección WM_PAINT

TUTORIAL 5: MÁS SOBRE EL TEXTO

Vamos a experimentar más con los atributos de texto, es decir. fuente y el color.

TEORÍA:

Sistema de colores de Windows se basa en los valores RGB, R = rojo, G = verde, B = Azul. Si desea especificar un color en Windows, usted debe indicar el color deseado en términos de estos tres colores principales. Cada valor de color tiene un rango de 0 a 255 (un valor de byte). Por ejemplo, si desea que el color rojo puro, debe usar 255,0,0. O si quieres un color blanco puro, debe utilizar 255.255.255. Se puede ver en los ejemplos que conseguir el color que necesitas es muy difícil con este sistema, ya que tienes que tener una buena comprensión de cómo mezclar y combinar colores.
Por el color del texto y el fondo, se utiliza SetTextColor y SetBkColor, tanto de ellos requieren un identificador de contexto de dispositivo y un valor de 32-bit RGB. La estructura del valor de 32-bits RGB se define como:

RGB_value estructura
sin usar 0 db
azul db?
verde db?
de color rojo db?
RGB_value termina

Nótese que el primer byte no se utiliza y debe ser cero. El orden de los tres bytes restantes se invierte, es decir. azul, verde, rojo. Sin embargo, no vamos a utilizar esta estructura, ya que es engorroso para inicializar y utilizar. Vamos a crear una macro. La macro recibe tres parámetros: los valores de rojo, verde y azul. Se va a producir el deseado valor de 32 bits RGB y guárdelo en eax. La macro es como sigue:

```
RGB macro rojo, verde, azul
xor eax, eax
mov ah, azul
SHL eax, 8
mov ah, verde
mov al, rojo
ENDM
```

Usted puede poner esta macro en el archivo de inclusión para su uso futuro.
Usted puede "crear" un tipo de letra, o llamando al CreateFont CreateFontIndirect. La diferencia entre las dos funciones es que CreateFontIndirect recibe un solo parámetro: un puntero a una estructura de fuente lógica, LOGFONT. CreateFontIndirect es el más flexible de los dos, especialmente si sus programas tienen que cambiar las fuentes con frecuencia. Sin embargo, en nuestro ejemplo, vamos a "crear" una sola fuente para la demostración, que puede salirse con la CreateFont. Después de la llamada a CreateFont, devolverá un identificador de un tipo de letra que se debe seleccionar en el contexto de dispositivo. Después de eso, todas las funciones API de texto se utiliza la fuente que hemos seleccionado en el contexto de dispositivo.

Contenido:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ gdi32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ gdi32.lib

RGB macro rojo, verde, azul
xor eax, eax
mov ah, azul
SHL eax, 8
mov ah, verde
mov al, rojo
ENDM

. Datos
ClassName db "SimpleWinClass", 0
```

```
AppName db "Nuestra Primera Ventana", 0
TestString db "Win32 conjunto es grande y fácil!", 0
FontName db "guión", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL hdc: HDC
LOCAL ps: PAINTSTRUCT
LOCAL HFONT: HFONT
```

```
. SI uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_PAINT
invocar BeginPaint, hWnd, ADDR ps
mov hdc, eax
invocar CreateFont, 24,16,0,0,400,0,0,0, OEM_CHARSET, \
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, \
DEFAULT_QUALITY, DEFAULT_PITCH o FF_SCRIPT, \
ADDR FontName
invocar SelectObject, hdc, eax
mov HFONT, eax
RGB 200,200,50
invocar SetTextColor, hdc, eax
RGB 0,0,255
invocar SetBkColor, hdc, eax
invocar TextOut, hdc, 0,0, ADDR TestString, sizeof TestString
invocar SelectObject, hdc, HFONT
invocar EndPaint, hWnd, ADDR ps
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

poner fin a empezar a
```

Análisis:

```
invocar CreateFont, 24,16,0,0,400,0,0,0, OEM_CHARSET, \
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, \
DEFAULT_QUALITY, DEFAULT_PITCH o FF_SCRIPT, \
ADDR FontName
```

CreateFont crea una fuente lógica de que es lo más parecido a los parámetros dados y los datos de fuentes disponibles. Esta función tiene más parámetros que cualquier otra función en Windows. Devuelve un identificador de fuente lógica para ser utilizado por la función SelectObject. Vamos a examinar sus parámetros en detalle.

```
CreateFont proto nHeight: DWORD, \
nWidth: DWORD, \
nEscapement: DWORD, \
nOrientation: DWORD, \
nWeight: DWORD, \
cItalic: DWORD, \
cUnderline: DWORD, \
cStrikeOut: DWORD, \
cconjunto: DWORD, \
cOutputPrecision: DWORD, \
cClipPrecision: DWORD, \
cQuality: DWORD, \
cPitchAndFamily: DWORD, \
lpFacename: DWORD
```

nHeight La altura deseada de los caracteres. 0 significa que el tamaño de su uso por defecto.
nWidth La anchura deseada de los personajes. Normalmente, este valor debe ser 0, lo que permite a Windows para que coincida con el ancho a la altura. Sin embargo, en nuestro

ejemplo, el ancho por defecto hace que los personajes difícil de leer, así que usar el ancho de 16 en su lugar.
nEscapement Especifica la orientación de la salida del siguiente carácter en relación con la anterior en décimas de grado. Normalmente, el valor 0. Se establece en 900 para que todos los personajes van hacia arriba desde el primer carácter, 1800 a escribir al revés, o 2700 para escribir los caracteres de arriba hacia abajo.
nOrientation especifica en qué medida el personaje debe girar al reproducirlas en décimas de grado. Se establece en 900 para que todos los personajes que mienten sobre sus espaldas, 1800 para la escritura al revés, etc
nWeight Establece el grosor de las líneas de cada personaje. Windows define las siguientes medidas:

FW_DONTCARE equ 0
FW_THIN equ 100
FW_EXTRALIGHT equ 200
FW_ULTRALIGHT equ 200
FW_LIGHT equ 300
FW_NORMAL equ 400
FW_REGULAR equ 400
FW_MEDIUM equ 500
FW_SEMIBOLD equ 600
FW_DEMIBOLD equ 600
FW_BOLD equ 700
FW_EXTRABOLD equ 800
FW_ULTRABOLD equ 800
FW_HEAVY equ 900
FW_BLACK equ 900

cItalic 0 para normal, cualquier otro valor para caracteres en cursiva.
cUnderline 0 para normal, cualquier otro valor para caracteres subrayados.
cStrikeOut 0 para normal, cualquier otro valor para los personajes con una línea por el centro.
cconjunto El juego de caracteres de la fuente. Normalmente debe ser OEM_CHARSET que permite a Windows para seleccionar la fuente que está dependiente del sistema operativo.
cOutputPrecision Especifica la cantidad de la fuente seleccionada debe estar estrechamente ligada a las características que queremos. Normalmente debe ser OUT_DEFAULT_PRECIS que define la fuente por defecto comportamiento de asignación.
cClipPrecision Especifica la precisión de recorte. La precisión de recorte define cómo recortar los caracteres que están parcialmente fuera de la región de recorte. Usted debe ser capaz de pasar con CLIP_DEFAULT_PRECIS que define el comportamiento de recorte por defecto.
cQuality Especifica la calidad de salida. La calidad de salida define con cuanto cuidado GDI debe intentar hacer coincidir los atributos de la fuente a los de una fuente física actual. Hay tres opciones: DEFAULT_QUALITY, PROOF_QUALITY y DRAFT_QUALITY.
cPitchAndFamily Especifica el tono y la familia de la fuente. Usted debe combinar el valor del tono y el valor de la familia con "o" del operador.
lpFacename Un puntero a una cadena terminada en cero que especifica el tipo de letra de la fuente.

La descripción anterior no es en absoluto exhaustiva. Usted debe consultar a su referencia de la API de Win32 para obtener más detalles.

```
  invocar SelectObject, hdc, eax
mov HFONT, eax
```

Después de obtener el identificador de la fuente lógica, que debemos utilizar para seleccionar la fuente en el contexto de dispositivo llamando a SelectObject. SelectObject pone a los nuevos objetos GDI tales como bolígrafos, brochas y las fuentes en el contexto de dispositivo para ser utilizado por las funciones GDI. Devuelve el identificador del objeto sustituye la que debemos ahorrar para la llamada SelectObject futuro. Después de la llamada SelectObject, cualquier función de salida de texto se utiliza la fuente que acaba de seleccionar en el contexto de dispositivo.

RGB 200,200,50
invocar SetTextColor, hdc, eax
RGB 0,0,255
invocar SetBkColor, hdc, eax

Utilice RGB macro para crear un valor de 32-bit RGB para ser utilizado por SetColorText y SetBkColor.

invocar TextOut, hdc, 0,0, ADDR TestString, sizeof TestString

Llame a la función TextOut para dibujar el texto en el área de cliente. El texto estará en la fuente y el color que se especifica anteriormente.

invocar SelectObject, hdc, HFONT

Cuando acabemos con la fuente, debemos restaurar la fuente vieja de nuevo en el contexto de dispositivo. Siempre se debe restaurar el objeto que ha sustituido en el contexto de dispositivo

TUTORIAL 6: ENTRADA DE TECLADO

Vamos a aprender cómo un programa de Windows recibe la entrada del teclado.

Teoría:

Ya que normalmente sólo hay un teclado en cada PC, todos los programas de Windows tienen que compartir entre ellos. Windows es el responsable de enviar las pulsaciones de teclas a la ventana que tiene el foco de entrada.
Aunque puede haber varias ventanas en la pantalla, sólo uno de ellos tiene el foco de entrada. La ventana que tiene el foco de entrada es el único que puede recibir pulsaciones de teclas. Se puede diferenciar la ventana que tiene el foco de entrada de otras ventanas mirando a la barra de título. La barra de título de la ventana que tiene el foco de entrada se pone de relieve.
En realidad, hay dos tipos principales de mensajes del teclado, dependiendo de su punto de vista del teclado. Puede ver un teclado como una colección de claves. En este caso, si se presiona una tecla, Windows envía un mensaje WM_KEYDOWN a la ventana que tiene el foco de entrada, notificando que se pulsa una tecla. Cuando se suelta la tecla, Windows envía un mensaje WM_KEYUP. Usted trata a una tecla, un botón. Otra forma de mirar el teclado es que es un dispositivo de entrada de caracteres. Al pulsar tecla "a", Windows envía un mensaje WM_CHAR a la ventana que tiene el foco de entrada, diciéndole que el usuario envía "un" carácter a la misma. De hecho, Windows envía mensajes WM_KEYDOWN y WM_KEYUP a la ventana que tiene el foco de entrada y los mensajes serán traducidos a los mensajes WM_CHAR por la llamada TranslateMessage. El procedimiento de ventana puede decidir procesar a los tres mensajes, o sólo los mensajes que se interese mayoría de las veces, puede pasar por alto WM_KEYDOWN y WM_KEYUP ya llamada a la función TranslateMessage en el bucle de mensajes WM_KEYDOWN y traducir mensajes WM_KEYUP a los mensajes WM_CHAR. Nos centraremos en WM_CHAR en este tutorial.

Ejemplo:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ gdi32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ gdi32.lib

. Datos
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0
caracteres WPARAM 20h, y el carácter que el programa reciba desde el teclado

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
```

```
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL hdc: HDC
LOCAL ps: PAINTSTRUCT

. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_CHAR
impulsar wParam
pop caracteres
invocar InvalidateRect, hWnd, NULL, TRUE
. UMsg ELSEIF == WM_PAINT
invocar BeginPaint, hWnd, ADDR ps
mov hdc, eax
invocar TextOut, hdc, 0,0, ADDR char, un
invocar EndPaint, hWnd, ADDR ps
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

caracteres WPARAM 20h, y el carácter que el programa reciba desde el teclado

Esta es la variable que almacenará el carácter recibido desde el teclado. Dado que el carácter se envía en wParam del procedimiento de ventana, se define la variable como el tipo WPARAM por la simplicidad. El valor inicial es de 20 horas o el espacio ya que cuando nuestras actualizaciones de las ventanas de su área de cliente la primera vez, no hay ninguna entrada de caracteres. Así que queremos ver el espacio en su lugar.

```
. UMsg ELSEIF == WM_CHAR
impulsar wParam
pop caracteres
invocar InvalidateRect, hWnd, NULL, TRUE
```

Esto se añade en el procedimiento de ventana para manejar el mensaje WM_CHAR. Simplemente pone el carácter en la variable llamada "char" y luego llama a InvalidateRect. InvalidateRect hace que el rectángulo especificado no válido en el área de cliente de Windows que obliga a enviar el mensaje WM_PAINT al procedimiento de ventana. Su sintaxis es la siguiente:

**InvalidateRect proto hWnd: HWND, **
**lpRect: DWORD, **
bErase: DWORD

lpRect es un puntero a la rectagle en el área de cliente que queremos declarar no válido. Si este parámetro es nulo, toda el área cliente será marcado como inválido.
bErase es un indicador que Windows si tiene que borrar el fondo. Si este indicador es TRUE, Windows borrará el fondo de color del rectángulo no válido cuando BeginPaint se llama.

Así que la estrategia que se utiliza aquí es que: nosotros guardamos toda la información necesaria relativa a pintar el área cliente y generar mensaje WM_PAINT para pintar el área cliente. Por supuesto, los códigos de la sección WM_PAINT debe saber de antemano lo que se espera de ellos. Esto parece una manera indirecta de hacer las cosas, pero es el camino de Windows.
En realidad, podemos pintar el área cliente durante el procesamiento de mensajes WM_CHAR llamando al par GetDC y ReleaseDC. No hay ningún problema allí. Pero la diversión comienza cuando la ventana tiene que volver a pintar su área cliente. Dado que los códigos que pintan el personaje está en la sección WM_CHAR, el procedimiento de ventana no podrá volver a pintar nuestro personaje en el área de cliente. Así que la conclusión es la siguiente: poner todos los datos necesarios y los códigos que se pintan en WM_PAINT. Puede enviar un mensaje WM_PAINT desde cualquier lugar de su código en cualquier momento que desee volver a pintar el área cliente.

invocar TextOut, hdc, 0,0, ADDR char, un

Cuando InvalidateRect se llama, envía un mensaje WM_PAINT de nuevo al procedimiento de ventana. Por lo tanto los códigos de la sección WM_PAINT se llama. Hace un llamamiento BeginPaint como de costumbre para obtener el identificador de contexto de dispositivo y luego llamar a TextOut que atrae a nuestro personaje en el área de cliente en x = 0, y = 0. Cuando se ejecuta el programa y pulse cualquier tecla, usted verá que el eco de caracteres en la esquina superior izquierda de la ventana del cliente. Y cuando la ventana está minimizada y maximizada una vez más, el personaje sigue ahí ya que todos los códigos y los datos esenciales para volver a pintar, todos reunidos en la sección WM_PAINT.

TUTORIAL 7: LA ENTRADA DEL MOUSE

Vamos a aprender a recibir y responder a la entrada del ratón en nuestro procedimiento de ventana. El programa de ejemplo va a esperar por los clics de ratón izquierdo y mostrar una cadena de texto en el punto exacto clic en el área de cliente.

.

TEORÍA:

Al igual que con la entrada de teclado, Windows detecta y envía notificaciones sobre las actividades del ratón que son relevantes para cada ventana. Estas actividades incluyen los clics derecho e izquierdo, el movimiento del cursor del ratón sobre la ventana, los clics dobles. A diferencia de la entrada de teclado que se dirige a la ventana que tiene el foco de entrada, mensajes de ratón se envían a cualquier ventana que el cursor del ratón encima, activa o no. Además, hay mensajes del ratón sobre el área no cliente también. Pero la mayor parte del tiempo, que felizmente puede ignorar. Podemos centrarnos en las que se refieren al área de cliente.
Hay dos mensajes para cada botón del ratón: WM_LBUTTONDOWN, WM_RBUTTONDOWN y WM_LBUTTONUP, WM_RBUTTONUP mensajes. Para un ratón con tres botones, también

hay WM_MBUTTONDOWN y WM_MBUTTONUP. Cuando el cursor del ratón se mueve sobre el área de cliente, Windows envía mensajes WM_MOUSEMOVE a la ventana bajo el cursor.
Una ventana puede recibir mensajes de doble clic, o WM_LBUTTONDBCLK WM_RBUTTONDBCLK, ***si y sólo si*** su clase de la ventana tiene bandera de estilo CS_DBLCLKS, de lo contrario la ventana sólo recibirá una serie de botón del ratón arriba y abajo de los mensajes.
Para todos estos mensajes, el valor de lParam contiene la posición del ratón. La palabra baja es la coordenada x, y la palabra de mayor peso es la coordenada relativa a la esquina superior izquierda del área cliente de la ventana. wParam indica el estado de los botones del ratón y las teclas Shift y Ctrl.

EJEMPLO:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ gdi32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ gdi32.lib

. Datos
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0
MouseClick db 0; 0 = sin embargo, haga clic en

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
Hitpoint PUNTO <>

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
```

```
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL hdc: HDC
LOCAL ps: PAINTSTRUCT

. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_LBUTTONDOWN
mov eax, lParam
y EAX, 0FFFFh
mov hitpoint.x, eax
mov eax, lParam
SHR EAX, 16
mov hitpoint.y, eax
mov MouseClick, TRUE
invocar InvalidateRect, hWnd, NULL, TRUE
. UMsg ELSEIF == WM_PAINT
invocar BeginPaint, hWnd, ADDR ps
mov hdc, eax
. Sl MouseClick
invocar lstrlen, ADDR AppName
invocar TextOut, hdc, hitpoint.x, hitpoint.y, ADDR AppName, eax
. ENDIF
invocar EndPaint, hWnd, ADDR ps
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

```
. UMsg ELSEIF == WM_LBUTTONDOWN
mov eax, lParam
y EAX, 0FFFFh
mov hitpoint.x, eax
mov eax, lParam
SHR EAX, 16
mov hitpoint.y, eax
mov MouseClick, TRUE
invocar InvalidateRect, hWnd, NULL, TRUE
```

El procedimiento de ventana espera a que haga clic en el botón izquierdo del ratón. Cuando se recibe WM_LBUTTONDOWN, lParam contiene las coordenadas del cursor del ratón en el área de cliente. Se guarda la coordenada en una variable de tipo POINT que se define como:

```
PUNTO STRUCT
x dd?
y dd?
PUNTO DE FIN
```

y establece la marca, MouseClick, en TRUE, lo que significa que hay al menos un clic con el botón izquierdo del ratón en el área de cliente.

```
mov eax, lParam
y EAX, 0FFFFh
mov hitpoint.x, eax
```

Desde la coordenada x es la palabra baja de lParam y los miembros de la estructura POINT son de 32 bits de tamaño, tenemos que ajustar a cero la palabra alta de eax antes de almacenarla en hitpoint.x.

```
SHR EAX, 16
mov hitpoint.y, eax
```

Debido a que la coordenada y es la palabra alta de lParam, hay que ponerlo en la palabra baja de eax antes de almacenarla en hitpoint.y. Hacemos esto al cambiar eax 16 bits a la derecha.
Después de almacenar la posición del ratón, nos pusimos la bandera, MouseClick, en TRUE con el fin de permitir que el código de dibujo en la sección WM_PAINT saber que hay al menos un clic en el área de cliente para que pueda señalar a la cadena en la posición del ratón. A continuación llamamos a la función InvalidateRect para forzar la ventana para volver a pintar su área cliente.

```
. SI MouseClick
invocar lstrlen, ADDR AppName
invocar TextOut, hdc, hitpoint.x, hitpoint.y, ADDR AppName, eax
. ENDIF
```

El código de la pintura en la sección WM_PAINT debe comprobar si MouseClick es cierto, ya que cuando la ventana se creó, recibió un mensaje WM_PAINT que en ese momento, no hay un clic del ratón se había producido por lo que no debería llamar la cadena en el área de cliente. Inicializamos MouseClick a FALSE y cambiar su valor a true cuando un clic del ratón real ocurre.
Si por lo menos un clic del ratón se ha producido, señala a la cadena en el área de cliente en la posición del ratón. Tenga en cuenta que se llama lstrlen para obtener la longitud de la cadena para mostrar y envía la longitud como el último parámetro de la función TextOut.

TUTORIAL 8: MENÚ

En este tutorial, vamos a aprender cómo incorporar un menú a nuestra ventana.
Descargar el ejemplo 1 y ejemplo 2.

TEORÍA:

El menú es uno de los componentes más importante de su ventana. Menú presenta una lista de servicios que un programa ofrece al usuario. El usuario no tiene que leer el manual que se incluye con el programa para poder usarlo, se puede leer el menú para obtener una visión general de la capacidad de un determinado programa y empezar a jugar con él de inmediato. Desde un menú es una herramienta para conseguir que el usuario en marcha rápidamente, usted debe seguir la norma. Sucintamente palabras, los dos primeros elementos del menú debería ser Archivo y Edición y el último debe ser de ayuda. Puede insertar sus propios elementos de menú entre Editar y Ayuda. Si un elemento de menú invoca un cuadro de diálogo, se debe agregar unos puntos suspensivos (...) a la cadena de menú.
Menú es un tipo de recurso. Hay varios tipos de recursos, como el cuadro de diálogo, la tabla de cadenas, icono, mapa de bits, los recursos de menú, etc, se describen en un archivo separado llamado archivo de recursos que normalmente tiene. Rc. A continuación, combinar los recursos con el código fuente durante la etapa de enlace. El producto final es un archivo ejecutable que contiene tanto las instrucciones y los recursos.
Puede escribir secuencias de comandos de recursos con cualquier editor de texto. Están compuestos de frases que describen las apariencias y otros atributos de los recursos utilizados en un programa en particular Aunque se puede escribir secuencias de comandos de recursos con un editor de texto, es bastante engorroso. Una mejor alternativa es utilizar un editor de recursos que le permite diseñar visualmente los recursos con facilidad. Editores de recursos se suelen incluir en los paquetes del compilador como Visual C + +, Borland C + +, etc
Usted describe un recurso de menú como este:

```
MiMenú MENÚ
{
[Menú de la lista aquí]
}
```

Los programadores de C puede reconocer que es similar a la declaración de una estructura. **MiMenú** ser un nombre de menú seguido por palabra clave **MENU** y la lista de menú dentro de llaves. Alternativamente, puede usar BEGIN y END en lugar de las llaves, si lo desea. Esta sintaxis es más aceptable para los programadores de Pascal.
Lista de menús puede ser comunicado **MENUITEM** o **POPUP.**
Declaración **MENUITEM** define una barra de menú, que no invoca un menú emergente cuando la sintaxis selected.The es el siguiente:

```
MENUITEM "& texto", id [, opciones]
```

Se inicia con la palabra MENUITEM seguido por el texto que desea utilizar como cadena de la barra de menús. Tenga en cuenta el signo. Se hace que el carácter que sigue a ser subrayado. A raíz de la cadena de texto es el ID del elemento de menú. El ID es un número que se utiliza para identificar el elemento de menú en el mensaje enviado al procedimiento de ventana, cuando el elemento de menú seleccionado. Como tal, cada identificador de menú debe ser único entre ellos.
Las opciones son opcionales. Las opciones disponibles son las siguientes:

- **Atenuado** El elemento de menú está inactivo, y que no genera un mensaje WM_COMMAND. El texto está en gris.
- **INACTIVO** El elemento del menú está inactivo, y que no genera un mensaje WM_COMMAND. El texto se muestra normalmente.
- **MENUBREAK** Este artículo y los artículos siguientes aparecen en una nueva línea del menú.
- **AYUDA** Este artículo y los siguientes puntos son justificado a la derecha.

Usted puede utilizar uno de la opción anterior o en combinación con "o" del operador. Tenga en cuenta que **INACTIVO** y en **gris** no se pueden combinar entre sí.
Declaración **POPUP** tiene la siguiente sintaxis:

```
POPUP "& texto" [, opciones]
{
```

[Lista del menú]
} Declaración POPUP define una barra de menús que, cuando se selecciona, se despliega una lista de elementos de menú en una pequeña ventana emergente. La lista del menú puede ser una declaración **MENUTIEM o** emergente. Hay un tipo especial de declaración de **MENUITEM, SEPARATOR MENUITEM,** que se trace una línea horizontal en la ventana emergente.
El siguiente paso después de haber terminado con el guión recurso de menú es hacer referencia a ella en su programa.
Usted puede hacer esto en dos lugares diferentes en su programa.

- En lpszMenuName miembro de la estructura WNDCLASSEX. Es decir, si usted tiene un menú llamado "FirstMenu", se puede asignar el menú a su ventana como esta:

```
. DATOS

        MenuName db "FirstMenu", 0

.....................
.....................
. CÓDIGO

        .....................
        mov wc.lpszMenuName, MenuName OFFSET
        .....................
```

- En el parámetro del menú mango de CreateWindowEx de esta manera:

```
. DATOS

        MenuName db "FirstMenu", 0
        HMENU hMenu?

.....................
.....................
. CÓDIGO

        .....................
        invocar LoadMenu, hInst, MenuName OFFSET
        mov hMenu, eax
        invocar CreateWindowEx, NULL, ClsName OFFSET, \
        OFFSET Leyenda, WS_OVERLAPPEDWINDOW \
        CW_USEDEFAULT, CW_USEDEFAULT, \
        CW_USEDEFAULT, CW_USEDEFAULT, \
        NULL, \
        hMenu, \
        hInst, \
        NULL \
        .....................
```

Así que usted puede preguntar, ¿cuál es la diferencia entre estos dos métodos?
Cuando se hace referencia en el menú en la estructura WNDCLASSEX, el menú se convierte en el "incumplimiento" del menú para la clase de ventana. Todas las ventanas de esa clase tendrán el mismo menú.
Si desea que cada ventana creada de la misma clase que tienen los diferentes menús, se debe elegir la segunda forma. En este caso, cualquier ventana que se le pasa un mango de menú en su función CreateWindowEx tendrá un menú que "anula" el menú por defecto definido en la estructura WNDCLASSEX.
A continuación vamos a examinar la forma de un menú notifica al procedimiento de ventana

cuando el usuario selecciona un elemento de menú.
Cuando el usuario selecciona un elemento de menú, el procedimiento de ventana recibirá un mensaje WM_COMMAND. La palabra baja de wParam contiene el identificador del menú de la opción del menú seleccionado.
Ahora tenemos suficiente información para crear y utilizar un menú. Vamos a hacerlo.

EJEMPLO:

El primer ejemplo muestra cómo crear y utilizar un menú especificando el nombre del menú en la clase de ventana.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

. Datos
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0
MenuName db "FirstMenu", 0; El nombre de nuestro menú en el archivo de recursos.
Test_string db "Se prueba el ítem del menú", 0
Hello_string db "Hola, mi amigo", 0
Goodbye_string db "nos vemos otra vez, adiós", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?

Const.
IDM_TEST equ 1; Menú ID
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
```

```
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET; Ponemos nuestro nombre menú aquí
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si ax == IDM_TEST
invocar el cuadro de mensaje, NULL, Test_string ADDR, OFFSET AppName, MB_OK
. ELSEIF ax == IDM_HELLO
invocar el cuadro de mensaje, NULL, Hello_string ADDR, OFFSET AppName, MB_OK
. ELSEIF ax == IDM_GOODBYE
invocar el cuadro de mensaje, NULL, Goodbye_string ADDR, OFFSET AppName,
MB_OK
. MÁS
invocar DestroyWindow, hWnd
. ENDIF
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Menu.rc

```
# Define IDM_TEST 1
# Define IDM_HELLO 2
# Define IDM_GOODBYE 3
# Define IDM_EXIT 4
```

```
FirstMenu MENÚ
{
POPUP "& PopUp"
{
MENUITEM "& Say Hello", IDM_HELLO
MENUITEM "Say & Goodbye", IDM_GOODBYE
MENUITEM SEPARATOR
MENUITEM "& Salir", IDM_EXIT
}
MENUITEM "& Prueba", IDM_TEST
}
```

Análisis:

Vamos a analizar el archivo de recursos en primer lugar.

```
# IDM_TEST definir 1 / * igual a IDM_TEST equ 1 * /
# Define IDM_HELLO 2
# Define IDM_GOODBYE 3
# Define IDM_EXIT 4
```

Las líneas anteriores definen los identificadores de menú utilizados por la secuencia de comandos de menú. Usted puede asignar cualquier valor a la identificación, siempre y cuando el valor es único en el menú.

```
FirstMenu MENÚ
```

Declare su menú con la palabra clave MENU.

```
POPUP "& PopUp"
{
MENUITEM "& Say Hello", IDM_HELLO
MENUITEM "Say & Goodbye", IDM_GOODBYE
MENUITEM SEPARATOR
MENUITEM "& Salir", IDM_EXIT
}
```

Definir un menú con cuatro opciones de menú, la tercera es un separador de menú.

```
MENUITEM "& Prueba", IDM_TEST
```

Definir una barra de menú en el menú principal.
A continuación vamos a examinar el código fuente.

```
MenuName db "FirstMenu", 0; El nombre de nuestro menú en el archivo de recursos.
Test_string db "Se prueba el ítem del menú", 0
Hello_string db "Hola, mi amigo", 0
Goodbye_string db "nos vemos otra vez, adiós", 0
```

MenuName es el nombre del menú en el archivo de recursos. Tenga en cuenta que puede definir más de un menú en el archivo de recursos de modo que debe especificar qué menú que desea utilizar. Las otras tres líneas definen las cadenas de texto que se mostrará en los cuadros de mensaje que se invocan cuando el elemento de menú correspondiente es seleccionada por el usuario.

```
IDM_TEST equ 1; Menú ID
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4
```

Definir identificadores de menú para su uso en el procedimiento de ventana. Estos valores deben ser idénticos a los definidos en el archivo de recursos.

```
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si ax == IDM_TEST
invocar el cuadro de mensaje, NULL, Test_string ADDR, OFFSET AppName, MB_OK
. ELSEIF ax == IDM_HELLO
invocar el cuadro de mensaje, NULL, Hello_string ADDR, OFFSET AppName, MB_OK
. ELSEIF ax == IDM_GOODBYE
invocar el cuadro de mensaje, NULL, Goodbye_string ADDR, OFFSET AppName,
MB_OK
. MÁS
invocar DestroyWindow, hWnd
. ENDIF
```

En el procedimiento de ventana, de procesar mensajes WM_COMMAND. Cuando el usuario selecciona un elemento de menú, el menú de Identificación de ese elemento de menú se mandó al procedimiento de ventana en la palabra baja de wParam, junto con el mensaje WM_COMMAND. Por eso, cuando se guarda el valor de wParam en eax, comparamos el valor de hacha para el menú de los identificadores de que hemos definido previamente y actuar en consecuencia. En los tres primeros casos, cuando el usuario selecciona de prueba, Say Hello y Say artículos GoodBye menú, que acabamos de mostrar una cadena de texto en un cuadro de mensaje.

Si el usuario selecciona el punto de salida del menú, que llamamos DestroyWindow con la manija de la ventana como parámetro, que se cerrará la ventana.

Como puede ver, especificando el nombre del menú en una clase de ventana es muy fácil y sencillo. Sin embargo, usted también puede utilizar un método alternativo para cargar un menú en la ventana. No voy a mostrar el código fuente completo aquí. El archivo de recursos es el mismo en ambos métodos. Hay algunos pequeños cambios en el archivo de origen que voy a mostrar a continuación.

```
. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
HMENU hMenu?  ; El mango de nuestro menú
```

Definir una variable de tipo de HMENU para almacenar nuestra mango menú.

```
invocar LoadMenu, hInst, MenuName OFFSET
mov hMenu, eax
INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, hMenu, \
hInst, NULL
```

Antes de llamar a CreateWindowEx, que llamamos LoadMenu con el identificador de instancia y un puntero al nombre de nuestro menú. LoadMenu devuelve el identificador de nuestro menú en el archivo de recursos, que pasamos a CreateWindowEx.

TUTORIAL 9: CONTROLES DE VENTANA SECUNDARIA

En este tutorial, vamos a explorar los controles secundarios de la ventana de entrada que son muy importantes y los dispositivos de salida de nuestros programas.

TEORÍA:

Windows ofrece varias clases de ventana predefinidos que de inmediato se pueden utilizar en nuestros propios programas. La mayoría de las veces los usamos como componentes de un cuadro de diálogo por lo que son por lo general llamados controles secundarios de la ventana. Los controles de ventana hija procesar su propio ratón y teclado y mensajes de notificar a la ventana padre cuando sus estados han cambiado. Ellos aliviar la carga de los programadores enormemente por lo que debe utilizar lo más posible. En este tutorial, los puse en una ventana normal, sólo para demostrar cómo se puede crear y utilizarlos, pero en realidad usted debe ponerlos en un cuadro de diálogo.

Ejemplos de clases de ventana predefinidos son botones, cuadro de lista, casilla de verificación, botón de radio, edición, etc

Con el fin de utilizar un control de ventana hija, se debe crear con CreateWindow o CreateWindowEx. Tenga en cuenta que usted no tiene el registro de la clase de ventana, ya que está registrado por usted por Windows. El parámetro nombre de la clase debe ser el nombre de la clase predefinida. Es decir, si desea crear un botón, se debe especificar el "botón" como el nombre de clase en CreateWindowEx. Los demás parámetros se debe rellenar es el handle de la ventana principal y el identificador de control. El ID de control debe ser único entre los controles. El ID es el identificador de control de ese control. Se utiliza para diferenciar entre los controles.

Después de que el control de su creación, que enviará mensajes de notificación a la ventana padre cuando su estado ha cambiado. Normalmente, se crean las ventanas secundarias durante el mensaje WM_CREATE de la ventana principal. La ventana secundaria envía mensajes WM_COMMAND a la ventana padre con su ID de control en la palabra baja de wParam, el código de notificación en la palabra alta de wParam, y su identificador de ventana en lParam. Cada control de ventana secundaria tiene diferentes códigos de notificación, consulte la referencia de la API de Win32 para más información.

La ventana principal se puede enviar comandos a las ventanas secundarias también, llamando a la función SendMessage. Función SendMessage envía el mensaje especificado con el acompañamiento de los valores en wParam y lParam a la ventana especificada por el identificador de ventana. Es una función muy útil, ya que puede enviar mensajes a cualquier ventana siempre se conoce su identificador de ventana.

Así que, después de crear al niño las ventanas, la ventana padre debe procesar mensajes WM_COMMAND para poder recibir los códigos de notificación de las ventanas secundarias.

EJEMPLO:

Vamos a crear una ventana que contiene un control de edición y de un pulsador. Al hacer clic en el botón, un cuadro de mensaje aparecerá mostrando el texto que escribió en el cuadro de edición. También hay un menú con 4 opciones de menú:

1. **Say Hello** - Coloque una cadena de texto en el cuadro de edición
2. **Edit Clear Box** - Borrar el contenido de la caja de edición
3. **Obtener texto** - Muestra un cuadro de mensaje con el texto en el cuadro de edición
4. **Salir** - Cierra el programa.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
```

```
. Datos
ClassName db "SimpleWinClass", 0
AppName db "Nuestra Primera Ventana", 0
MenuName db "FirstMenu", 0
ButtonClassName db "botón", 0
ButtonText db "Mi primer botón", 0
EditClassName db "editar", 0
TestString db "Wow, estoy en un cuadro de edición ahora", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwndButton HWND?
hwndEdit HWND?
tampón db 512 dup (?); buffer para almacenar el texto extraído de la caja de edición

Const.
ButtonId ecuación 1, el identificador del control del control de botón
EditID equ 2; El identificador del control del control de edición
IDM_HELLO un equ
IDM_CLEAR equ 2
IDM_GETTEXT equ 3
IDM_EXIT equ 4

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_BTNFACE un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, \
ADDR AppName, WS_OVERLAPPEDWINDOW \
CW_USEDEFAULT, CW_USEDEFAULT, \
300.200, NULL, NULL, hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
```

```
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_CREATE
invocar CreateWindowEx, WS_EX_CLIENTEDGE, EditClassName ADDR, NULL, \
WS_CHILD o WS_VISIBLE o WS_BORDER o ES_LEFT o \
ES_AUTOHSCROLL, \
50,35,200,25, hWnd, 8, hInstance, NULL
mov hwndEdit, eax
invocar SetFocus, hwndEdit
invocar CreateWindowEx, NULL, ButtonClassName ADDR, ButtonText ADDR, \
WS_CHILD o WS_VISIBLE o BS_DEFPUSHBUTTON, \
75,70,140,25, hWnd, buttonId, hInstance, NULL
mov hwndButton, eax
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Sl lParam == 0
. Si ax == IDM_HELLO
invocar SetWindowText, hwndEdit, ADDR TestString
. ELSEIF ax == IDM_CLEAR
invocar SetWindowText, hwndEdit, NULL
. ELSEIF ax == IDM_GETTEXT
invocar GetWindowText, hwndEdit, tampón ADDR, 512
invocar el cuadro de mensaje, NULL, ADDR tampón, ADDR AppName, MB_OK
. MÁS
invocar DestroyWindow, hWnd
. ENDIF
. MÁS
. Si ax == buttonId
SHR EAX, 16
. Si ax == BN_CLICKED
invoca SendMessage, hWnd, WM_COMMAND, IDM_GETTEXT, 0
. ENDIF
. ENDIF
. ENDIF
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

Vamos a analizar el programa.

```
. UMsg ELSEIF == WM_CREATE
invocar CreateWindowEx, WS_EX_CLIENTEDGE, \
```

```
EditClassName ADDR, NULL, \
WS_CHILD o WS_VISIBLE o WS_BORDER o ES_LEFT \
o ES_AUTOHSCROLL, \
50,35,200,25, hWnd, EditID, hInstance, NULL
mov hwndEdit, eax
invocar SetFocus, hwndEdit
invocar CreateWindowEx, NULL, ButtonClassName ADDR, \
ADDR ButtonText, \
WS_CHILD o WS_VISIBLE o BS_DEFPUSHBUTTON, \
75,70,140,25, hWnd, buttonId, hInstance, NULL
mov hwndButton, eax
```

Creamos los controles durante el procesamiento del mensaje WM_CREATE. Hacemos un llamado CreateWindowEx con un estilo de ventana extra, WS_EX_CLIENTEDGE, lo que hace que el área de cliente se ven hundidos. El nombre de cada control es uno predefinido, "editar" para editar el control, "botón" para el botón de control. A continuación especificar los estilos de la ventana de niño. Cada control tiene estilos adicionales además de los estilos de ventana normales. Por ejemplo, los estilos tienen el prefijo "BS_" para "estilo de botón", los estilos de edición llevan el prefijo "ES_" por "el estilo de edición". Hay que mirar estos estilos en una referencia de la API Win32. Tenga en cuenta que poner un identificador de control en el lugar de la manija del menú. Esto no causa ningún daño desde un control de la ventana niño no puede tener un menú.

Después de crear cada control, mantenemos el mango en una variable para su uso futuro.

SetFocus está llamado a dar el foco de entrada al cuadro de edición para que el usuario puede escribir el texto en él inmediatamente.

Ahora viene la parte realmente emocionante. Cada control de ventana hija envía una notificación a su ventana padre con WM_COMMAND.

```
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. SI lParam == 0
```

Recordemos que un menú también envía mensajes WM_COMMAND para notificar a la ventana de su estado también. ¿Cómo se puede diferenciar entre mensajes WM_COMMAND originó a partir de un menú o un control? A continuación se muestra la respuesta

	Palabra baja de wParam	Palabra alta de wParam	lParam
Menú	Menú ID	0	0
Control de	Control de Identificación	Notificación de código	Niño identificador de ventana

Se puede ver que usted debe comprobar lParam. Si es cero, la corriente mensaje WM_COMMAND es de un menú. No se puede usar wParam para diferenciar entre un menú y un control desde el menú de ID y el ID de control podrán ser idénticas y el código de notificación puede ser cero.

```
. Si ax == IDM_HELLO
invocar SetWindowText, hwndEdit, ADDR TestString
. ELSEIF ax == IDM_CLEAR
invocar SetWindowText, hwndEdit, NULL
. ELSEIF ax == IDM_GETTEXT
invocar GetWindowText, hwndEdit, tampón ADDR, 512
invocar el cuadro de mensaje, NULL, ADDR tampón, ADDR AppName, MB_OK
```

Usted puede poner una cadena de texto en un cuadro de edición llamando SetWindowText. Usted borrar el contenido de un cuadro de edición llamando SetWindowText con NULL. SetWindowText es una función de API propósito general. Usted puede utilizar SetWindowText para cambiar el título de una ventana o el texto de un botón.

Para obtener el texto en un cuadro de edición, se utiliza GetWindowText.

```
. Si ax == buttonId
SHR EAX, 16
. Si ax == BN_CLICKED
invoca SendMessage, hWnd, WM_COMMAND, IDM_GETTEXT, 0
. ENDIF
. ENDIF
```

El siguiente fragmento de código anterior se refiere a la condición cuando el usuario presiona el botón. En primer lugar, se comprueba la palabra baja de wParam para ver si el identificador de control coincide con el del botón. Si es así, comprueba la palabra alta de wParam para ver si es el código de notificación BN_CLICKED que se envía cuando se pulsa el botón.
La parte interesante es cuando ya es seguro que el código de notificación se BN_CLICKED. Queremos que el texto de la caja de edición y lo mostrará en un cuadro de mensaje. Se puede duplicar el código de la sección IDM_GETTEXT arriba, pero no tiene sentido. Si de alguna manera podemos enviar un mensaje WM_COMMAND con la palabra baja de wParam contiene el IDM_GETTEXT valor a nuestro procedimiento de ventana, podemos evitar la duplicación de código y simplificar nuestro programa. Función SendMessage es la respuesta. Esta función envía cualquier mensaje a cualquier ventana con cualquier wParam y lParam que queremos. Así que en lugar de duplicar el código, que llamamos SendMessage con la manija de la ventana padre, WM_COMMAND, IDM_GETTEXT, y 0. Esto tiene un efecto idéntico al seleccionar "Obtener texto" del menú en el menú. El procedimiento de ventana no percibe ninguna diferencia entre los dos.
Usted debe utilizar esta técnica tanto como sea posible para hacer que su código sea más organizado.
Por último, pero no menos importante, no se olvide la función TranslateMessage en el bucle de mensajes. Puesto que usted debe escribir algún texto en el cuadro de edición, el programa debe traducir la entrada de teclado en bruto en texto legible. Si se omite esta función, usted no será capaz de escribir nada en la caja de edición

TUTORIAL 10: CUADRO DE DIÁLOGO COMO VENTANA PRINCIPAL

Ahora viene la parte realmente interesante de interfaz gráfica de usuario, el cuadro de diálogo. En este tutorial (y la siguiente), vamos a aprender cómo utilizar un cuadro de diálogo como la ventana principal.

Descargue el primer ejemplo aquí , el segundo ejemplo aquí .

TEORÍA:

Si juegas con los ejemplos en el tutorial anterior el tiempo suficiente, usted encontrará que usted no puede cambiar el foco de entrada de un control de ventana hija a otra con la tecla Tab. La única manera de hacerlo es haciendo clic en el control que desea que gane el foco de entrada. Esta situación es bastante engorroso. Otra cosa que usted puede notar es que he cambiado el color de fondo de la ventana principal de gris en lugar de blanco normal como en los ejemplos anteriores. Esto se hace para que el color de los controles secundarios de la ventana pueden combinar a la perfección con el color del área de cliente de la ventana principal. Hay una manera de sortear este problema, pero no es fácil. Usted tiene que subclase todos los controles de ventana hija en su ventana principal.
La razón por la cual existe tal inconveniente es que los controles secundarios de ventana se diseñó originalmente para trabajar con un cuadro de diálogo no, una ventana normal. El color por defecto de la ventana de los controles secundarios, tales como un botón es de color gris debido a que el área de cliente de un cuadro de diálogo es normalmente gris, por lo que se mezclan entre sí sin ningún tipo de sudor por parte del programador.
Antes de adentrarse en los detalles, debemos saber primero qué es un cuadro de diálogo. Un cuadro de diálogo no es más que una ventana normal, que está diseñado para trabajar con controles de ventana hija. Windows también proporciona interna "gestor de cuadro de diálogo",

que es responsable de la mayor parte de la lógica del teclado, tales como cambiar el foco de entrada cuando el usuario presiona TAB, al pulsar el botón por defecto si se pulsa la tecla Enter, etc por lo que los programadores pueden hacer frente a tareas de nivel superior. Los cuadros de diálogo se utilizan principalmente como dispositivos de entrada / salida. Como tal, un cuadro de diálogo puede ser considerado como una entrada / salida de "recuadro negro", que significa que usted no tiene que saber cómo funciona un cuadro de diálogo interno con el fin de ser capaces de usar, sólo tienes que saber cómo interactuar con él. Eso es un principio de la programación orientada a objetos (POO) llama ocultación de información. Si el cuadro negro es * perfectamente * diseñada, el usuario puede hacer uso de ella sin ningún tipo de conocimiento sobre cómo funciona. El problema es que el cuadro negro tiene que ser perfecto, que es difícil de lograr en el mundo real. API de Win32 también está diseñado como un cuadro negro también.

Bueno, parece que nos desviamos de nuestro camino. Volvamos a nuestro tema. Los cuadros de diálogo están diseñados para reducir la carga de trabajo de un programador. Normalmente, si pones los controles secundarios de ventana en una ventana normal, usted tiene que subclase ellos y escribir la lógica del teclado usted mismo. Pero si usted los pone en un cuadro de diálogo, que se encargará de la lógica para usted. Sólo tienes que saber cómo conseguir la entrada del usuario desde el cuadro de diálogo o la forma de enviar comandos a la misma.

Un cuadro de diálogo se define como un recurso de la misma manera como un menú. Usted escribe una plantilla de cuadro de diálogo que describe las características del cuadro de diálogo y sus controles y luego compilar la secuencia de comandos de recursos con un editor de recursos.

Tenga en cuenta que todos los recursos se ponen juntos en el archivo de recursos misma secuencia de comandos. Se puede utilizar cualquier editor de texto para escribir una plantilla de cuadro de diálogo pero yo no lo recomiendo. Usted debe usar un editor de recursos para hacer el trabajo visualmente, ya que la organización de los controles secundarios de la ventana en un cuadro de diálogo es difícil de hacer manualmente. Varios editores de recursos excelentes que están disponibles. La mayoría de las suites de compiladores más importantes incluyen sus editores de recursos propios. Usted las puede utilizar para crear un script de recursos para su programa y luego cortar las líneas irrelevantes, tales como los relacionados con MFC.

Hay dos tipos principales de cuadro de diálogo: modal y no modal. Un cuadro de diálogo modal permite cambiar el foco de entrada a la otra ventana. El ejemplo es el cuadro de diálogo Buscar de MS Word. Hay dos subtipos de cuadro de diálogo modal: modal aplicación y modal del sistema. Una aplicación de cuadro de diálogo modal no le permiten cambiar el foco de entrada a otra ventana en la misma aplicación pero se puede cambiar el foco de entrada a la ventana de otra **aplicación.** Un cuadro de diálogo modal del sistema no le permiten cambiar el foco de entrada a cualquier otra ventana hasta que responder a ella en primer lugar.

Un cuadro de diálogo modal se crea llamando CreateDialogParam función de la API. Un cuadro de diálogo modal se crea llamando DialogBoxParam. La única diferencia entre un cuadro de diálogo modal y aplicación de un modal del sistema uno es el estilo DS_SYSMODAL. Si se incluye el estilo DS_SYSMODAL en una plantilla de cuadro de diálogo, ese cuadro de diálogo será un referente de sistema.

Usted puede comunicarse con cualquier control de ventana hija en un cuadro de diálogo mediante el uso de la función SendDlgItemMessage. Su sintaxis es la siguiente:

**SendDlgItemMessage proto hwndDlg: DWORD, **
**idControl: DWORD, **
**uMsg: DWORD, **
**wParam: DWORD, **
lParam: DWORD

Esta llamada a la API es inmensamente útil para interactuar con un control de ventana hija. Por ejemplo, si desea obtener el texto de un control de edición, usted puede hacer esto:

llamada SendDlgItemMessage, hDlg, ID_EDITBOX, WM_GETTEXT, 256, ADDR text_buffer

Para saber cuál es el mensaje a enviar, usted debe consultar a su referencia de la API de Win32.

Windows también ofrece varios específicos de control de funciones de la API para obtener y establecer datos de forma rápida, por ejemplo, GetDlgItemText, CheckDlgButton etc Estas funciones específicas de control se proporcionan para la conveniencia del programador para

que no se tienen que buscar los significados de wParam y lParam de cada mensaje. Normalmente, usted debe utilizar el control de llamadas específicas de la API cuando están disponibles, ya que hacen que el mantenimiento del código fuente más fácil. El recurso a la SendDlgItemMessage sólo si no hay llamadas de control específicas de la API están disponibles.

El cuadro de diálogo de Windows gerente envía algunos mensajes a una función de devolución de llamada especial llamado un procedimiento de diálogo que tiene el siguiente formato:

**DlgProc proto hDlg: DWORD, **
**lMSG: DWORD, **
**wParam: DWORD, **
lParam: DWORD

El procedimiento de diálogo es muy similar a un procedimiento de ventana, excepto para el tipo de valor de retorno que es verdadero / falso en lugar de LRESULT. El gerente del cuadro de diálogo interno dentro de **Windows** es el procedimiento de ventana válido para el cuadro de diálogo. Se llama a nuestro procedimiento de cuadro de diálogo con algunos de los mensajes que ha recibido. Así que la regla general es que: si nuestro procedimiento de diálogo procesa un mensaje, **debe** retornar TRUE en eax y si no procesa el mensaje, debe devolver FALSE en eax. Tenga en cuenta que un procedimiento de diálogo no pasa los mensajes que no se procesan a la llamada DefWindowProc ya que no es un procedimiento de ventana real.

Hay dos usos distintos de un cuadro de diálogo. Se puede utilizar como la ventana principal de la aplicación o utilizarlo como un dispositivo de entrada. Vamos a examinar la primera aproximación en este tutorial.

"El uso de un cuadro de diálogo como ventana principal" puede interpretarse en dos sentidos diferentes.

1. Puede utilizar la plantilla de cuadro de diálogo como una plantilla de clase que se registra en el llamado RegisterClassEx. En este caso, el cuadro de diálogo se comporta como una persona "normal" de la ventana: recibe mensajes a través de un procedimiento de ventana a que se refiere al miembro lpfnWndProc de la clase de ventana, no a través de un procedimiento de diálogo. La ventaja de este enfoque es que usted no tiene que crear los controles secundarios de la ventana a ti mismo, Windows los crea para usted cuando el cuadro de diálogo se crea. También en que Windows maneja la lógica de teclado para usted, tales como orden de tabulación, etc Además, usted puede especificar el cursor y el icono de la ventana en la estructura de clase de ventana.

   Su programa sólo crea el cuadro de diálogo sin crear ninguna ventana padre. Este enfoque hace un bucle de mensajes innecesaria, ya que los mensajes se envían directamente al procedimiento de diálogo. Ni siquiera tiene que registrarse una clase de ventana!

Este tutorial va a ser larga. Voy a presentar el primer enfoque seguido por el segundo.

Ejemplos:

---

dialog.asm

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
```

```
. Datos
ClassName db "DLGCLASS", 0
MenuName db "MiMenú", 0
DlgName db "MyDialog", 0
AppName db "Nuestro primer cuadro de diálogo", 0
TestString db "Wow, estoy en un cuadro de edición ahora", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
tampón db 512 dup (?)

Const.
IDC_EDIT equ 3000
IDC_BUTTON equ 3001
IDC_EXIT equ 3002
IDM_GETTEXT equ 32000
IDM_CLEAR equ 32001
IDM_EXIT equ 32002

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hDlg: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, DLGWINDOWEXTRA
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_BTNFACE un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateDialogParam, hInstance, DlgName ADDR, NULL, NULL, NULL
mov hDlg, eax
invocar ShowWindow, hDlg, SW_SHOWNORMAL
invocar UpdateWindow, hDlg
invocar GetDlgItem, hDlg, IDC_EDIT
invocar SetFocus, eax
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar IsDialogMessage, hDlg, ADDR msg
```

```
. SI eax == FALSO
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDIF
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. SI uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. SI lParam == 0
. Si ax == IDM_GETTEXT
invocar GetDlgItemText, hWnd, IDC_EDIT, tampón ADDR, 512
invocar el cuadro de mensaje, NULL, ADDR tampón, ADDR AppName, MB_OK
. ELSEIF ax == IDM_CLEAR
invocar SetDlgItemText, hWnd, IDC_EDIT, NULL
. MÁS
invocar DestroyWindow, hWnd
. ENDIF
. MÁS
mov edx, wParam
shr edx, 16
. SI dx == BN_CLICKED
. Si ax == IDC_BUTTON
invocar SetDlgItemText, hWnd, IDC_EDIT, ADDR TestString
. ELSEIF ax == IDC_EXIT
invoca SendMessage, hWnd, WM_COMMAND, IDM_EXIT, 0
. ENDIF
. ENDIF
. ENDIF
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Dialog.rc

```
# Include "resource.h"

# Define IDC_EDIT 3000
# Define IDC_BUTTON 3001
# Define IDC_EXIT 3002

# Define IDM_GETTEXT 32000
# Define IDM_CLEAR 32001
# Define IDM_EXIT 32003

MyDialog DIALOG 10, 10, 205, 60
ESTILO 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
```

```
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
TITULO "Nuestro primer cuadro de diálogo"
CLASE "DLGCLASS"
COMENZAR
EDITTEXT IDC_EDIT, 15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
DEFPUSHBUTTON "Say Hello", IDC_BUTTON, 141,10,52,13
PUSHBUTTON "& Salir", IDC_EXIT, 141,26,52,13, WS_GROUP
FIN

MiMenú MENÚ
COMENZAR
POPUP "controles de prueba"
COMENZAR
MENUITEM "Obtener texto", IDM_GETTEXT
MENUITEM "texto claro", IDM_CLEAR
MENUITEM "",, 0x0800 / * MFT_SEPARATOR * /
MENUITEM "& Salir", IDM_EXIT
FIN
FIN
```

Análisis:

Vamos a analizar este primer ejemplo.
Este ejemplo muestra cómo registrar una plantilla de diálogo como una clase de ventana y crear una "ventana" de esa clase. Se simplifica su programa ya que no tiene que crear la ventana secundaria se controla.
Primero vamos a analizar la plantilla de diálogo.

MyDialog DIALOG 10, 10, 205, 60

Declarar el nombre de un cuadro de diálogo, en este caso, "MyDialog", seguido de la palabra "diálogo". Los siguientes cuatro números son: x, y, ancho y alto del cuadro de diálogo en las unidades del cuadro de diálogo (no es el mismo en forma de píxeles).

ESTILO 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK

Declarar los estilos del cuadro de diálogo.

TITULO "Nuestro primer cuadro de diálogo"

Este es el texto que aparecerá en la barra de título del cuadro de diálogo.

CLASE "DLGCLASS"

Esta línea es crucial. Es esta palabra clave **CLASS,** que nos permite utilizar la plantilla de cuadro de diálogo como una clase de ventana. A raíz de la palabra clave es el nombre de la "clase de ventana"

```
COMENZAR
EDITTEXT IDC_EDIT, 15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
DEFPUSHBUTTON "Say Hello", IDC_BUTTON, 141,10,52,13
PUSHBUTTON "& Salir", IDC_EXIT, 141,26,52,13
FIN
```

El bloque anterior define los controles de ventana hija en el cuadro de diálogo. Están definidos entre las palabras clave **BEGIN** y **END.** En general, la sintaxis es la siguiente:

de control de tipo "texto", controlID, x, y, ancho, alto, [estilos]

control de los tipos son constantes del compilador de recursos de lo que tiene que consultar el manual.

Ahora vamos al código fuente de la Asamblea. La parte interesante es en la estructura de clase de ventana:

mov wc.cbWndExtra, DLGWINDOWEXTRA
wc.lpszClassName mov, ClassName OFFSET

Normalmente, este miembro se deja NULL, pero si queremos registrar una plantilla de cuadro de diálogo como una clase de ventana, tenemos que establezca este miembro en la **DLGWINDOWEXTRA** valor. Tenga en cuenta que el nombre de la clase debe ser idéntica a la que después de la palabra clave **CLASS** en la plantilla del cuadro de diálogo. Los miembros restantes se inicializan como de costumbre. Después de llenar la estructura de clase de ventana, que se registra en el RegisterClassEx. Parece familiar? Esta es la misma rutina que tiene que hacer para registrar una clase de ventana normal.

invocar CreateDialogParam, hInstance, DlgName ADDR, NULL, NULL, NULL

Después de registrar la "clase de ventana", creamos nuestra ventana de diálogo. En este ejemplo, se crea como un cuadro de diálogo modal con función CreateDialogParam. Esta función toma 5 parámetros, pero sólo tiene que rellenar los dos primeros: el identificador de instancia y el puntero al nombre de la plantilla de cuadro de diálogo. Tenga en cuenta que el parámetro de segundo no es un puntero al nombre de la clase.

En este punto, el cuadro de diálogo y sus controles de ventana hija son creados por Windows. El procedimiento de ventana recibirá el mensaje WM_CREATE, como de costumbre.

invocar GetDlgItem, hDlg, IDC_EDIT
invocar SetFocus, eax

Después de que el cuadro de diálogo se crea, quiero poner el foco de entrada al control de edición. Si pongo estos códigos en la sección WM_CREATE, llamada GetDlgItem fallará ya que en ese momento, los controles de ventana hija no se crean todavía. La única manera de hacerlo es llamar después de que el cuadro de diálogo y todos sus controles de ventana hija son creados. Así que puse estas dos líneas después de la llamada UpdateWindow. Función GetDlgItem obtiene el identificador de control y devuelve el identificador del control asociado a la ventana. Así es como se puede obtener un identificador de ventana si se conoce su identificador de control.

```
invocar IsDialogMessage, hDlg, ADDR msg
. Si eax == FALSO
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDIF
```

El programa entra en el bucle de mensajes y antes de que traducir y enviar mensajes, llamamos a la función IsDialogMessage para que el cuadro de diálogo Administrador se encarga de la lógica del teclado de nuestro cuadro de diálogo para nosotros. Si esta función devuelve TRUE, significa que el mensaje va dirigido a la caja de diálogo y es procesada por el administrador de cuadro de diálogo. Tenga en cuenta otra diferencia con el tutorial anterior. Cuando el procedimiento de ventana quiere obtener el texto del control de edición, se llama a la función GetDlgItemText lugar de GetWindowText. GetDlgItemText acepta un identificador de control en lugar de un identificador de ventana. Eso hace que la llamada más fácil en el caso de que utilice un cuadro de diálogo.

Ahora vamos a ir a la segunda aproximación a la utilización de un cuadro de diálogo como ventana principal. En el siguiente ejemplo, voy a crear una aplicación de cuadro de diálogo modal. Usted no encontrará un bucle de mensajes o de un procedimiento de ventana, ya que no es necesario!

---

dialog.asm (parte 2)

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno

DlgProc proto: DWORD,: DWORD,: DWORD,: DWORD

include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

. Datos
DlgName db "MyDialog", 0
AppName db "Nuestro segundo cuadro de diálogo", 0
TestString db "Wow, estoy en un cuadro de edición ahora", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
tampón db 512 dup (?)

Const.
IDC_EDIT equ 3000
IDC_BUTTON equ 3001
IDC_EXIT equ 3002
IDM_GETTEXT equ 32000
IDM_CLEAR equ 32001
IDM_EXIT equ 32002

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar DialogBoxParam, hInstance, DlgName ADDR, NULL, addr DlgProc, NULL
invocar ExitProcess, eax

DlgProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Sl uMsg == WM_INITDIALOG
invocar GetDlgItem, hWnd, IDC_EDIT
invocar SetFocus, eax
. ELSEIF uMsg == WM_CLOSE
invoca SendMessage, hWnd, WM_COMMAND, IDM_EXIT, 0
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Sl lParam == 0
. Si ax == IDM_GETTEXT
invocar GetDlgItemText, hWnd, IDC_EDIT, tampón ADDR, 512
invocar el cuadro de mensaje, NULL, ADDR tampón, ADDR AppName, MB_OK
. ELSEIF ax == IDM_CLEAR
invocar SetDlgItemText, hWnd, IDC_EDIT, NULL
. ELSEIF ax == IDM_EXIT
invocar EndDialog, hWnd, NULL
. ENDIF
. MÁS
mov edx, wParam
shr edx, 16
. Si dx == BN_CLICKED
```

```
. Si ax == IDC_BUTTON
invocar SetDlgItemText, hWnd, IDC_EDIT, ADDR TestString
. ELSEIF ax == IDC_EXIT
invoca SendMessage, hWnd, WM_COMMAND, IDM_EXIT, 0
. ENDIF
. ENDIF
. ENDIF
. MÁS
mov eax, FALSO
ret
. ENDIF
mov eax, TRUE
ret
DlgProc endp
poner fin a empezar a
```

---

dialog.rc (parte 2)

```
# Include "resource.h"

# Define IDC_EDIT 3000
# Define IDC_BUTTON 3001
# Define IDC_EXIT 3002

# Define IDR_MENU1 3003

# Define IDM_GETTEXT 32000
# Define IDM_CLEAR 32001
# Define IDM_EXIT 32003

MyDialog DIALOG 10, 10, 205, 60
ESTILO 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
TITULO "Nuestro segundo cuadro de diálogo"
MENÚ IDR_MENU1
COMENZAR
EDITTEXT IDC_EDIT, 15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
DEFPUSHBUTTON "Say Hello", IDC_BUTTON, 141,10,52,13
PUSHBUTTON "& Salir", IDC_EXIT, 141,26,52,13
FIN

IDR_MENU1 MENÚ
COMENZAR
POPUP "controles de prueba"
COMENZAR
MENUITEM "Obtener texto", IDM_GETTEXT
MENUITEM "texto claro", IDM_CLEAR
MENUITEM "",, 0x0800 / * MFT_SEPARATOR * /
MENUITEM "& Salir", IDM_EXIT
FIN
FIN
```

El análisis siguiente:

DlgProc proto: DWORD,: DWORD,: DWORD,: DWORD

Declaramos que el prototipo de función para DlgProc por lo que pueda referirse a ella con el operador **addr** en la línea de abajo:

invocar DialogBoxParam, hInstance, DlgName ADDR, NULL, addr DlgProc, NULL

La línea anterior llama a la función DialogBoxParam que tiene 5 parámetros: el identificador de instancia, el nombre de la plantilla de cuadro de diálogo, la manija de la ventana principal, la dirección del procedimiento de diálogo, y los datos específicos de diálogo. DialogBoxParam crea un cuadro de diálogo modal. No volverá hasta que el cuadro de diálogo se destruye.

```
. Si uMsg == WM_INITDIALOG
invocar GetDlgItem, hWnd, IDC_EDIT
invocar SetFocus, eax
. ELSEIF uMsg == WM_CLOSE
invoca SendMessage, hWnd, WM_COMMAND, IDM_EXIT, 0
```

El procedimiento de diálogo se parece a un procedimiento de ventana, excepto que no recibe el mensaje WM_CREATE. El primer mensaje que recibe es WM_INITDIALOG. Normalmente, usted puede poner el código de inicialización aquí. Tenga en cuenta que debe devolver el valor TRUE en eax si se procesa el mensaje.

El gerente del cuadro de diálogo interno no enviar a nuestro procedimiento de diálogo del mensaje WM_DESTROY por defecto cuando WM_CLOSE se envía a nuestro cuadro de diálogo. Así que si queremos reaccionar cuando el usuario presiona el botón de cierre en nuestro cuadro de diálogo, que debe procesar el mensaje WM_CLOSE. En nuestro ejemplo, podemos enviar un mensaje WM_COMMAND con la IDM_EXIT valor en wParam. Esto tiene el mismo efecto que cuando el usuario selecciona el punto de salida del menú. EndDialog es llamado en respuesta a la IDM_EXIT.

El procesamiento de los mensajes WM_COMMAND sigue siendo el mismo.

Cuando quiera destruir el cuadro de diálogo, la única manera es llamar a la función EndDialog. No trate de DestroyWindow! EndDialog no destruye el cuadro de diálogo de inmediato. Sólo se establece un indicador para el gerente del cuadro de diálogo interno y continúa ejecutando las siguientes instrucciones.

Ahora vamos a examinar el archivo de recursos. El cambio notable es que en lugar de utilizar una cadena de texto como nombre de menú se utiliza un valor, IDR_MENU1. Esto es necesario si desea adjuntar un menú a un cuadro de diálogo creado con DialogBoxParam. Tenga en cuenta que en la plantilla del cuadro de diálogo, hay que añadir el **menú** palabra clave seguida por el ID de recurso de menú.

Una diferencia entre los dos ejemplos de este tutorial que siempre se puede observar es la falta de un icono en el último ejemplo. Sin embargo, puede establecer el icono enviando el mensaje a WM_SETICON el cuadro de diálogo durante el WM_INITDIALOG.

TUTORIAL 11: MÁS SOBRE EL CUADRO DE DIÁLOGO

Vamos a aprender más sobre el cuadro de diálogo en este tutorial. En concreto, vamos a explorar el tema de cómo utilizar boxs de diálogo como nuestros dispositivos de entrada-salida. Si usted lee el tutorial anterior, ésta será una brisa, ya que sólo una pequeña modificación es todo lo que se necesita para ser capaz de utilizar los cuadros de diálogo como adjuntos a la ventana principal. También en este tutorial, vamos a aprender cómo utilizar cuadros de diálogo comunes.

Descargar los ejemplos del cuadro de diálogo aquí y aquí . Descargar ejemplo común cuadro de diálogo aquí .

TEORÍA:

Muy poco se puede decir acerca de cómo utilizar los cuadros de diálogo como dispositivos de entrada y salida de nuestro programa. El programa crea la ventana principal como de costumbre y cuando se desea mostrar el cuadro de diálogo, simplemente llame a CreateDialogParam o DialogBoxParam. Con una llamada DialogBoxParam, usted no tiene que hacer nada más, sólo procesar los mensajes en el procedimiento de diálogo. Con CreateDialogParam, debe insertar llamada IsDialogMessage en el bucle de mensajes para que el cuadro de diálogo Administrador de manejar la navegación con el teclado en el cuadro de diálogo para usted. Puesto que los dos casos son triviales, no me voy a poner el código fuente aquí. Puede descargar los ejemplos y examinar usted mismo, aquí y aquí .

Vamos a pasar a los cuadros de diálogo comunes. Windows ha elaborado cuadros de diálogo predefinidos para su uso por las aplicaciones. Estos cuadros de diálogo que permitan ofrecer la interfaz de usuario estándar. Se componen de archivos, impresión, color, fuente, y los cuadros de búsqueda de diálogo. Debe usarlos tanto como sea posible. Los cuadros de diálogo residen en comdlg32.dll. Con el fin de usarlos, usted tiene que enlazar con comdlg32.lib. Puede crear estos cuadros de diálogo llamando a las funciones correspondientes en la biblioteca de diálogo común. Para abrir el archivo de diálogo, es GetOpenFileName, de diálogo Guardar como es GetSaveFileName, de diálogo de impresión se PrintDlg y así sucesivamente. Cada una de estas funciones toma un puntero a una estructura como su parámetro. Usted debe mirar hacia arriba en la referencia de la API de Win32. En este tutorial, voy a demostrar cómo crear y utilizar un cuadro de diálogo de abrir archivo.

A continuación se muestra el prototipo de función de la función GetOpenFileName:

GetOpenFileName proto lpofn: DWORD

Se puede ver que sólo recibe un parámetro, un puntero a una estructura OPENFILENAME. El valor de retorno es TRUE significa que el usuario selecciona un archivo para abrirlo, es de lo contrario. Vamos a mirar en la estructura OPENFILENAME siguiente.

OPENFILENAME STRUCT
lStructSize DWORD?
HWND hwndOwner?
HINSTANCE hInstance?
lpstrFilter LPCSTR?
lpstrCustomFilter LPSTR?
nMaxCustFilter DWORD?
nFilterIndex DWORD?
lpstrFile LPSTR?
nMaxFile DWORD?
lpstrFileTitle LPSTR?
nMaxFileTitle DWORD?
lpstrInitialDir LPCSTR?
lpstrTitle LPCSTR?
DWORD de los indicadores?
PALABRA nFileOffset?
PALABRA nFileExtension?
lpstrDefExt LPCSTR?
lCustData LPARAM?
lpfnHook DWORD?
lpTemplateName LPCSTR?
OPENFILENAME TERMINA

Veamos el significado de los miembros de uso frecuente.

lStructSize	El tamaño de la estructura OPENFILENAME, en bytes
hwndOwner	El identificador de ventana del cuadro de diálogo Abrir.
hInstance	Instancia mango de la aplicación que crea el cuadro de diálogo Abrir archivo
lpstrFilter	Las cadenas de filtro en el formato de los pares de cadenas nulos terminados. La primera cadena en cada par es la descripción. La segunda cadena es el patrón del filtro. por ejemplo: FilterString db "Todos los archivos (*. *)", 0, "*. *", 0 db "archivos de texto (*. txt)", 0, "*. txt", 0,0

	Tenga en cuenta que sólo el patrón de la segunda cadena en cada par es en realidad utilizado por Windows para filtrar los archivos. Igualmente, señaló que hay que poner un 0 extra al final de las cadenas de filtro para indicar el final de la misma.
nFilterIndex	Especifique qué par de las cadenas de filtro se utilizó inicialmente en el cuadro de diálogo de abrir archivo se muestra por primera vez. El índice es 1-basado, que es el primer par es 1, el segundo par es 2 y así sucesivamente. Así que en el ejemplo anterior, si se especifica nFilterIndex como 2, el segundo patrón, "*. Txt" se utilizará.
lpstrFile	Puntero al buffer que contiene el nombre del archivo utilizado para inicializar el control de nombre de archivo de edición en el cuadro de diálogo. El tampón debe ser de al menos 260 bytes de longitud. Después de que el usuario selecciona un archivo para abrirlo, el nombre del archivo con la ruta completa se almacena en el buffer. Se puede extraer la información de la misma más adelante.
nMaxFile	El tamaño de la memoria intermedia lpstrFile.
lpstrTitle	Puntero al título del cuadro de diálogo Abrir archivo
Banderas	Determinar los estilos y las características del cuadro de diálogo.
nFileOffset	Después de que el usuario selecciona un archivo para abrir, este miembro contiene el índice del primer carácter del nombre de archivo real. Por ejemplo, si el nombre completo con la ruta de acceso es "c: \ windows \ system \ lz32.dll", el miembro de este contendrá el valor 18.
nFileExtension	Después de que el usuario selecciona un archivo para abrir, este miembro contiene el índice del primer carácter de la extensión de archivo

Ejemplo:

El siguiente programa muestra un cuadro de diálogo Abrir archivo cuando el usuario selecciona Archivo-> Abrir en el menú. Cuando el usuario selecciona un archivo en el cuadro de diálogo, el programa muestra un cuadro de mensaje muestra el nombre completo, nombre de archivo y la extensión del archivo seleccionado.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib

Const.
IDM_OPEN un equ
IDM_EXIT equ 2
MAXSIZE equ 260
Outputsize equ 512

. Datos
ClassName db "SimpleWinClass", 0
AppName db "Nuestra ventana principal", 0
MenuName db "FirstMenu", 0
OFN OPENFILENAME <>
FilterString db "Todos los archivos", 0, "*. *", 0
```

```
 db "Archivos de texto", 0, "*. txt", 0,0
 tampón db MAXSIZE dup (0)
 OurTitle db "- = Primer Nuestro cuadro de diálogo Abrir archivo = -: Seleccione el
archivo para abrir", 0
 Db FullPathName "el nombre completo con la ruta es:", 0
 FullName db "El Nombre es:", 0
 ExtensionName db "La extensión es la siguiente:", 0
 OutputString db outputsize dup (0)
 CrLf db 0Dh, 0Ah, 0

 . Datos?
 HINSTANCE hInstance?
 LPSTR de línea de comandos?

 . Código
 empezar:
 invocar GetModuleHandle, NULL
 mov hInstance, eax
 invocar GetCommandLine
 CommandLine mov, eax
 WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
 invocar ExitProcess, eax

 WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
 LOCAL wc: WNDCLASSEX
 Msg LOCAL: MSG
 LOCAL hwnd: HWND
 mov wc.cbSize, sizeof WNDCLASSEX
 mov wc.style, CS_HREDRAW o CS_VREDRAW
 mov wc.lpfnWndProc, OFFSET WndProc
 mov wc.cbClsExtra, NULL
 mov wc.cbWndExtra, NULL
 impulsar hInst
 pop wc.hInstance
 mov wc.hbrBackground, COLOR_WINDOW un
 mov wc.lpszMenuName, MenuName OFFSET
 wc.lpszClassName mov, ClassName OFFSET
 invocar LoadIcon, NULL, IDI_APPLICATION
 mov wc.hIcon, eax
 mov wc.hIconSm, eax
 invocar LoadCursor, NULL, IDC_ARROW
 wc.hCursor mov, eax
 invocar RegisterClassEx, addr wc
 invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
 WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
 CW_USEDEFAULT, 300.200, NULL, NULL, \
 hInst, NULL
 mov hwnd, eax
 invocar ShowWindow, hwnd, SW_SHOWNORMAL
 invocar UpdateWindow, hwnd
 . MIENTRAS VERDADERO
 invocar GetMessage, msg ADDR, NULL, 0,0
 . INTER. If (! Eax)
 invocar TranslateMessage, ADDR msg
 invocar DispatchMessage, ADDR msg
 . ENDW
 mov eax, msg.wParam
 ret
 WinMain endp
```

```
WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. SI uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si ax == IDM_OPEN
mov ofn.lStructSize, sizeof OFN
impulsar hWnd
ofn.hwndOwner pop
impulsar hInstance
pop ofn.hInstance
mov ofn.lpstrFilter, FilterString OFFSET
mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, MAXSIZE
mov ofn.Flags, OFN_FILEMUSTEXIST o \
OFN_PATHMUSTEXIST o OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
mov ofn.lpstrTitle, OFFSET OurTitle
invocar GetOpenFileName, ADDR OFN
. Si eax == TRUE
invocar lstrcat, OutputString offset, FullPathName OFFSET
invocar lstrcat, OutputString offset, ofn.lpstrFile
invocar lstrcat, OutputString offset, desplazamiento CrLf
invocar lstrcat, OutputString offset, desplazamiento FullName
mov eax, ofn.lpstrFile
PUSH EBX
xor ebx, ebx
mov bx, ofn.nFileOffset
añadir EAX, EBX
pop ebx
invocar lstrcat, compensado OutputString, eax
invocar lstrcat, OutputString offset, desplazamiento CrLf
invocar lstrcat, OutputString offset, desplazamiento ExtensionName
mov eax, ofn.lpstrFile
PUSH EBX
xor ebx, ebx
mov bx, ofn.nFileExtension
añadir EAX, EBX
pop ebx
invocar lstrcat, compensado OutputString, eax
invocar el cuadro de mensajes, hWnd, OutputString OFFSET, ADDR AppName, MB_OK
invocar RtlZeroMemory, OutputString offset, outputsize
. Endif
. Más
invocar DestroyWindow, hWnd
. Endif
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

mov ofn.lStructSize, sizeof OFN
impulsar hWnd
ofn.hwndOwner pop
impulsar hInstance
pop ofn.hInstance

Llenamos de los miembros de la rutina de las estructuras de n.

mov ofn.lpstrFilter, FilterString OFFSET

Este FilterString es el filtro de nombre de archivo que se especifica de la siguiente manera:

FilterString db "Todos los archivos", 0, "*. *", 0
db "Archivos de texto", 0, "*. txt", 0,0

Tenga en cuenta que las cuatro cuerdas se terminada en cero. La primera cadena es la descripción de la cadena siguiente. El patrón actual es la cadena de número par, en este caso, "*. *" Y "*. Txt". En realidad, se puede especificar cualquier patrón que queremos aquí. Tenemos que poner un cero después de la cadena último patrón para indicar el final de la cadena de filtro. No se olvide de esta otra cosa que su cuadro de diálogo se comportan de manera extraña.

mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, MAXSIZE

Aquí especificamos dónde está el cuadro de diálogo le puso el nombre de archivo que el usuario selecciona. Tenga en cuenta que debe especificar su tamaño en el miembro nMaxFile. Más tarde se puede extraer el nombre del archivo de este búfer.

**mov ofn.Flags, OFN_FILEMUSTEXIST o **
**OFN_PATHMUSTEXIST o OFN_LONGNAMES o **
OFN_EXPLORER o OFN_HIDEREADONLY

Banderas especifica las características del cuadro de diálogo.
OFN_FILEMUSTEXIST OFN_PATHMUSTEXIST y banderas de la demanda de que el nombre del archivo y la ruta que el usuario escribe en el control de nombre de archivo de edición debe existir.
OFN_LONGNAMES bandera le dice el cuadro de diálogo para mostrar los nombres de archivo largos.
OFN_EXPLORER bandera especifica que la aparición del cuadro de diálogo debe ser similar al Explorador.
OFN_HIDEREADONLY bandera esconde la casilla de verificación de sólo lectura en el cuadro de diálogo.
Hay banderas de muchos más que se pueden utilizar. Consulte a su referencia de la API de Win32.

mov ofn.lpstrTitle, OFFSET OurTitle

Especifique el título del cuadro de diálogo.

invocar GetOpenFileName, ADDR OFN

Llame a la función GetOpenFileName. Al pasar el puntero a la estructura de n como parámetro.
En este momento, el cuadro de diálogo Abrir archivo se muestra en la pantalla. La función no regresará hasta que el usuario selecciona un archivo para abrir o pulsa el botón de cancelar o cerrar el cuadro de diálogo.

Se va a devolver el valor TRUE en eax si el usuario selecciona un archivo para abrirlo. Devuelve FALSE de lo contrario.

. Si eax == TRUE
invocar lstrcat, OutputString offset, FullPathName OFFSET
invocar lstrcat, OutputString offset, ofn.lpstrFile
invocar lstrcat, OutputString offset, desplazamiento CrLf
invocar lstrcat, OutputString offset, desplazamiento FullName

En caso de que el usuario selecciona un archivo para abrirlo, nos preparamos una cadena de salida que se mostrará en un cuadro de mensaje. Asignamos un bloque de memoria en la variable OutputString y luego usamos una función de la API, lstrcat, para concatenar las cadenas juntas. Con el fin de poner las cadenas en varias líneas, tenemos que separar cada línea con un par de retorno de carro y avance de línea.

mov eax, ofn.lpstrFile
PUSH EBX
xor ebx, ebx
mov bx, ofn.nFileOffset
añadir EAX, EBX
pop ebx
invocar lstrcat, compensado OutputString, eax

Las líneas anteriores requieren una explicación. nFileOffset contiene el índice en la ofn.lpstrFile. Pero no se pueden sumar directamente desde nFileOffset es una variable de tamaño word y lpstrFile es un valor DWORD de tamaño. Así que tengo que poner el valor de nFileOffset en la palabra baja de EBX y agregarlo al valor de lpstrFile.

invocar el cuadro de mensajes, hWnd, OutputString OFFSET, ADDR AppName, MB_OK

Mostramos la cadena en un cuadro de mensaje.

invocar RtlZeroIMemory, OutputString offset, outputsize

Debemos * clara * la OutputString antes de que podamos completar en otra cadena. Por lo tanto, utilizar la función de RtlZeroMemory para hacer el trabajo

TUTORIAL 12: MEMORIA DE GESTIÓN Y ARCHIVO DE E / S

Vamos a aprender de la rudimentaria de administración de memoria y presentar la operación de E / S en este tutorial. Además vamos a utilizar cuadros de diálogo comunes como dispositivos de entrada-salida.

Bajar el ejemplo aquí .

TEORÍA:

Gestión de memoria en Win32 desde el punto de la aplicación de vista es bastante simple y directo. Cada proceso posee una memoria de 4 GB de espacio de direcciones. El modelo de memoria utilizada se llama modelo de memoria plana. En este modelo, todos los registros de

segmento (o selectores) apuntan a la misma dirección de partida y el desplazamiento es de 32 bits para una aplicación puede acceder a la memoria en cualquier punto de su propio espacio de direcciones sin necesidad de cambiar el valor de los selectores. Esto simplifica la gestión de memoria un montón. No hay ningún puntero "cerca" o "mucho" más.
Bajo Win16, hay dos categorías principales de funciones de la API de memoria: global y local. Mundial de tipo llamadas a la API se ocupan de la memoria asignada en otros segmentos por lo que está "lejos" funciones de la memoria. Local tipo de llamadas a la API acuerdo con el montón local del proceso por lo que son "cerca de" funciones de la memoria. Bajo Win32, estos dos tipos son idénticos. Ya sea que usted llame a GlobalAlloc o LocalAlloc, se obtiene el mismo resultado.
Pasos en la asignación y uso de la memoria son los siguientes:

1. Asignar un bloque de memoria llamando a **GlobalAlloc.** Esta función devuelve un identificador para el bloque de memoria solicitado.
2. "Bloquear" el bloque de memoria llamando **GlobalLock.** Esta función acepta un identificador para el bloque de memoria y devuelve un puntero al bloque de memoria.
3. Usted puede utilizar el puntero para leer o escribir en la memoria.
4. "Desbloquear" el bloque de memoria llamando **GlobalUnlock.** Esta función invalida el puntero al bloque de memoria.
5. Liberación de los bloque de memoria llamando **GlobalFree.** Esta función acepta el identificador para el bloque de memoria.

También puede sustituir "global" por "local" como LocalAlloc, LocalLock, etc
El método anterior se puede simplificar mediante el uso de una bandera en la llamada GlobalAlloc, GMEM_FIXED. Si se utiliza esta opción, el valor de retorno de LocalAlloc Global / será el puntero al bloque de memoria asignada, no el mango de bloque de memoria. Usted no tiene que llamar a Global / LocalLock y usted puede pasar el puntero a Global / LocalFree sin llamar a Global / LocalUnlock primero. Sin embargo, en este tutorial, voy a utilizar el enfoque "tradicional" ya que puede surgir al leer el código fuente de otros programas.

File I / O en Win32 tiene semejanza notable que bajo DOS. Los pasos necesarios son los mismos. Usted sólo tiene que cambiar las interrupciones de llamadas a la API y ya está. Los pasos requeridos son los siguientes:

1. Abrir o crear el archivo llamando a la función **CreateFile.** Esta función es muy versátil: además de los archivos, puede abrir puertos de comunicaciones, tuberías, unidades de disco o de la consola. En caso de éxito, devuelve un identificador de archivo o dispositivo. A continuación, puede utilizar este identificador para realizar operaciones en el archivo o dispositivo.
   Mueva el puntero del archivo en la ubicación deseada llamando **SetFilePointer.**
2. Realizar lectura o escritura llamando a **ReadFile** o **WriteFile.** Estas funciones de transferencia de datos desde un bloque de memoria hacia o desde el archivo. Así que hay que asignar un bloque de memoria lo suficientemente grande para contener los datos.
3. Cierre el archivo llamando a **CloseHandle.** Esta función acepta el identificador de archivo.

Contenido:

El programa que aparece a continuación muestra un cuadro de diálogo Abrir archivo. Le permite al usuario seleccionar un archivo de texto para abrir y muestra el contenido de ese archivo en un control de edición en su área de cliente. El usuario puede modificar el texto en el control de edición como desee, y puede optar por guardar el contenido en un archivo.

0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

```
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib

Const.
IDM_OPEN un equ
IDM_SAVE equ 2
IDM_EXIT equ 3
MAXSIZE equ 260
Memsize equ 65535

EditID equ 1; ID del control de edición

. Datos
ClassName db "Win32ASMEditClass", 0
AppName db "Win32 ASM Editar", 0
EditClass db "editar", 0
MenuName db "FirstMenu", 0
OFN OPENFILENAME <>
FilterString db "Todos los archivos", 0, "*. *", 0
db "Archivos de texto", 0, "*. txt", 0,0
tampón db MAXSIZE dup (0)

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwndEdit HWND? , Identifica el control de edición
hFile HANDLE? , El gestor de archivo
HRanura MANGO? ; El mango para el bloque de memoria asignada
pMemory DWORD? ; Puntero al bloque de memoria asignada
SizeReadWrite DWORD? , El número de bytes realmente leer o escribir

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
SDWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
```

```
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc utiliza ebx hWnd: HWND, uMsg: UINT, wParam: wParam, lParam:
LPARAM
. Sl uMsg == WM_CREATE
invocar CreateWindowEx, NULL, ADDR EditClass, NULL, \
WS_VISIBLE o WS_CHILD o ES_LEFT o ES_MULTILINE o \
ES_AUTOHSCROLL o ES_AUTOVSCROLL, 0, \
0,0,0, hWnd, EditID, \
hInstance, NULL
mov hwndEdit, eax
invocar SetFocus, hwndEdit
; ================================================
; Inicialice los miembros de la estructura OPENFILENAME
; ================================================
mov ofn.lStructSize, sizeof OFN
impulsar hWnd
ofn.hWndOwner pop
impulsar hInstance
pop ofn.hInstance
mov ofn.lpstrFilter, FilterString OFFSET
mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, MAXSIZE
. ELSEIF uMsg == WM_SIZE
mov eax, lParam
mov edx, eax
shr edx, 16
y EAX, 0FFFFh
invocar MoveWindow, hwndEdit, 0,0, eax, edx, TRUE
. ELSEIF uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_OPEN
mov ofn.Flags, OFN_FILEMUSTEXIST o \
OFN_PATHMUSTEXIST o OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
```

```
invocar GetOpenFileName, ADDR OFN
. Si eax == TRUE
invocar CreateFile, ADDR tampón, \
GENERIC_READ o GENERIC_WRITE, \
FILE_SHARE_READ o FILE_SHARE_WRITE, \
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, \
NULL
mov hFile, eax
invocar GlobalAlloc, GMEM_MOVEABLE o GMEM_ZEROINIT, memsize
mov HRanura, eax
invocar GlobalLock, HRanura
mov pMemory, eax
invocar ReadFile, hFile, pMemory, memsize-1, ADDR SizeReadWrite, NULL
invoca SendMessage, hwndEdit, WM_SETTEXT, NULL, pMemory
invocar CloseHandle, hFile
invocar GlobalUnlock pMemory
invocar GlobalFree, HRanura
. Endif
invocar SetFocus, hwndEdit
. Elseif ax == IDM_SAVE
mov, los ofn.Flags OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
invocar GetSaveFileName, ADDR OFN
. Si eax == TRUE
invocar CreateFile, ADDR tampón, \
GENERIC_READ o GENERIC_WRITE, \
FILE_SHARE_READ o FILE_SHARE_WRITE, \
NULL, CREATE_NEW, FILE_ATTRIBUTE_ARCHIVE, \
NULL
mov hFile, eax
invocar GlobalAlloc, GMEM_MOVEABLE o GMEM_ZEROINIT, memsize
mov HRanura, eax
invocar GlobalLock, HRanura
mov pMemory, eax
invoca SendMessage, hwndEdit, WM_GETTEXT, memsize-1, pMemory
invocar WriteFile, hFile, pMemory, eax, ADDR SizeReadWrite, NULL
invocar CloseHandle, hFile
invocar GlobalUnlock pMemory
invocar GlobalFree, HRanura
. Endif
invocar SetFocus, hwndEdit
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

```
invocar CreateWindowEx, NULL, ADDR EditClass, NULL, \
WS_VISIBLE o WS_CHILD o ES_LEFT o ES_MULTILINE o \
```

```
ES_AUTOHSCROLL o ES_AUTOVSCROLL, 0, \
0,0,0, hWnd, EditID, \
hInstance, NULL
mov hwndEdit, eax
```

En la sección WM_CREATE, creamos un control de edición. Tenga en cuenta que los parámetros que especifican x, y, anchura, altura del control son todos los ceros ya que vamos a cambiar el tamaño del control posterior para cubrir el área cliente de la ventana principal.
Tenga en cuenta que en este caso, no tenemos que llamar a ShowWindow para hacer el control de edición aparecerá en la pantalla, ya que incluyen el estilo WS_VISIBLE. Usted puede usar este truco en la ventana principal también.

```
; ===========================================
; Inicialice los miembros de la estructura OPENFILENAME
; ===========================================
mov ofn.lStructSize, sizeof OFN
impulsar hWnd
ofn.hWndOwner pop
impulsar hInstance
pop ofn.hInstance
mov ofn.lpstrFilter, FilterString OFFSET
mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, MAXSIZE
```

Después de crear el control de edición, podemos aprovechar este momento para inicializar los miembros de la OFN. Porque queremos volver a utilizar de n en los cuadro de diálogo también se cumplimente sólo la * común * miembros that're utilizado tanto por GetOpenFileName y GetSaveFileName.
La sección WM_CREATE es un gran lugar para hacer una sola vez la inicialización.

```
. ELSEIF uMsg == WM_SIZE
mov eax, lParam
mov edx, eax
shr edx, 16
y EAX, 0FFFFh
invocar MoveWindow, hwndEdit, 0,0, eax, edx, TRUE
```

Recibimos mensajes WM_SIZE cuando el tamaño del área de cliente de nuestros cambios de la ventana principal. También se reciben cuando la ventana se crea por primera vez. Con el fin de poder recibir este mensaje, los estilos de clase de ventana debe incluir CS_VREDRAW y CS_HREDRAW estilos. Usamos esta oportunidad para cambiar el tamaño de nuestro control de edición para el mismo tamaño que el área de cliente de la ventana principal. En primer lugar tenemos que saber la anchura y la altura del área de cliente de la ventana principal. Recibimos esta información de lParam. La palabra alta de lParam contiene la altura y la palabra baja de lParam el ancho del área de cliente. A continuación, utilizar la información para cambiar el tamaño del control de edición llamando a la función MoveWindow que, además de cambiar la posición de la ventana, puede alterar el tamaño también.

```
. Si ax == IDM_OPEN
mov ofn.Flags, OFN_FILEMUSTEXIST o \
OFN_PATHMUSTEXIST o OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
invocar GetOpenFileName, ADDR OFN
```

Cuando el usuario selecciona Archivo / Abrir elemento de menú, que complete el miembro Flags de la estructura de n y llamar a la función GetOpenFileName para mostrar el cuadro de diálogo Abrir archivo.

. Si eax == TRUE
**invocar CreateFile, ADDR tampón, **
**GENERIC_READ o GENERIC_WRITE, **
**FILE_SHARE_READ o FILE_SHARE_WRITE, **
**NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, **
NULL
mov hFile, eax

Después de que el usuario selecciona un archivo para abrir, llamamos a CreateFile para abrir el archivo. Nos indica que la función debe tratar de abrir el archivo para leer y escribir. Después de que el archivo se abre, la función devuelve el identificador para el archivo abierto que guardamos en una variable global para su uso futuro. Esta función tiene la siguiente sintaxis:

**CreateFile proto lpFileName: DWORD, **
**dwDesiredAccess: DWORD, **
**dwShareMode: DWORD, **
**lpSecurityAttributes: DWORD, **
dwCreationDistribution: DWORD \,
dwFlagsAndAttributes: DWORD \,
hTemplateFile: DWORD

dwDesiredAccess especifica la operación que desea realizar en el archivo.

- **0** Abra el archivo para consultar sus atributos. Usted tiene los derechos para leer o escribir los datos.
- **GENERIC_READ** Abra el archivo para su lectura.
- **GENERIC_WRITE** Abra el archivo para escribir.

dwShareMode especifica la operación que desea permitir que otros procesos para llevar a cabo en el archivo que 's que se abre.

- **0** No compartir el archivo con otros procesos.
- **FILE_SHARE_READ** permitir otros procesos para leer los datos del archivo que se abre
- **FILE_SHARE_WRITE** permitir que otros procesos para escribir datos en el archivo que se abrió.

lpSecurityAttributes no tiene ningún significado en Windows 95.
dwCreationDistribution especifica el CreateFile acción se realiza cuando el archivo especificado en lpFileName existe o cuando no existe.

- **CREATE_NEW** Crea un nuevo archivo. La función falla si el archivo especificado ya existe.
- **CREATE_ALWAYS** Crea un nuevo archivo. La función sobrescribe el archivo, si existe.
- **OPEN_EXISTING** Abre el archivo. La función falla si el archivo no existe.
- **OPEN_ALWAYS** Abre el archivo, si es que existe. Si el archivo no existe, la función crea el archivo como si fuera dwCreationDistribution CREATE_NEW.
- **TRUNCATE_EXISTING** Abre el archivo. Una vez abierto, el archivo se trunca de manera que su tamaño es de cero bytes. El proceso de llamada debe abrir el archivo con acceso GENERIC_WRITE por lo menos. La función falla si el archivo no existe.

dwFlagsAndAttributes especifica los atributos de archivo

- **FILE_ATTRIBUTE_ARCHIVE** El archivo es un archivo histórico. Las aplicaciones utilizan este atributo para marcar los archivos para copia de seguridad o de eliminación.

- **FILE_ATTRIBUTE_COMPRESSED** El archivo o directorio está comprimido. Para un archivo, esto significa que todos los datos en el archivo está comprimido. Para un directorio, esto significa que la compresión es el predeterminado para los archivos recién creados y subdirectorios.
- **FILE_ATTRIBUTE_NORMAL** El archivo no tiene un conjunto de otros atributos. Este atributo sólo es válido si se utiliza solo.
- **FILE_ATTRIBUTE_HIDDEN** El archivo está oculto. No es para ser incluidos en una lista de directorios ordinario.
- **FILE_ATTRIBUTE_READONLY** El archivo es de sólo lectura. Las aplicaciones pueden leer el archivo pero no puede escribir en él o eliminarlo.
- **FILE_ATTRIBUTE_SYSTEM** El archivo es parte de, o es utilizada exclusivamente por el sistema operativo.

invocar GlobalAlloc, GMEM_MOVEABLE o GMEM_ZEROINIT, memsize
mov HRanura, eax
invocar GlobalLock, HRanura
mov pMemory, eax

Cuando se abre el archivo, se asignará un bloque de memoria para el uso de las funciones ReadFile y WriteFile. Se especifica la bandera GMEM_MOVEABLE para permitir que Windows mover el bloque de memoria en torno a la consolidación de la memoria. Bandera de GMEM_ZEROINIT dice GlobalAlloc para llenar el bloque de memoria recién asignada con ceros.
Cuando GlobalAlloc devuelve correctamente, eax contiene el identificador en el bloque de memoria asignada. Pasamos este identificador para la función GlobalLock que devuelve un puntero al bloque de memoria.

invocar ReadFile, hFile, pMemory, memsize-1, ADDR SizeReadWrite, NULL
invoca SendMessage, hwndEdit, WM_SETTEXT, NULL, pMemory

Cuando el bloque de memoria está listo para su uso, que llamamos ReadFile función para leer los datos del archivo. Cuando un archivo se abrió por primera vez o creado, el puntero de archivo se encuentra en desplazamiento 0. Así, en este caso, se comienza la lectura desde el primer byte de los archivos en adelante. El primer parámetro de ReadFile es el identificador del archivo a leer, el segundo es el puntero al bloque de memoria para almacenar los datos, el siguiente es el número de bytes a leer desde el archivo, el cuarto parámetro es la dirección de la variable de tamaño DWORD que se llenará con el número de bytes realmente lee desde el archivo.
Después de llenar el bloque de memoria con los datos, ponemos los datos en el control de edición mediante el envío de mensaje WM_SETTEXT el control de edición con lParam contiene el puntero al bloque de memoria. Después de esta llamada, el control de edición muestra los datos en su área de cliente.

invocar CloseHandle, hFile
invocar GlobalUnlock pMemory
invocar GlobalFree, HRanura
. Endif

En este punto, no tenemos necesidad de mantener el archivo abierto por más tiempo ya que nuestro propósito es escribir los datos modificados desde el control de edición a otro archivo, no el archivo original. Así que cerramos el archivo llamando a CloseHandle con el identificador de archivo como parámetro. A continuación abrimos el bloque de memoria y liberarla. En realidad, usted no tiene que liberar la memoria en este momento, puede volver a utilizar el bloque de memoria durante la operación de almacenamiento más adelante. Sin embargo, para fines de demostración, elijo liberar aquí.

invocar SetFocus, hwndEdit

Cuando aparezca el cuadro de diálogo Abrir archivo se muestra en la pantalla, el foco de entrada se desplaza a la misma. Así que después de que el diálogo de abrir archivo está cerrado, hay que mover el foco de entrada de nuevo al control de edición.
Este extremo de la operación de lectura en el archivo. En este punto, el usuario puede editar el contenido de la edición control.And cuando quiere guardar los datos en otro archivo, debe seleccionar Archivo / Guardar como menuitem que muestra un cuadro de diálogo Guardar como. La creación de los cuadro de diálogo no es muy diferente en el cuadro de diálogo Abrir archivo. De hecho, se diferencian sólo en el nombre de las funciones, GetOpenFileName y GetSaveFileName. Puede volver a utilizar la mayoría de los miembros de la estructura de n demasiado, excepto el miembro Flags.

**mov, los ofn.Flags OFN_LONGNAMES o **
OFN_EXPLORER o OFN_HIDEREADONLY

En nuestro caso, queremos crear un nuevo archivo, por lo que OFN_FILEMUSTEXIST y OFN_PATHMUSTEXIST debe quedar fuera de lo contrario el cuadro de diálogo no nos permitirá crear un archivo que no existe ya.
El parámetro dwCreationDistribution de la función CreateFile se debe cambiar para **CREATE_NEW** ya que queremos crear un nuevo archivo.
El resto del código es idéntico a los de la sección de archivo abierto, excepto los siguientes:

invoca SendMessage, hwndEdit, WM_GETTEXT, memsize-1, pMemory
invocar WriteFile, hFile, pMemory, eax, ADDR SizeReadWrite, NULL

Enviamos mensaje WM_GETTEXT al control de edición para copiar los datos de la misma para el bloque de memoria que ofrecen, el valor devuelto en eax es la longitud de los datos contenidos en el búfer. Después de los datos están en el bloque de memoria, las escribimos en el nuevo archivo

TUTORIAL 13: ARCHIVOS ASIGNADOS EN MEMORIA

Te voy a mostrar lo que los archivos de memoria asignadas son y cómo usarlos para sus ventajas. El uso de un archivo de la memoria asignada es muy fácil ya que puedes encontrar en este tutorial.

TEORÍA:

Si usted examina el ejemplo en el tutorial anterior de cerca, verá que tiene un grave defecto: ¿y si el archivo que desea leer es más grande que el bloque de memoria asignada? o ¿qué pasa si la cadena que desea buscar se corta por la mitad al final del bloque de memoria? La respuesta tradicional a la primera pregunta es que usted debe leer varias veces en los datos desde el archivo hasta el final del archivo que se encuentre. La respuesta a la segunda pregunta es que usted debe prepararse para el caso especial en el final del bloque de memoria. Esto se llama un problema de contorno. Se presenta dolor de cabeza para los programadores y causa de innumerables fallos.
Sería bueno si podemos asignar un bloque muy grande de memoria, suficiente para almacenar todo el archivo, pero nuestro programa será un consumidor de recursos. Archivo de asignación para el rescate. Mediante el uso de asignación de archivos, usted puede pensar en la totalidad del expediente por estar ya cargado en memoria y se puede utilizar un puntero de memoria para leer o escribir datos desde el archivo. Tan fácil como eso. No es necesario utilizar la API de funciones de la memoria y de archivo por separado funciones I / O API más, son una y la

misma en virtud de asignación de archivos. Asignación de archivos también se utiliza como un medio para compartir datos entre procesos. Uso de la asignación del archivo de esta manera, no hay ningún archivo que existe realmente. Es más como un bloque de memoria reservado que todos los procesos * puede ver *. Sin embargo, el intercambio de datos entre los procesos es un tema delicado, que no debe tratarse a la ligera. Usted tiene que poner en práctica procesos y sincronización de subprocesos otra de sus aplicaciones se colgará en muy poco tiempo.

No vamos a tocar el tema de asignación de archivos como un medio para crear una región de memoria compartida en este tutorial. Nos concentraremos en cómo utilizar la asignación de archivo como un medio para "asignar" un archivo en la memoria. De hecho, el cargador PE utiliza asignación de archivos para cargar los archivos ejecutables en la memoria. Es muy conveniente ya que sólo las porciones necesarias pueden ser selectivamente lee desde el archivo en el disco. Bajo Win32, debe utilizar la asignación de archivo tanto como sea posible.

Hay algunas limitaciones para presentar la cartografía sin embargo. Una vez que se crea un archivo de la memoria asignada, su tamaño no puede ser cambiado durante ese período de sesiones. Por lo tanto la asignación de archivos es muy bueno para archivos de sólo lectura o de las operaciones de archivos que no afectan el tamaño del archivo. Eso no quiere decir que no se puede utilizar la asignación de archivo si desea aumentar el tamaño del archivo. Se puede estimar el tamaño de la nueva y crear el archivo de la memoria asignada en función del tamaño del archivo nuevo y la voluntad de crecer hasta ese tamaño. Es incómodo, eso es todo.

Suficiente para la explicación. ¡Entremos en la aplicación de asignación de archivos. Con el fin de utilizar la asignación de archivo, estos pasos se deben realizar:

1. llamar a **CreateFile** para abrir el archivo que desee asignar.
2. llame **CreateFileMapping** con el identificador de archivo devuelto por CreateFile como uno de sus parámetros. Esta función crea un objeto de asignación de archivo del expediente abierto por CreateFile.
3. llame **MapViewOfFile** para mapear una región archivo seleccionado o todo el archivo a la memoria. Esta función devuelve un puntero al primer byte de la región archivo asignado.
4. Utilice el puntero para leer o escribir en el archivo
5. llame **UnmapViewOfFile** Para desasignar el archivo.
6. llamar a **CloseHandle** con el mango en el fichero de mapeado como el parámetro para cerrar el archivo asignado.
7. llamar a **CloseHandle** de nuevo esta vez con el archivo devuelto por CreateFile para cerrar el archivo real.

EJEMPLO:

El programa se enumeran a continuación le permite abrir un archivo a través de un cuadro de diálogo Abrir archivo. Se abre el archivo con el archivo de asignación, si es exitoso, el título de la ventana se cambia por el nombre del archivo abierto. Puede guardar el archivo con otro nombre por el del archivo de selección / Guardar como menuitem. El programa copiará todo el contenido del archivo abierto en el nuevo archivo. Tenga en cuenta que usted no tiene que llamar a GlobalAlloc para asignar un bloque de memoria en este programa.

```
0.386
. Modelo plano, stdcall
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib
```

```
Const.
IDM_OPEN un equ
IDM_SAVE equ 2
IDM_EXIT equ 3
MAXSIZE equ 260

. Datos
ClassName db "Win32ASMFileMappingClass", 0
AppName db "Win32 ASM Ejemplo de asignación de archivo", 0
MenuName db "FirstMenu", 0
OFN OPENFILENAME <>
FilterString db "Todos los archivos", 0, "*. *", 0
db "Archivos de texto", 0, "*. txt", 0,0
tampón db MAXSIZE dup (0)
hMapFile MANGO 0; identifica el archivo de la memoria asignada, debe ser
; Inicializa con 0, ya que también lo utilizan como
; Una bandera en la sección WM_DESTROY demasiado

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hFileRead MANGO? , Identifica el archivo de origen
hFileWrite MANGO? , Identifica el archivo de salida
MANGO hMenu?
pMemory DWORD? ; Puntero a los datos en el archivo fuente
SizeWritten DWORD? , El número de bytes realmente escritos por WriteFile

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, \
ADDR AppName, WS_OVERLAPPEDWINDOW CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
```

```
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. SI uMsg == WM_CREATE
invocar GetMenu, hWnd, obtener el identificador del menú
mov hMenu, eax
mov ofn.lStructSize, sizeof OFN
impulsar hWnd
ofn.hWndOwner pop
impulsar hInstance
pop ofn.hInstance
mov ofn.lpstrFilter, FilterString OFFSET
mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, MAXSIZE
. ELSEIF uMsg == WM_DESTROY
. Si hMapFile! = 0
llame CloseMapFile
. Endif
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_OPEN
mov ofn.Flags, OFN_FILEMUSTEXIST o \
OFN_PATHMUSTEXIST o OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
invocar GetOpenFileName, ADDR OFN
. Si eax == TRUE
invocar CreateFile, ADDR tampón, \
GENERIC_READ, \
0, \
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, \
NULL
mov hFileRead, eax
invocar CreateFileMapping, hFileRead, NULL, PAGE_READONLY, 0,0, NULL
mov hMapFile, eax
mov eax, desplazamiento de búfer
movzx edx, ofn.nFileOffset
add eax, edx
invocar SetWindowText, hWnd, eax
invocar EnableMenuItem, hMenu, IDM_OPEN, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_SAVE, MF_ENABLED
. Endif
. Elseif ax == IDM_SAVE
mov, los ofn.Flags OFN_LONGNAMES o \
OFN_EXPLORER o OFN_HIDEREADONLY
invocar GetSaveFileName, ADDR OFN
. Si eax == TRUE
```

```
invocar CreateFile, ADDR tampón, \
GENERIC_READ o GENERIC_WRITE, \
FILE_SHARE_READ o FILE_SHARE_WRITE, \
NULL, CREATE_NEW, FILE_ATTRIBUTE_ARCHIVE, \
NULL
mov hFileWrite, eax
invocar MapViewOfFile, hMapFile, FILE_MAP_READ, 0,0,0
mov pMemory, eax
invocar GetFileSize, hFileRead, NULL
invocar WriteFile, hFileWrite, pMemory, eax, ADDR SizeWritten, NULL
invocar UnmapViewOfFile, pMemory
llame CloseMapFile
invocar CloseHandle, hFileWrite
invocar SetWindowText, hWnd, ADDR AppName
invocar EnableMenuItem, hMenu, IDM_OPEN, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_SAVE, MF_GRAYED
. Endif
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

CloseMapFile PROC
invocar CloseHandle, hMapFile
mov hMapFile, 0
invocar CloseHandle, hFileRead
ret
CloseMapFile endp

poner fin a empezar a
```

Análisis:

```
invocar CreateFile, ADDR tampón, \
GENERIC_READ, \
0, \
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, \
NULL
```

Cuando el usuario selecciona un archivo en el archivo de diálogo abierto, llamamos a CreateFile para abrirlo. Tenga en cuenta que se especifica GENERIC_READ para abrir este archivo para acceso de sólo lectura y dwShareMode es cero porque no queremos que algún otro proceso para modificar el archivo en nuestra operación.

```
invocar CreateFileMapping, hFileRead, NULL, PAGE_READONLY, 0,0, NULL
```

Entonces te llamamos para CreateFileMapping para crear un archivo de memoria asignada desde el archivo abierto. CreateFileMapping tiene la siguiente sintaxis:

```
CreateFileMapping proto hFile: DWORD, \
lpFileMappingAttributes: DWORD, \
```

```
flProtect: DWORD, \
dwMaximumSizeHigh: DWORD, \
dwMaximumSizeLow: DWORD, \
lpName: DWORD
```

Usted debe saber primero que CreateFileMapping no tiene que asignar el archivo completo a la memoria. Es posible utilizar esta función para asignar sólo una parte del archivo de efectivo a la memoria. Se especifica el tamaño del archivo de la memoria asignada en el params dwMaximumSizeHigh y dwMaximumSizeLow. Si se especifica el tamaño de eso es más grande que el archivo actual, el archivo real se ampliará para el nuevo tamaño. Si desea que el archivo de la memoria asignada a ser del mismo tamaño que el archivo actual, poner ceros en los dos parametros.
Usted puede utilizar NULL en el parámetro lpFileMappingAttributes para que Windows crea un archivo de la memoria asignada con los atributos de seguridad por defecto.
flProtect define la protección deseada para el archivo de la memoria asignada. En nuestro ejemplo, usamos PAGE_READONLY para que sólo permita la operación de lectura en el archivo de la memoria asignada. Tenga en cuenta que este atributo no debe contradecir el atributo utilizado en CreateFile demás CreateFileMapping fallará.
puntos lpName al nombre del archivo de la memoria asignada. Si desea compartir este archivo con otro proceso, debe proporcionar un nombre. Sin embargo, en nuestro ejemplo, nuestro proceso es el único que utiliza este archivo por lo que ignorar este parámetro.

```
mov eax, desplazamiento de búfer
movzx edx, ofn.nFileOffset
add eax, edx
invocar SetWindowText, hWnd, eax
```

Si CreateFileMapping tiene éxito, cambiar el título de la ventana con el nombre del archivo abierto. El nombre del archivo con la ruta completa se almacena en el buffer, que queremos mostrar sólo el nombre en el título, así que hay que añadir el valor del miembro nFileOffset de la estructura OPENFILENAME a la dirección de buffer.

```
invocar EnableMenuItem, hMenu, IDM_OPEN, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_SAVE, MF_ENABLED
```

Como medida de precaución, no queremos que el usuario abra varios archivos a la vez, por lo que en gris la opción de menú Abrir y permitir la opción de menú Guardar. EnableMenuItem se utiliza para cambiar el atributo de elemento de menú.
Después de esto, esperamos que el usuario seleccione Archivo / Guardar como del menú o cerrar nuestro programa. Si el usuario elige para cerrar nuestro programa, debemos cerrar el archivo de la memoria asignada y el archivo real, como el código de abajo:

```
. ELSEIF uMsg == WM_DESTROY
. Si hMapFile! = 0
llame CloseMapFile
. Endif
invocar PostQuitMessage, NULL
```

En el fragmento de código anterior, cuando el procedimiento de ventana recibe el mensaje WM_DESTROY, se comprueba el valor de hMapFile primer lugar, si es cero o no. Si no es cero, se llama a la función CloseMapFile que contiene el siguiente código:

```
CloseMapFile PROC
invocar CloseHandle, hMapFile
mov hMapFile, 0
invocar CloseHandle, hFileRead
ret
CloseMapFile endp
```

CloseMapFile cierra el archivo mapeado en memoria y el archivo real de modo que no va a haber ninguna fuga de recursos cuando nuestras salidas del programa para Windows.
Si el usuario opta por guardar los datos en otro archivo, el programa lo presenta con un cuadro de diálogo. Después de que él los tipos en el nombre del nuevo archivo, el archivo es creado por la función CreateFile.

invocar MapViewOfFile, hMapFile, FILE_MAP_READ, 0,0,0
mov pMemory, eax

Inmediatamente después del archivo de salida se crea, que nosotros llamamos MapViewOfFile para asignar la parte deseada del archivo de la memoria asignada en la memoria. Esta función tiene la siguiente sintaxis:

**MapViewOfFile proto hFileMappingObject: DWORD, **
**dwDesiredAccess: DWORD, **
**dwFileOffsetHigh: DWORD, **
**dwFileOffsetLow: DWORD, **
dwNumberOfBytesToMap: DWORD

dwDesiredAccess especifica la operación que queremos hacer en el archivo. En nuestro ejemplo, queremos leer los datos sólo por lo que usamos FILE_MAP_READ.
dwFileOffsetHigh dwFileOffsetLow y especificar el archivo de desplazamiento inicial de la parte del archivo que se desea asignar a la memoria. En nuestro caso, queremos leer en todo el archivo para empezar la cartografía de la compensación de 0 en adelante.
dwNumberOfBytesToMap especifica el número de bytes al mapa en la memoria. Si desea asignar el archivo completo (especificado por CreateFileMapping), pasar de 0 a MapViewOfFile.
Después de llamar a MapViewOfFile, la parte deseada se carga en memoria. Se le dará el puntero al bloque de memoria que contiene los datos del archivo.

invocar GetFileSize, hFileRead, NULL

Averigua qué tan grande es el archivo. El tamaño del archivo se devuelve en eax. Si el archivo es más de 4 GB, el valor DWORD de alta del tamaño del archivo se almacena en FileSizeHighWord. Puesto que no hay que esperar para manejar archivos de gran tamaño tal, que podemos ignorar.

invocar WriteFile, hFileWrite, pMemory, eax, ADDR SizeWritten, NULL

Escriba los datos que está asignado a la memoria en el archivo de salida.

invocar UnmapViewOfFile, pMemory

Cuando hayamos terminado con el archivo de entrada, desasignar de la memoria.

llame CloseMapFile
invocar CloseHandle, hFileWrite

Y cierra todos los archivos.

invocar SetWindowText, hWnd, ADDR AppName

Restaurar el texto del título original.

invocar EnableMenuItem, hMenu, IDM_OPEN, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_SAVE, MF_GRAYED

Habilitar la opción de menú abierta y en gris la opción Guardar como del menú.

TUTORIAL 14: PROCESO

Vamos a aprender lo que es un proceso y cómo crear y poner fin a la misma.

.

PRELIMINAR:

¿Qué es un proceso? Cito esta definición de la API Win32 de Referencia:
"Un proceso es una aplicación en ejecución que consiste en un espacio de direcciones privadas virtuales, código, datos y otros recursos del sistema operativo, tales como archivos, tuberías, y los objetos de sincronización que son visibles para el proceso".
Como se puede ver en la definición anterior, un proceso de "dueño" de varios objetos: el espacio de direcciones, el módulo de ejecución (s), y todo lo que los módulos de ejecución crear o abrir. Como mínimo, un proceso debe consistir en la ejecución de un módulo, un espacio de direcciones privadas y un hilo. Cada proceso debe tener al menos un hilo. ¿Qué es un hilo? Un hilo es en realidad una cola de ejecución. Cuando Windows crea en primer lugar un proceso, se crea sólo un hilo por proceso. Este hilo por lo general se inicia la ejecución de la primera instrucción en el módulo. Si el proceso más adelante necesita más temas, de manera explícita los puede crear.
Cuando Windows se recibe un comando para crear un proceso, se crea el espacio de direcciones de memoria privada para el proceso y luego se asigna el archivo ejecutable en el espacio. Después de que crea el hilo principal para el proceso.
Bajo Win32, también puede crear procesos a partir de sus propios programas llamando a la función CreateProcess. CreateProcess tiene la siguiente sintaxis:

```
CreateProcess proto lpApplicationName: DWORD, \
lpCommandLine: DWORD, \
lpProcessAttributes: DWORD, \
lpThreadAttributes: DWORD, \
bInheritHandles: DWORD, \
dwCreationFlags: DWORD, \
lpEnvironment: DWORD, \
lpCurrentDirectory: DWORD, \
lpStartupInfo: DWORD, \
lpProcessInformation: DWORD
```

No se alarme por el número de parámetros. Podemos ignorar la mayoría de ellos.

lpApplicationName -> El nombre del archivo ejecutable, con o sin nombre de ruta que se desea ejecutar. Si este parámetro es nulo, debe proporcionar el nombre del archivo ejecutable en el parámetro lpCommandLine.
lpCommandLine -> Los argumentos de línea de comandos para el programa que desea ejecutar. Tenga en cuenta que si el lpApplicationName es NULL, este parámetro debe contener el nombre del archivo ejecutable también. De esta manera: "notepad.exe readme.txt"
lpProcessAttributes y lpThreadAttributes -> Especifica los atributos de seguridad para el proceso y el hilo principal. Si son valores NULL, los atributos de seguridad por defecto se utilizan.
bInheritHandles -> Un indicador que especifique si desea que el nuevo proceso para heredar todos los identificadores abiertos de su proceso.
dwCreationFlags -> Varias banderas que determinan el comportamiento del proceso que se desea crear, como por ejemplo, se quiere procesar a ser creado, sino suspendido de inmediato para que pueda examinar o modificar antes de que se ejecuta? También puede especificar la clase de prioridad del hilo (s) en el nuevo proceso. Esta clase de prioridad se utiliza para determinar la prioridad de programación de los hilos dentro del proceso. Normalmente usamos

la bandera NORMAL_PRIORITY_CLASS.
lpEnvironment -> Un puntero al bloque de entorno que contiene varias cadenas de entorno para el nuevo proceso. Si este parámetro es NULL, el nuevo proceso hereda el bloque de entorno del proceso padre.
lpCurrentDirectory -> Puntero a la cadena que especifica la unidad actual y el directorio para el proceso hijo. NULL si desea que el proceso hijo hereda del proceso padre.
lpStartupInfo -> apunta a una estructura STARTUPINFO que especifica cómo la ventana principal para el nuevo proceso debería aparecer. La estructura STARTUPINFO contiene muchos miembros que especifica la apariencia de la ventana principal del proceso hijo. Si usted no quiere nada especial, usted puede llenar la estructura STARTUPINFO con los valores del proceso padre llamando a la función GetStartupInfo.
lpProcessInformation -> apunta a una estructura PROCESS_INFORMATION que recibe la información de identificación sobre el nuevo proceso. La estructura PROCESS_INFORMATION tiene los siguientes miembros:

PROCESS_INFORMATION STRUCT
hProcess MANGO? ; La manija para que el proceso hijo
hThread MANGO? ; Manejar el hilo principal del proceso hijo
dwProcessId DWORD? ; ID del proceso hijo
dwThreadId DWORD? ; Identificador del subproceso principal del proceso hijo
PROCESS_INFORMATION TERMINA

Identificador de proceso y el ID del proceso son dos cosas diferentes. Un identificador de proceso es un identificador único para el proceso en el sistema. Un identificador de proceso es un valor devuelto por Windows para su uso con otras funciones de la API relacionados con el proceso. Un identificador de proceso no puede ser utilizado para identificar un proceso ya que no es único.

Después de la llamada CreateProcess, un nuevo proceso se crea y la llamada CreateProcess regresar de inmediato. Puede comprobar si el nuevo proceso todavía está activo llamando a la función GetExitCodeProcess que tiene la siguiente sintaxis:

GetExitCodeProcess proto hProcess: DWORD, lpExitCode: DWORD

Si esta llamada tiene éxito, lpExitCode contiene el estado de terminación del proceso en cuestión. Si el valor es igual a lpExitCode **STILL_ACTIVE,** entonces ese proceso todavía se está ejecutando.

Puede obligar a terminar un proceso llamando a la función TerminateProcess. Se tiene la siguiente sintaxis:

TerminateProcess proto hProcess: DWORD, uExitCode: DWORD

Se puede especificar el código de salida deseada para el proceso, cualquier valor que desee. TerminateProcess no es una forma limpia para terminar un proceso, ya que cualquier dll, unido a que el proceso no se le notificará que el proceso terminó.

EJEMPLO:

En el siguiente ejemplo se crea un nuevo proceso cuando el usuario selecciona el "proceso de creación de" elemento de menú. Se tratará de ejecutar "msgbox.exe". Si el usuario desea dar por terminado el proceso de nuevo, puede seleccionar la opción "terminar el proceso" del menú. El programa comprobará primero si el nuevo proceso ya está destruido, si no lo es, el programa llama a la función TerminateProcess para destruir el nuevo proceso.

0.386
. Modelo plano, stdcall
casemap opción: ninguno

```
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

Const.
IDM_CREATE_PROCESS un equ
IDM_TERMINATE equ 2
IDM_EXIT equ 3

. Datos
ClassName db "Win32ASMProcessClass", 0
AppName db "Win32 ASM Ejemplo de proceso", 0
MenuName db "FirstMenu", 0
ProcessInfo PROCESS_INFORMATION <>
programname db "msgbox.exe", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
MANGO hMenu?
ExitCode DWORD? , Contiene el estado de código de salida del proceso de llamada
GetExitCodeProcess.

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
```

```
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
invocar GetMenu, hwnd
mov hMenu, eax
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL StartInfo: STARTUPINFO
. SI uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. UMsg ELSEIF == WM_INITMENUPOPUP
invocar GetExitCodeProcess, processInfo.hProcess, ADDR ExitCode
. Si eax == TRUE
. Si ExitCode == STILL_ACTIVE
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_ENABLED
. Más
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_GRAYED
. Endif
. Más
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_GRAYED
. Endif
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_CREATE_PROCESS
. Si processInfo.hProcess! = 0
invocar CloseHandle, processInfo.hProcess
mov processInfo.hProcess, 0
. Endif
invocar GetStartupInfo, ADDR StartInfo
invocar CreateProcess, programname ADDR, NULL, NULL, NULL, FALSE \
NORMAL_PRIORITY_CLASS, \
NULL, NULL, ADDR StartInfo ADDR ProcessInfo
invocar CloseHandle, processInfo.hThread
. Elseif ax == IDM_TERMINATE
invocar GetExitCodeProcess, processInfo.hProcess, ADDR ExitCode
. Si ExitCode == STILL_ACTIVE
invocar TerminateProcess, processInfo.hProcess, 0
. Endif
invocar CloseHandle, processInfo.hProcess
mov processInfo.hProcess, 0
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
```

```
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a
```

Análisis:

El programa crea la ventana principal y recupera la manija del menú para su uso futuro. A continuación, espera para que el usuario seleccione un comando desde el menú. Cuando el usuario selecciona "Proceso" del menú en el menú principal, procesamos mensaje WM_INITMENUPOPUP para modificar las opciones del menú en el menú emergente antes de que se muestra.

```
. UMsg ELSEIF == WM_INITMENUPOPUP
invocar GetExitCodeProcess, processInfo.hProcess, ADDR ExitCode
. Si eax == TRUE
. Si ExitCode == STILL_ACTIVE
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_ENABLED
. Más
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_GRAYED
. Endif
. Más
invocar EnableMenuItem, hMenu, IDM_CREATE_PROCESS, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_TERMINATE, MF_GRAYED
. Endif
```

¿Por qué queremos para procesar este mensaje? Porque queremos preparar a los elementos de menú en el menú emergente para que el usuario pueda verlas. En nuestro ejemplo, si el nuevo proceso no se ha iniciado, sin embargo, queremos permitir que el "proceso de arranque" y en gris los "terminar proceso" elementos de menú. Lo hacemos a la inversa, si el nuevo proceso ya está activo.

En primer lugar, comprobar si el nuevo proceso está aún en marcha llamando a la función GetExitCodeProcess con el identificador del proceso que fue completado por la función CreateProcess. Si GetExitCodeProcess devuelve FALSE, significa que el proceso no se ha iniciado todavía por lo que el gris "terminar el proceso" del menú. Si GetExitCodeProcess devuelve TRUE, sabemos que un nuevo proceso se ha iniciado, pero tenemos que comprobar aún más si aún se está ejecutando. Por lo tanto, comparar el valor en ExitCode a la **STILL_ACTIVE** valor, si son iguales, el proceso está todavía en marcha: debemos en gris el tema "inicio del proceso" en el menú ya que no desea iniciar varios procesos concurrentes.

```
. Si ax == IDM_CREATE_PROCESS
. Si processInfo.hProcess! = 0
invocar CloseHandle, processInfo.hProcess
mov processInfo.hProcess, 0
. Endif
invocar GetStartupInfo, ADDR StartInfo
invocar CreateProcess, programname ADDR, NULL, NULL, NULL, FALSE \
NORMAL_PRIORITY_CLASS, \
NULL, NULL, ADDR StartInfo ADDR ProcessInfo
invocar CloseHandle, processInfo.hThread
```

Cuando el usuario selecciona "proceso de inicio" del menú, en primer lugar comprobar si los miembros de la estructura hProcess PROCESS_INFORMATION ya está cerrado. Si esta es la primera vez, el valor de hProcess siempre será cero puesto que definen la estructura PROCESS_INFORMATION en. Sección de datos. Si el valor del miembro hProcess no es 0,

significa que el proceso hijo ha terminado, pero no hemos cerrado el proceso de manejar aún. Así que este es el momento de hacerlo.
Llamamos a la función GetStartupInfo para rellenar la estructura STARTUPINFO que vamos a pasar a la función CreateProcess. Después de eso que llamamos la función CreateProcess para iniciar el nuevo proceso. Tenga en cuenta que no he comprobado el valor de retorno de CreateProcess, ya que hará que el ejemplo más complejo. En la vida real, usted debe comprobar el valor de retorno de CreateProcess. Inmediatamente después de CreateProcess, se cierra el mango hilo primario devuelto en la estructura de ProcessInfo. Cerrando el mango no significa que terminará el hilo, sólo que no queremos usar el mango para referirse al hilo de nuestro programa. Si no se cierra, causará una pérdida de recursos.

```
. Elseif ax == IDM_TERMINATE
invocar GetExitCodeProcess, processInfo.hProcess, ADDR ExitCode
. Si ExitCode == STILL_ACTIVE
invocar TerminateProcess, processInfo.hProcess, 0
. Endif
invocar CloseHandle, processInfo.hProcess
mov processInfo.hProcess, 0
```

Cuando el usuario selecciona "terminar el proceso" del menú, se comprueba si el nuevo proceso todavía está activo llamando a la función GetExitCodeProcess. Si aún está activo, llamamos a la función TerminateProcess para matar el proceso. También se cierra el identificador del proceso hijo ya que no tenemos necesidad de ella nunca más.

TUTORIAL 15: PROGRAMACIÓN MULTIPROCESO

Vamos a aprender a crear un programa multihilo en este tutorial. También estudiamos los métodos de comunicación entre los hilos.

TEORÍA:

En el tutorial anterior, a aprender el proceso de un compuesto de al menos un hilo: el hilo principal. Un hilo es una cadena de ejecución. También puede crear subprocesos adicionales en su programa. Usted puede ver como la multitarea multithreading dentro de un programa. En términos de implementación, un hilo es una función que se ejecuta conjuntamente con el programa principal. Puede ejecutar varias instancias de la misma función o puede ejecutar varias funciones simultáneamente, dependiendo de sus necesidades. Multithreading es específica para Win32, Win16 existe ninguna contrapartida.
Temas se ejecutan en el mismo proceso para que puedan acceder a los recursos en el proceso, tales como variables globales, maneja, etc Sin embargo, cada hilo tiene su propia pila para las variables locales en cada hilo son privadas. Cada hilo también es propietaria de su registro privado situado de modo que cuando Windows cambia a otros hilos, el hilo puede "recordar" su estado anterior y puede "reanudar" la tarea cuando se hace con el control de nuevo. Esto se maneja internamente por Windows.
Podemos dividir en dos temas caterories:

1. Hilo de la interfaz de usuario: Este tipo de hilo crea su propia ventana para que reciba mensajes de Windows. Se puede responder a la información a través de su propia ventana de ahí el nombre. Este tipo de hilo está sujeto a la regla de Mutex de Win16 que permite sólo un hilo interfaz de usuario en usuario de 16 bits y el kernel de GDI. Mientras que un subproceso de la interfaz de usuario está ejecutando código de usuario de 16 bits y el kernel de GDI, otros subprocesos de interfaz de usuario no puede utilizar el servicio del usuario de 16 bits y el kernel de GDI. Tenga en cuenta que este Mutex de Win16 es específico de Windows 95 desde debajo, de Windows 95

funciones de la API thunk a código de 16 bits. Windows NT no tiene Mutex de Win16 para que los hilos de la interfaz de usuario en NT funciona más suavemente que en Windows 95.

2. Subproceso de trabajo: Este tipo de hilo no crea una ventana por lo que no puede recibir ningún mensaje de Windows. Existe principalmente para hacer el trabajo asignado en el fondo por lo tanto, el hilo el nombre del trabajador.

Te aconsejo la siguiente estrategia cuando se utiliza la capacidad multihilo de Win32: dejar que el hilo principal de hacer cosas de interfaz de usuario y los hilos de otros hacer el trabajo duro en el fondo. De esta manera, el hilo principal es como un Gobernador, otros hilos son como el personal del gobernador. Los delegados del Gobernador puestos de trabajo a su personal, mientras que mantiene contacto con el público. El personal del gobernador obediente realiza el trabajo e informa al Gobernador. Si el gobernador fuera a realizar todas las tareas él mismo, no sería capaz de dar mucha atención al público oa la prensa. Eso es similar a una ventana que está ocupado haciendo un trabajo muy largo en su hilo principal: no responde al usuario hasta que se complete el trabajo. Tal programa se pueden beneficiar de la creación de un hilo additonal que es responsable para el trabajo prolongado, permitiendo que el hilo principal para responder a los comandos del usuario.
Podemos crear un hilo llamando a la función CreateThread que tiene la siguiente sintaxis:

```
LpThreadAttributes CreateThread Proto: DWORD, \
dwStackSize: DWORD, \
                lpStartAddress: DWORD, \
lpParameter: DWORD, \
dwCreationFlags: DWORD, \
                lpThreadId: DWORD
```

Función CreateThread se parece mucho a CreateProcess.
lpThreadAttributes -> Puedes usar NULL si desea que el hilo tenga descriptor de seguridad predeterminado.
dwStackSize -> Especificar el tamaño de la pila de la rosca. Si desea que el hilo tenga el tamaño de pila lo mismo que el hilo principal, utilice NULL en este parámetro.
lpStartAddress -> Dirección de la rosca function.It 's de la función que realizará el trabajo de la rosca. Esta función debe recibir uno y sólo un parámetro de 32 bits y devuelven un valor de 32 bits.
lpParameter -> El parámetro que se desea pasar a la función del hilo.
dwCreationFlags -> 0 significa que el hilo se ejecuta inmediatamente después de su creación. Lo contrario se CREATE_SUSPENDED bandera.
lpThreadId -> La función CreateThread llenará el ID del hilo de la rosca de nueva creación en esta dirección.

Si la llamada es exitosa CreateThread, que devuelve el identificador del hilo de nueva creación. De lo contrario, devuelve NULL.
La función del hilo se ejecuta tan pronto como sea llamada CreateThread útil es el éxito a menos que especifique la bandera CREATE_SUSPENDED en dwCreationFlags. En ese caso, el hilo se suspende hasta que la función ResumeThread se llama.
Cuando la función del hilo regresa con instrucción ret, Windows llama a la función ExitThread para la función del hilo de manera implícita. Usted puede llamar a la función con ExitThread en su función de hilo de ti mismo, pero el punto no 's poco en hacerlo.
Usted puede recuperar el código de salida de un hilo llamando a la función GetExitCodeThread.
Si desea poner fin a un hilo de otro hilo, se puede llamar a la función TerminateThread. Pero usted debe utilizar esta función en virtud de circunstancias extremas ya que esta función termina el hilo de inmediato sin dar el hilo de cualquier posibilidad de limpiar después de sí mismo.

Ahora vamos a pasar a los métodos de comunicación entre hilos.
Hay tres de ellos:

- Uso de variables globales
- Mensajes de Windows
- Evento

Temas compartir los recursos del proceso, incluyendo las variables globales por lo que los temas pueden usar varibles globales para comunicarse entre sí. Sin embargo, este método debe usarse con cuidado. La sincronización de subprocesos debe entrar en consideración. Por ejemplo, si dos hilos de utilizar la misma estructura de 10 miembros, lo que sucede cuando de repente da un tirón de Windows el control de uno de los hilos cuando se encontraba en medio de la actualización de la estructura? El otro hilo se quedará con un conjunto de datos inconsistentes en la estructura! No te equivoques, los programas multihilo son más difíciles de depurar y mantener. Este tipo de error parece ocurrir al azar, que es muy difícil de rastrear.
También puede utilizar los mensajes de Windows para la comunicación entre hilos. Si los hilos son todos de interfaz de usuario, no hay problema: este método puede ser utilizado como una comunicación de dos vías. Todo lo que tienes que hacer es definir una o más ventanas de mensajes personalizados que son significativos para los hilos. Puede definir un mensaje personalizado mediante el uso de mensaje de WM_USER como el valor base por ejemplo, se puede definir así:

WM_MYCUSTOMMSG equ WM_USER 100 h

Windows no utilizará ningún valor a partir de WM_USER hacia arriba por sus propios mensajes para que pueda utilizar el WM_USER valor y por encima de su valor como propio mensaje personalizado.
Si uno de los que el hilo es un hilo de interfaz de usuario y el otro es un trabajador, usted no puede utilizar este método como la comunicación de dos vías desde un subproceso de trabajo no tiene su propia ventana para que no se tiene una cola de mensajes. Usted puede utilizar el siguiente esquema:

Tema Interfaz de usuario ------> variable global (s) ----> subproceso de trabajo
Subproceso de trabajo ------> Mensaje ventana personalizada (s) ----> User Interfaz Tema

De hecho, vamos a utilizar este método en nuestro ejemplo.
El método de la última comunicación es un objeto de evento. Puede ver un objeto de evento como una especie de bandera. Si el objeto de evento se encuentra en "unsignalled" del Estado, el hilo está dormido o para dormir, en este estado, el hilo no recibe rebanada de tiempo de CPU. Cuando el objeto de evento se encuentra en "señal" del Estado, de Windows "despierta" el hilo y comienza a realizar la tarea asignada.

EJEMPLO:

Usted debe descargar el archivo zip y ejecutar thread1.exe ejemplo. Haga clic en el "Cálculo Savage" del menú. Esto le indica al programa para realizar "add eax, eax" por 600.000.000 de veces. Tenga en cuenta que durante ese tiempo, usted no puede hacer nada con la ventana principal: no se puede mover, no se puede activar el menú, etc Cuando el cálculo se ha completado, aparecerá un mensaje. Después de que la ventana de su comando acepta con normalidad.
Para evitar este tipo de inconveniece para el usuario, que puede mover el "cálculo" de rutina en un subproceso de trabajo independiente y dejar que el hilo principal de continuar con su tarea de interfaz de usuario. Se puede ver que aunque la ventana principal responde más lentamente de lo usual, todavía responde

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
```

```
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

Const.
IDM_CREATE_THREAD un equ
IDM_EXIT equ 2
WM_FINISH equ WM_USER 100 h

. Datos
ClassName db "Win32ASMThreadClass", 0
AppName db "Win32 ASM Ejemplo Multithreading", 0
MenuName db "FirstMenu", 0
SuccessString db "El cálculo se ha completado!", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwnd MANGO?
ThreadID DWORD?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
```

```
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Sl uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_CREATE_THREAD
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
0, \
ADDR ThreadID
invocar CloseHandle, eax
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. ELSEIF uMsg == WM_FINISH
invocar el cuadro de mensaje, NULL, SuccessString ADDR, ADDR AppName, MB_OK
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

ThreadProc PROC USOS Param ecx: DWORD
mov ecx, 600000000
Loop1:
add eax, eax
diciembre ecx
jz Get_out
jmp Loop1
Get_out:
invocar PostMessage, hwnd, WM_FINISH, NULL, NULL
ret
ThreadProc ENDP

poner fin a empezar a
```

Análisis:

El programa principal presenta al usuario con una ventana normal con un menú. Si el usuario selecciona "Crear Tema" del menú, el programa crea un hilo de la siguiente manera:

```
. Si ax == IDM_CREATE_THREAD
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
```

La función anterior crea un hilo que se va a ejecutar un procedimiento llamado ThreadProc al mismo tiempo que el hilo principal. Después de la exitosa convocatoria, CreateThread regresa inmediatamente y ThreadProc comienza a correr. Debido a que no use la manija de hilo, hay que cerrarla bien habrá alguna pérdida de la memoria. Tenga en cuenta que el cierre de la manija de hilo no termina el hilo. Su único efecto es que no podemos utilizar el controlador de hilo de más.

```
ThreadProc PROC USOS Param ecx: DWORD
mov ecx, 600000000
Loop1:
add eax, eax
diciembre ecx
jz Get_out
jmp Loop1
Get_out:
invocar PostMessage, hwnd, WM_FINISH, NULL, NULL
ret
ThreadProc ENDP
```

Como puede ver, ThreadProc realiza un cálculo salvaje que lleva bastante tiempo para terminar y cuando finishs sus mensajes un mensaje WM_FINISH a la ventana principal. WM_FINISH es nuestro mensaje personalizado definido así:

```
WM_FINISH equ WM_USER 100 h
```

Usted no tiene que agregar WM_USER con 100h pero es más seguro para hacerlo.
El mensaje WM_FINISH sólo tiene sentido dentro de nuestro programa. Cuando aparezca la ventana principal recibe el mensaje WM_FINISH, se respons por mostrar un cuadro de mensaje que indica que el cálculo se ha completado.
Puede crear varios hilos en la sucesión mediante la opción "Crear Tema" varias veces.
En este ejemplo, la comunicación es un paso en que sólo el hilo puede notificar a la ventana principal. Si desea que el hilo principal para enviar comandos al subproceso de trabajo, puede que la siguiente manera:

- agregar un elemento de menú diciendo algo como "Kill Thread" en el menú
- una variable global que se usa como un indicador de comandos. TRUE = Detener el hilo, = FALSO continuar el hilo
- Modificar ThreadProc para comprobar el valor de la bandera de comandos en el circuito.

Cuando el usuario selecciona "Kill Thread" del menú, el programa principal se establece el valor TRUE en la bandera de comandos. Cuando ThreadProc ve que el valor del indicador de comando es TRUE, sale del bucle y vuelve Así termina el hilo.

TUTORIAL 16: OBJETO DE EVENTOS

Vamos a aprender lo que es un objeto de evento es y cómo usarlo en un programa multi-hilo.

TEORÍA:

Desde el tutorial anterior, que demostró cómo las discusiones comunicarse con un mensaje de ventana personalizada. Me fui a otros dos métodos: variable global y objeto de evento. Vamos a utilizar dos de ellas en este tutorial.

Un objeto de evento es como un interruptor: tiene sólo dos estados: encendido o apagado. Cuando un objeto de evento se enciende, es en la "señal" del Estado. Cuando está apagado, está en el "nonsignalled" del Estado. Para crear un objeto de evento y poner en un fragmento de código en los temas relevantes para observar el estado del objeto de evento. Si el objeto de evento es en el estado nonsignalled, los hilos que esperar a que se asleep.When los hilos están en estado de espera, que consumen tiempo de CPU poco.

Para crear un objeto de evento llamando a la función CreateEvent que tiene la siguiente sintaxis:

**CreateEvent lpEventAttributes proto: DWORD, **
**bManualReset: DWORD, **
**bInitialState: DWORD, **
lpName: DWORD

lpEventAttribute -> Si se especifica el valor NULL, el objeto evento se crea con el descriptor de seguridad predeterminado.

bManualReset -> Si desea que Windows para reiniciar automáticamente el objeto de evento para nonsignalled Estado después de la llamada WaitForSingleObject, debe especificar FALSO ya que este parámetro. Cosa que usted debe restablecer manualmente el objeto de evento con la llamada a ResetEvent.

bInitialState -> Si desea que el objeto de evento que se creará en el estado señalado, especifique este parámetro como TRUE de lo contrario el objeto de evento se creará en el estado nonsignalled.

lpName -> Puntero a una cadena ASCIIZ que es el nombre del objeto de evento. Este nombre se utiliza cuando se desea llamar OpenEvent.

Si la llamada tiene éxito, devuelve el identificador para el objeto de evento de nueva creación si no devuelve NULL.

Puede modificar el estado de un objeto de evento con dos llamadas a la API: SetEvent y ResetEvent. Función SetEvent establece el objeto evento en estado señalado. ResetEvent hace lo contrario.

Cuando el objeto evento se crea, debe colocar la llamada a WaitForSingleObject en el hilo que quiere ver el estado del objeto de evento. WaitForSingleObject tiene la siguiente sintaxis:

WaitForSingleObject proto hObject: DWORD, dwTimeout: DWORD

hObject -> Un asa a uno de los objetos de sincronización. Objeto de evento es un tipo de objeto de sincronización.

dwTimeout -> especifica el tiempo en milisegundos que esta función va a esperar a que el objeto se encuentre en estado señalado. Si el tiempo especificado ha pasado y el objeto de evento se encuentra todavía en estado de nonsignalled, WaitForSingleObject devuelve la persona que llama. Si desea esperar a que el objeto de forma indefinida, se debe especificar el valor infinito ya que este parámetro.

Ejemplo:

El siguiente ejemplo muestra una ventana de espera para que el usuario seleccione un comando desde el menú. Si el usuario selecciona "hilo de ejecutar", el hilo se inicia el cálculo salvaje. Cuando haya terminado, aparecerá un mensaje informando al usuario que el trabajo está hecho. Durante el tiempo que el hilo está en funcionamiento, el usuario puede seleccionar "hilo de parada" para detener el hilo.

0.386
. Modelo plano, stdcall
casemap opción: ninguno

```
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

Const.
IDM_START_THREAD un equ
IDM_STOP_THREAD equ 2
IDM_EXIT equ 3
WM_FINISH equ WM_USER 100 h

. Datos
ClassName db "Win32ASMEventClass", 0
AppName db "Win32 ASM Ejemplo de eventos", 0
MenuName db "FirstMenu", 0
SuccessString db "El cálculo se ha completado!", 0
StopString db "El hilo se detiene", 0
EventStop bool false

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwnd MANGO?
MANGO hMenu?
ThreadID DWORD?
ExitCode DWORD?
hEventStart MANGO?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
```

```
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, \
ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
invocar GetMenu, hwnd
mov hMenu, eax
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Sl uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
. ELSEIF uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_ENABLED
. Elseif ax == IDM_STOP_THREAD
mov EventStop, TRUE
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. ELSEIF uMsg == WM_FINISH
invocar el cuadro de mensaje, NULL, SuccessString ADDR, ADDR AppName, MB_OK
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
. Ecx tiempo! = 0
```

```
. Si EventStop! = TRUE
add eax, eax
diciembre ecx
. Más
invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
jmp ThreadProc
. Endif
. ENDW
invocar PostMessage, hwnd, WM_FINISH, NULL, NULL
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
jmp ThreadProc
ret
ThreadProc ENDP
poner fin a empezar a
```

Análisis:

En este ejemplo, demuestro otra técnica de hilo.

```
. SI uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
```

Se puede ver que puedo crear el objeto de evento y el hilo durante el procesamiento del mensaje WM_CREATE. Creo el objeto de evento en el estado nonsignalled con rearme automático. Después de que el objeto de evento se ha creado, creo el hilo. Sin embargo, el hilo no se ejecuta de inmediato, ya que espera a que el objeto de evento para estar en el estado señalado como el código de abajo:

```
ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
```

La primera línea del procedimiento de subproceso es la llamada a WaitForSingleObject. Se espera infinitamente por el estado señalado del objeto de evento antes de que vuelva. Esto significa que incluso cuando el hilo se crea, la ponemos en un estado latente.
Cuando el usuario selecciona "hilo de ejecutar" comando en el menú, se establece el objeto de evento en estado señalado a continuación:

```
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
```

La llamada a SetEvent convierte el objeto de evento en el estado señaló que a su vez hace que la llamada a WaitForSingleObject en el procedimiento de retorno hilo y el hilo conductor comienza. Cuando el usuario selecciona "hilo stop", que establece el valor de la variable global "EventStop" en TRUE.

```
. Si EventStop == FALSO
add eax, eax
diciembre ecx
. Más
```

invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
jmp ThreadProc
. Endif

Esto evita que el hilo y salta a la llamada a WaitForSingleObject de nuevo. Tenga en cuenta que no tenemos que reiniciar manualmente el objeto de evento en el estado nonsignalled porque especifica el parámetro de la llamada bManualReset CreateEvent como FALSE.

TUTORIAL 16: OBJETO DE EVENTOS

Vamos a aprender lo que es un objeto de evento es y cómo usarlo en un programa multi-hilo.

TEORÍA:

Desde el tutorial anterior, que demostró cómo las discusiones comunicarse con un mensaje de ventana personalizada. Me fui a otros dos métodos: variable global y objeto de evento. Vamos a utilizar dos de ellas en este tutorial.
Un objeto de evento es como un interruptor: tiene sólo dos estados: encendido o apagado. Cuando un objeto de evento se enciende, es en la "señal" del Estado. Cuando está apagado, está en el "nonsignalled" del Estado. Para crear un objeto de evento y poner en un fragmento de código en los temas relevantes para observar el estado del objeto de evento. Si el objeto de evento es en el estado nonsignalled, los hilos que esperar a que se asleep.When los hilos están en estado de espera, que consumen tiempo de CPU poco.
Para crear un objeto de evento llamando a la función CreateEvent que tiene la siguiente sintaxis:

**CreateEvent lpEventAttributes proto: DWORD, **
**bManualReset: DWORD, **
**bInitialState: DWORD, **
lpName: DWORD

lpEventAttribute -> Si se especifica el valor NULL, el objeto evento se crea con el descriptor de seguridad predeterminado.
bManualReset -> Si desea que Windows para reiniciar automáticamente el objeto de evento para nonsignalled Estado después de la llamada WaitForSingleObject, debe especificar FALSO ya que este parámetro. Cosa que usted debe restablecer manualmente el objeto de evento con la llamada a ResetEvent.
bInitialState -> Si desea que el objeto de evento que se creará en el estado señalado, especifique este parámetro como TRUE de lo contrario el objeto de evento se creará en el estado nonsignalled.
lpName -> Puntero a una cadena ASCIIZ que es el nombre del objeto de evento. Este nombre se utiliza cuando se desea llamar OpenEvent.

Si la llamada tiene éxito, devuelve el identificador para el objeto de evento de nueva creación si no devuelve NULL.
Puede modificar el estado de un objeto de evento con dos llamadas a la API: SetEvent y ResetEvent. Función SetEvent establece el objeto evento en estado señalado. ResetEvent hace lo contrario.
Cuando el objeto evento se crea, debe colocar la llamada a WaitForSingleObject en el hilo que quiere ver el estado del objeto de evento. WaitForSingleObject tiene la siguiente sintaxis:

WaitForSingleObject proto hObject: DWORD, dwTimeout: DWORD

hObject -> Un asa a uno de los objetos de sincronización. Objeto de evento es un tipo de objeto de sincronización.
dwTimeout -> especifica el tiempo en milisegundos que esta función va a esperar a que el objeto se encuentre en estado señalado. Si el tiempo especificado ha pasado y el objeto de evento se encuentra todavía en estado de nonsignalled, WaitForSingleObject devuelve la persona que llama. Si desea esperar a que el objeto de forma indefinida, se debe especificar el valor infinito ya que este parámetro.

EJEMPLO:

El siguiente ejemplo muestra una ventana de espera para que el usuario seleccione un comando desde el menú. Si el usuario selecciona "hilo de ejecutar", el hilo se inicia el cálculo salvaje. Cuando haya terminado, aparecerá un mensaje informando al usuario que el trabajo está hecho. Durante el tiempo que el hilo está en funcionamiento, el usuario puede seleccionar "hilo de parada" para detener el hilo.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

Const.
IDM_START_THREAD un equ
IDM_STOP_THREAD equ 2
IDM_EXIT equ 3
WM_FINISH equ WM_USER 100 h

. Datos
ClassName db "Win32ASMEventClass", 0
AppName db "Win32 ASM Ejemplo de eventos", 0
MenuName db "FirstMenu", 0
SuccessString db "El cálculo se ha completado!", 0
StopString db "El hilo se detiene", 0
EventStop bool false

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwnd MANGO?
MANGO hMenu?
ThreadID DWORD?
ExitCode DWORD?
hEventStart MANGO?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax
```

```
WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, \
ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, 300.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
invocar GetMenu, hwnd
mov hMenu, eax
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. SI uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
. ELSEIF uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_ENABLED
. Elseif ax == IDM_STOP_THREAD
mov EventStop, TRUE
```

```
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. ELSEIF uMsg == WM_FINISH
invocar el cuadro de mensaje, NULL, SuccessString ADDR, ADDR AppName, MB_OK
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
. Ecx tiempo! = 0
. Si EventStop! = TRUE
add eax, eax
diciembre ecx
. Más
invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
jmp ThreadProc
. Endif
. ENDW
invocar PostMessage, hwnd, WM_FINISH, NULL, NULL
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
jmp ThreadProc
ret
ThreadProc ENDP
poner fin a empezar a
```

Análisis:

En este ejemplo, demuestro otra técnica de hilo.

```
. SI uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
```

Se puede ver que puedo crear el objeto de evento y el hilo durante el procesamiento del mensaje WM_CREATE. Creo el objeto de evento en el estado nonsignalled con rearme automático. Después de que el objeto de evento se ha creado, creo el hilo. Sin embargo, el hilo no se ejecuta de inmediato, ya que espera a que el objeto de evento para estar en el estado señalado como el código de abajo:

```
ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
```

La primera línea del procedimiento de subproceso es la llamada a WaitForSingleObject. Se espera infinitamente por el estado señalado del objeto de evento antes de que vuelva. Esto significa que incluso cuando el hilo se crea, la ponemos en un estado latente.
Cuando el usuario selecciona "hilo de ejecutar" comando en el menú, se establece el objeto de evento en estado señalado a continuación:

```
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
```

La llamada a SetEvent convierte el objeto de evento en el estado señaló que a su vez hace que la llamada a WaitForSingleObject en el procedimiento de retorno hilo y el hilo conductor comienza. Cuando el usuario selecciona "hilo stop", que establece el valor de la variable global "EventStop" en TRUE.

```
. Si EventStop == FALSO
add eax, eax
diciembre ecx
. Más
invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
jmp ThreadProc
. Endif
```

Esto evita que el hilo y salta a la llamada a WaitForSingleObject de nuevo. Tenga en cuenta que no tenemos que reiniciar manualmente el objeto de evento en el estado nonsignalled porque especifica el parámetro de la llamada bManualReset CreateEvent como FALSE.

TUTORIAL 17: LAS BIBLIOTECAS DE VÍNCULOS DINÁMICOS

En este tutorial, vamos a aprender acerca de los archivos DLL, ¿cuáles son y cómo crearlos.
.

TEORÍA:

Si se programa el tiempo suficiente, usted encontrará que los programas que ha escrito por lo general tienen algunas rutinas de código en común. Es una pérdida de tiempo volver a escribir cada vez que empezar a programar nuevos programas. De vuelta en los viejos tiempos de DOS, los programadores de almacenar las rutinas de uso común en una o más bibliotecas. Cuando se desea utilizar las funciones, que ponga el enlace de la biblioteca del archivo del objeto y el enlazador extrae de las funciones de la biblioteca y los inserta en el archivo ejecutable final. Este proceso se denomina vinculación estática. Las bibliotecas de C en tiempo de ejecución son buenos ejemplos. El inconveniente de este método es que tiene funciones idénticas en todos los programas que los llama. Su espacio en el disco se pierde el almacenamiento de varias copias idénticas de las funciones. Sin embargo, para los programas de DOS, este método es bastante aceptable, ya que normalmente sólo hay un programa que está activo en la memoria. Así que no hay pérdida de memoria preciosa.
En Windows, la situación se vuelve mucho más crítica, ya que puede tener varios programas que se ejecutan simultáneamente. La memoria se comen rápidamente si el programa es bastante grande. Windows tiene una solución para este tipo de problema: las bibliotecas de vínculos dinámicos. Una librería de enlace dinámico es una especie de fondo común de funciones. Windows no se carga varias copias de un archivo DLL en la memoria por lo que aunque hay muchos casos de su programa que se ejecuta al mismo tiempo, habrá sólo una copia de la DLL que el programa utiliza en la memoria. Y debo aclarar este punto un poco. En realidad, todos los procesos que utilizan la misma DLL tendrá sus propias copias de esa DLL.

Se verá como hay muchas copias de la DLL en la memoria. Pero, en realidad, Windows lo hace magia con la paginación, y todos comparten los procesos de la code.So misma DLL en la memoria física, sólo hay una copia del código de la DLL. Sin embargo, cada proceso tiene su propia sección de datos única de la DLL.

El programa vincula a un archivo DLL en tiempo de ejecución a diferencia de la antigua biblioteca estática. Es por eso que se llama biblioteca de vínculos dinámicos. También puede descargar un archivo DLL en tiempo de ejecución, así cuando usted no lo necesita. Si ese programa es el único que utiliza la DLL, va a ser descargado de la memoria inmediata. Sin embargo, si el archivo DLL se utiliza todavía por algún otro programa, el archivo DLL permanece en memoria hasta el último programa que utiliza sus servicios de descarga es.

Sin embargo, el vinculador tiene un trabajo más difícil cuando se realiza composturas de dirección para el archivo ejecutable final. Dado que no se puede "extraer" las funciones y los inserta en el archivo ejecutable final, de alguna manera se debe almacenar suficiente información sobre el archivo DLL y funciones en el archivo execuable final para que sea capaz de localizar y cargar el archivo DLL en tiempo de ejecución correcta.

Ahí es donde entra en juego la biblioteca de importación una biblioteca de importación contiene la información sobre el archivo DLL que representa. El enlazador puede extraer la información que necesita de las bibliotecas de importación y meterlo en el archivo ejecutable. Cuando cargador de Windows carga el programa en la memoria, se ve que los enlaces del programa a un archivo DLL por lo que busca ese archivo DLL y se asigna al espacio de direcciones del proceso, así y lleva a cabo las composturas de direcciones para las llamadas a las funciones de la DLL .

Usted puede optar por cargar la DLL de sí mismo sin depender de cargador de Windows. Este método tiene sus pros y sus contras:

- No necesita una biblioteca de importación para que pueda cargar y utilizar cualquier archivo DLL, aún si no viene con ninguna biblioteca de importación. Sin embargo, usted todavía tiene que saber acerca de las funciones dentro de ella, cuántos parámetros que toman y los gustos.
- Cuando dejas el cargador de cargar el archivo DLL para su programa, si el cargador no puede encontrar el archivo DLL que se informe "Un archivo requerido. DLL, xxxxx.dll falta", y ¡puf! su programa no tiene la oportunidad de correr, incluso si esa DLL no es indispensable para su funcionamiento. Si carga el archivo DLL de sí mismo, cuando el archivo DLL no se puede encontrar y no es esencial para el funcionamiento, el programa sólo puede avisar al usuario sobre el hecho y seguir adelante.
- Usted puede llamar al * indocumentados * funciones que no están incluidos en las bibliotecas de importación. Siempre que usted sabe suficiente información acerca de las funciones.
- Si utiliza LoadLibrary, usted tiene que llamar a GetProcAddress para cada función que se desea llamar. GetProcAddress recupera la dirección del punto de entrada de una función en una DLL en particular. Así que el código podría ser un poco más grande y más lento, pero no mucho.

Al ver las ventajas y desventajas de la llamada a LoadLibrary, de entrar en detalles de cómo crear un archivo DLL ahora.

El siguiente código es el esqueleto de DLL.

```
; ------------------------------------------------ ------------------------------------
; DLLSkeleton.asm
; ------------------------------------------------ ------------------------------------
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
```

```
. Datos
. Código
DllEntry proc hinstDLL: HINSTANCE, la razón: DWORD, reserved1: DWORD
mov eax, TRUE
ret
DllEntry Endp
; ------------------------------------------------ ----------------------------------------------------
; Esta es una función ficticia
; No hace nada.  Lo puse aquí para ver donde se puede insertar funciones en
; Un archivo DLL.

TestFunction proc
ret
TestFunction endp

Fin DllEntry

; ------------------------------------------------ ----------------------------------
; DLLSkeleton.def
; ------------------------------------------------ ----------------------------------
BIBLIOTECA DLLSkeleton
EXPORTACIONES TestFunction
```

El programa anterior es el esqueleto de DLL. Cada DLL debe tener una función de punto de entrada. Windows cada vez que llame a la función de punto de entrada que:

- El archivo DLL se carga por primera vez
- El archivo DLL se descarga
- Un hilo se crea en el mismo proceso
- Un hilo es destruido en el mismo proceso

```
DllEntry proc hinstDLL: HINSTANCE, la razón: DWORD, reserved1: DWORD
mov eax, TRUE
ret
DllEntry Endp
```

Puede asignar el nombre de la función de punto de entrada que desee, siempre y cuando usted tiene un FIN coincidencia <nombre de la función <Entrypoint. Esta función toma tres parámetros, sólo el primero de los cuales dos son importantes.
hinstDLL es el identificador de módulo de la DLL. No es el mismo que el identificador de instancia del proceso. Usted debe mantener este valor si necesita usarlo más tarde. No se puede obtener de nuevo con facilidad.
razón puede ser uno de los cuatro valores:

- **DLL_PROCESS_ATTACH** La DLL recibe este valor cuando se inyecta por primera vez en el espacio de direcciones del proceso. Usted puede aprovechar esta oportunidad para hacer la inicialización.
- **DLL_PROCESS_DETACH** La DLL recibe este valor cuando se está descargando desde el espacio de direcciones del proceso. Usted puede utilizar esta oportunidad para hacer algo de limpieza, como la memoria deallocate y así sucesivamente.
- **DLL_THREAD_ATTACH** La DLL recibe este valor cuando el proceso se crea un nuevo hilo.
- **DLL_THREAD_DETACH** La DLL recibe este valor cuando un hilo en el proceso se destruye.

Se devuelve TRUE en eax si desea que el archivo DLL que ir corriendo. Si usted devuelve FALSO, la DLL no se cargará. Por ejemplo, si el código de inicialización debe destinar parte de la memoria y no puede hacerlo con éxito, la función de punto de entrada debe devolver FALSE

para indicar que la DLL no se puede ejecutar.
Usted puede poner sus funciones en la DLL después de la función de punto de entrada o antes de ella. Pero si usted quisiera que fueran se puede llamar desde otros programas, debe poner sus nombres en la lista de exportación en el archivo de definición de módulo (. Def).
Un archivo DLL necesita un archivo de definición de módulo en su etapa de desarrollo. Vamos a echar un vistazo ahora.

BIBLIOTECA DLLSkeleton
EXPORTACIONES TestFunction

Normalmente, usted debe tener la sentencia line.The **BIBLIOTECA** primero define el nombre interno del módulo de la DLL. Usted debe coincidir con el nombre de la DLL.
La declaración **EXPORTACIONES** le indica al vinculador que funciona en el archivo DLL se exportan, es decir, se puede llamar desde otros programas. En el ejemplo, queremos que los otros módulos para poder llamar a TestFunction, por lo que poner su nombre en la declaración **EXPORTACIONES.**
Otro cambio está en el interruptor de enlazador. Usted debe poner **/ DLL** interruptor y **/ DEF: nombre de archivo <su definición** de los interruptores del vinculador como esta:

enlace / DLL / SUBSYSTEM: WINDOWS / DEF: DLLSkeleton.def / LIBPATH: c: \ masm32 \ lib DLLSkeleton.obj

Los interruptores de ensamblador son los mismos, a saber / c / coff / Cp. Así que después de enlazar el archivo objeto, va a conseguir. Dll y lib.. El lib. Es la biblioteca de importación que se puede utilizar para acceder a otros programas que utilizan las funciones de la DLL.
A continuación te mostraré cómo usar LoadLibrary para cargar un archivo DLL.

```
; ---------------------------------------------- ----------------------------------------
; UseDLL.asm
; ---------------------------------------------- ----------------------------------------
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ user32.lib

. Datos
LibName db "DLLSkeleton.dll", 0
FunctionName db "TestHello", 0
DllNotFound db "No se puede cargar la biblioteca", 0
AppName db "cargar la biblioteca", 0
FunctionNotFound db "función TestHello no encontrado", 0

. Datos?
hLib dd? ; El mango de la biblioteca (DLL)
TestHelloAddr dd? ; La dirección de la función TestHello

. Código
empezar:
invocar LoadLibrary, addr libName

, Llamar a LoadLibrary con el nombre de la DLL que desee. Si la llamada es exitosa
, Sino que se devuelva la palanca a la biblioteca (DLL). Si no, se devuelven NULL
; Puede pasar el identificador de la biblioteca a GetProcAddress o cualquier función que requiera
```

```
; Una biblioteca manejar como un parámetro.

. Si eax == NULL
invocar el cuadro de mensaje, NULL, DllNotFound addr, addr AppName, MB_OK
. Más
mov hLib, eax
invocar GetProcAddress, hLib, addr FunctionName

, Cuando llegue el mango biblioteca, se le pasa a GetProcAddress con la dirección
, Del nombre de la función en esa DLL al que desea llamar.  Devuelve la dirección
; De la función en caso de éxito.  De lo contrario, devuelve NULL
; Las direcciones de las funciones no cambian a menos que descargar y cargar la
biblioteca.
, Así que usted puede poner en las variables globales para su uso futuro.

. Si eax == NULL
invocar el cuadro de mensaje, NULL, FunctionNotFound addr, addr AppName, MB_OK
. Más
mov TestHelloAddr, eax
llamar al [TestHelloAddr]

; A continuación, puede llamar a la función con una simple llamada con la variable que
contiene
; La dirección de la función como el operando.

. Endif
invocar FreeLibrary, hLib

Cuando ya no necesita la biblioteca más, se descarga con FreeLibrary.

. Endif
invocar ExitProcess, NULL
poner fin a empezar a
```

Así se puede ver que el uso de LoadLibrary es un poco más complicado pero también es más flexible

TUTORIAL 16: OBJETO DE EVENTOS

Vamos a aprender lo que es un objeto de evento es y cómo usarlo en un programa multi-hilo.

TEORÍA:

Desde el tutorial anterior, que demostró cómo las discusiones comunicarse con un mensaje de ventana personalizada. Me fui a otros dos métodos: variable global y objeto de evento. Vamos a utilizar dos de ellas en este tutorial.

Un objeto de evento es como un interruptor: tiene sólo dos estados: encendido o apagado. Cuando un objeto de evento se enciende, es en la "señal" del Estado. Cuando está apagado, está en el "nonsignalled" del Estado. Para crear un objeto de evento y poner en un fragmento de código en los temas relevantes para observar el estado del objeto de evento. Si el objeto de evento es en el estado nonsignalled, los hilos que esperar a que se asleep.When los hilos están en estado de espera, que consumen tiempo de CPU poco.
Para crear un objeto de evento llamando a la función CreateEvent que tiene la siguiente sintaxis:

**CreateEvent lpEventAttributes proto: DWORD, **
**bManualReset: DWORD, **
**bInitialState: DWORD, **
lpName: DWORD

lpEventAttribute -> Si se especifica el valor NULL, el objeto evento se crea con el descriptor de seguridad predeterminado.
bManualReset -> Si desea que Windows para reiniciar automáticamente el objeto de evento para nonsignalled Estado después de la llamada WaitForSingleObject, debe especificar FALSO ya que este parámetro. Cosa que usted debe restablecer manualmente el objeto de evento con la llamada a ResetEvent.
bInitialState -> Si desea que el objeto de evento que se creará en el estado señalado, especifique este parámetro como TRUE de lo contrario el objeto de evento se creará en el estado nonsignalled.
lpName -> Puntero a una cadena ASCIIZ que es el nombre del objeto de evento. Este nombre se utiliza cuando se desea llamar OpenEvent.

Si la llamada tiene éxito, devuelve el identificador para el objeto de evento de nueva creación si no devuelve NULL.
Puede modificar el estado de un objeto de evento con dos llamadas a la API: SetEvent y ResetEvent. Función SetEvent establece el objeto evento en estado señalado. ResetEvent hace lo contrario.
Cuando el objeto evento se crea, debe colocar la llamada a WaitForSingleObject en el hilo que quiere ver el estado del objeto de evento. WaitForSingleObject tiene la siguiente sintaxis:

WaitForSingleObject proto hObject: DWORD, dwTimeout: DWORD

hObject -> Un asa a uno de los objetos de sincronización. Objeto de evento es un tipo de objeto de sincronización.
dwTimeout -> especifica el tiempo en milisegundos que esta función va a esperar a que el objeto se encuentre en estado señalado. Si el tiempo especificado ha pasado y el objeto de evento se encuentra todavía en estado de nonsignalled, WaitForSingleObject devuelve la persona que llama. Si desea esperar a que el objeto de forma indefinida, se debe especificar el valor infinito ya que este parámetro.

Ejemplo:

El siguiente ejemplo muestra una ventana de espera para que el usuario seleccione un comando desde el menú. Si el usuario selecciona "hilo de ejecutar", el hilo se inicia el cálculo salvaje. Cuando haya terminado, aparecerá un mensaje informando al usuario que el trabajo está hecho. Durante el tiempo que el hilo está en funcionamiento, el usuario puede seleccionar "hilo de parada" para detener el hilo.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
```

```
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

Const.
IDM_START_THREAD un equ
IDM_STOP_THREAD equ 2
IDM_EXIT equ 3
WM_FINISH equ WM_USER 100 h

. Datos
ClassName db "Win32ASMEventClass", 0
AppName db "Win32 ASM Ejemplo de eventos", 0
MenuName db "FirstMenu", 0
SuccessString db "El cálculo se ha completado!", 0
StopString db "El hilo se detiene", 0
EventStop bool false

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hwnd MANGO?
MANGO hMenu?
ThreadID DWORD?
ExitCode DWORD?
hEventStart MANGO?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, MenuName OFFSET
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, \
ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
```

```
CW_USEDEFAULT, 300.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
invocar GetMenu, hwnd
mov hMenu, eax
. MIENTRAS VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. ENDW
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. SI uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
. ELSEIF uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. ELSEIF uMsg == WM_COMMAND
mov eax, wParam
. Si lParam == 0
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_GRAYED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_ENABLED
. Elseif ax == IDM_STOP_THREAD
mov EventStop, TRUE
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. ELSEIF uMsg == WM_FINISH
invocar el cuadro de mensaje, NULL, SuccessString ADDR, ADDR AppName, MB_OK
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP

ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
. Ecx tiempo! = 0
. Si EventStop! = TRUE
add eax, eax
diciembre ecx
```

```
. Más
invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
jmp ThreadProc
. Endif
. ENDW
invocar PostMessage, hwnd, WM_FINISH, NULL, NULL
invocar EnableMenuItem, hMenu, IDM_START_THREAD, MF_ENABLED
invocar EnableMenuItem, hMenu, IDM_STOP_THREAD, MF_GRAYED
jmp ThreadProc
ret
ThreadProc ENDP
poner fin a empezar a
```

Análisis:

En este ejemplo, demuestro otra técnica de hilo.

```
. Si uMsg == WM_CREATE
invocar CreateEvent, NULL, FALSE, NULL FALSO,
mov hEventStart, eax
mov eax, OFFSET ThreadProc
invocar CreateThread, NULL, NULL, eax, \
NULL, 0, \
ADDR ThreadID
invocar CloseHandle, eax
```

Se puede ver que puedo crear el objeto de evento y el hilo durante el procesamiento del mensaje WM_CREATE. Creo el objeto de evento en el estado nonsignalled con rearme automático. Después de que el objeto de evento se ha creado, creo el hilo. Sin embargo, el hilo no se ejecuta de inmediato, ya que espera a que el objeto de evento para estar en el estado señalado como el código de abajo:

```
ThreadProc PROC USOS Param ecx: DWORD
invocar WaitForSingleObject, hEventStart, INFINITO
mov ecx, 600000000
```

La primera línea del procedimiento de subproceso es la llamada a WaitForSingleObject. Se espera infinitamente por el estado señalado del objeto de evento antes de que vuelva. Esto significa que incluso cuando el hilo se crea, la ponemos en un estado latente.
Cuando el usuario selecciona "hilo de ejecutar" comando en el menú, se establece el objeto de evento en estado señalado a continuación:

```
. Si ax == IDM_START_THREAD
invocar SetEvent, hEventStart
```

La llamada a SetEvent convierte el objeto de evento en el estado señaló que a su vez hace que la llamada a WaitForSingleObject en el procedimiento de retorno hilo y el hilo conductor comienza. Cuando el usuario selecciona "hilo stop", que establece el valor de la variable global "EventStop" en TRUE.

```
. Si EventStop == FALSO
add eax, eax
diciembre ecx
. Más
invocar el cuadro de mensajes, hwnd, StopString ADDR, ADDR AppName, MB_OK
mov EventStop, FALSO
```

jmp ThreadProc
. Endif

Esto evita que el hilo y salta a la llamada a WaitForSingleObject de nuevo. Tenga en cuenta que no tenemos que reiniciar manualmente el objeto de evento en el estado nonsignalled porque especifica el parámetro de la llamada bManualReset CreateEvent como FALSE.

TUTORIAL 17: LAS BIBLIOTECAS DE VÍNCULOS DINÁMICOS

En este tutorial, vamos a aprender acerca de los archivos DLL, ¿cuáles son y cómo crearlos.
.

TEORÍA:

Si se programa el tiempo suficiente, usted encontrará que los programas que ha escrito por lo general tienen algunas rutinas de código en común. Es una pérdida de tiempo volver a escribir cada vez que empezar a programar nuevos programas. De vuelta en los viejos tiempos de DOS, los programadores de almacenar las rutinas de uso común en una o más bibliotecas. Cuando se desea utilizar las funciones, que ponga el enlace de la biblioteca del archivo del objeto y el enlazador extrae de las funciones de la biblioteca y los inserta en el archivo ejecutable final. Este proceso se denomina vinculación estática. Las bibliotecas de C en tiempo de ejecución son buenos ejemplos. El inconveniente de este método es que tiene funciones idénticas en todos los programas que los llama. Su espacio en el disco se pierde el almacenamiento de varias copias idénticas de las funciones. Sin embargo, para los programas de DOS, este método es bastante aceptable, ya que normalmente sólo hay un programa que está activo en la memoria. Así que no hay pérdida de memoria preciosa.
En Windows, la situación se vuelve mucho más crítica, ya que puede tener varios programas que se ejecutan simultáneamente. La memoria se comen rápidamente si el programa es bastante grande. Windows tiene una solución para este tipo de problema: las bibliotecas de vínculos dinámicos. Una librería de enlace dinámico es una especie de fondo común de funciones. Windows no se carga varias copias de un archivo DLL en la memoria por lo que aunque hay muchos casos de su programa que se ejecuta al mismo tiempo, habrá sólo una copia de la DLL que el programa utiliza en la memoria. Y debo aclarar este punto un poco. En realidad, todos los procesos que utilizan la misma DLL tendrá sus propias copias de esa DLL. Se verá como hay muchas copias de la DLL en la memoria. Pero, en realidad, Windows lo hace magia con la paginación, y todos comparten los procesos de la code.So misma DLL en la memoria física, sólo hay una copia del código de la DLL. Sin embargo, cada proceso tiene su propia sección de datos única de la DLL.
El programa vincula a un archivo DLL en tiempo de ejecución a diferencia de la antigua biblioteca estática. Es por eso que se llama biblioteca de vínculos dinámicos. También puede descargar un archivo DLL en tiempo de ejecución, así cuando usted no lo necesita. Si ese programa es el único que utiliza la DLL, va a ser descargado de la memoria inmediata. Sin embargo, si el archivo DLL se utiliza todavía por algún otro programa, el archivo DLL permanece en memoria hasta el último programa que utiliza sus servicios de descarga es.
Sin embargo, el vinculador tiene un trabajo más difícil cuando se realiza composturas de dirección para el archivo ejecutable final. Dado que no se puede "extraer" las funciones y los inserta en el archivo ejecutable final, de alguna manera se debe almacenar suficiente información sobre el archivo DLL y funciones en el archivo execuable final para que sea capaz de localizar y cargar el archivo DLL en tiempo de ejecución correcta.
Ahí es donde entra en juego la biblioteca de importación una biblioteca de importación contiene la información sobre el archivo DLL que representa. El enlazador puede extraer la información que necesita de las bibliotecas de importación y meterlo en el archivo ejecutable. Cuando cargador de Windows carga el programa en la memoria, se ve que los enlaces del programa a un archivo DLL por lo que busca ese archivo DLL y se asigna al espacio de

direcciones del proceso, así y lleva a cabo las composturas de direcciones para las llamadas a las funciones de la DLL .
Usted puede optar por cargar la DLL de sí mismo sin depender de cargador de Windows. Este método tiene sus pros y sus contras:

- No necesita una biblioteca de importación para que pueda cargar y utilizar cualquier archivo DLL, aún si no viene con ninguna biblioteca de importación. Sin embargo, usted todavía tiene que saber acerca de las funciones dentro de ella, cuántos parámetros que toman y los gustos.
- Cuando dejas el cargador de cargar el archivo DLL para su programa, si el cargador no puede encontrar el archivo DLL que se informe "Un archivo requerido. DLL, xxxxx.dll falta", y ¡puf! su programa no tiene la oportunidad de correr, incluso si esa DLL no es indispensable para su funcionamiento. Si carga el archivo DLL de sí mismo, cuando el archivo DLL no se puede encontrar y no es esencial para el funcionamiento, el programa sólo puede avisar al usuario sobre el hecho y seguir adelante.
- Usted puede llamar al * indocumentados * funciones que no están incluidos en las bibliotecas de importación. Siempre que usted sabe suficiente información acerca de las funciones.
- Si utiliza LoadLibrary, usted tiene que llamar a GetProcAddress para cada función que se desea llamar. GetProcAddress recupera la dirección del punto de entrada de una función en una DLL en particular. Así que el código podría ser un poco más grande y más lento, pero no mucho.

Al ver las ventajas y desventajas de la llamada a LoadLibrary, de entrar en detalles de cómo crear un archivo DLL ahora.
El siguiente código es el esqueleto de DLL.

```
; ------------------------------------------------ ------------------------------------
; DLLSkeleton.asm
; ------------------------------------------------ ------------------------------------
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

. Datos
. Código
DllEntry proc hinstDLL: HINSTANCE, la razón: DWORD, reserved1: DWORD
mov eax, TRUE
ret
DllEntry Endp
; ------------------------------------------------ ------------------------------------------------
; Esta es una función ficticia
; No hace nada. Lo puse aquí para ver donde se puede insertar funciones en
; Un archivo DLL.

TestFunction proc
ret
TestFunction endp

Fin DllEntry

; ------------------------------------------------ ------------------------------------
; DLLSkeleton.def
; ------------------------------------------------ ------------------------------------
```

BIBLIOTECA DLLSkeleton
EXPORTACIONES TestFunction

El programa anterior es el esqueleto de DLL. Cada DLL debe tener una función de punto de entrada. Windows cada vez que llame a la función de punto de entrada que:

- El archivo DLL se carga por primera vez
- El archivo DLL se descarga
- Un hilo se crea en el mismo proceso
- Un hilo es destruido en el mismo proceso

DllEntry proc hinstDLL: HINSTANCE, la razón: DWORD, reserved1: DWORD
mov eax, TRUE
ret
DllEntry Endp

Puede asignar el nombre de la función de punto de entrada que desee, siempre y cuando usted tiene un FIN coincidencia <nombre de la función <Entrypoint. Esta función toma tres parámetros, sólo el primero de los cuales dos son importantes.
hinstDLL es el identificador de módulo de la DLL. No es el mismo que el identificador de instancia del proceso. Usted debe mantener este valor si necesita usarlo más tarde. No se puede obtener de nuevo con facilidad.
razón puede ser uno de los cuatro valores:

- **DLL_PROCESS_ATTACH** La DLL recibe este valor cuando se inyecta por primera vez en el espacio de direcciones del proceso. Usted puede aprovechar esta oportunidad para hacer la inicialización.
- **DLL_PROCESS_DETACH** La DLL recibe este valor cuando se está descargando desde el espacio de direcciones del proceso. Usted puede utilizar esta oportunidad para hacer algo de limpieza, como la memoria deallocate y así sucesivamente.
- **DLL_THREAD_ATTACH** La DLL recibe este valor cuando el proceso se crea un nuevo hilo.
- **DLL_THREAD_DETACH** La DLL recibe este valor cuando un hilo en el proceso se destruye.

Se devuelve TRUE en eax si desea que el archivo DLL que ir corriendo. Si usted devuelve FALSO, la DLL no se cargará. Por ejemplo, si el código de inicialización debe destinar parte de la memoria y no puede hacerlo con éxito, la función de punto de entrada debe devolver FALSE para indicar que la DLL no se puede ejecutar.
Usted puede poner sus funciones en la DLL después de la función de punto de entrada o antes de ella. Pero si usted quisiera que fueran se puede llamar desde otros programas, debe poner sus nombres en la lista de exportación en el archivo de definición de módulo (. Def).
Un archivo DLL necesita un archivo de definición de módulo en su etapa de desarrollo. Vamos a echar un vistazo ahora.

BIBLIOTECA DLLSkeleton
EXPORTACIONES TestFunction

Normalmente, usted debe tener la sentencia line.The **BIBLIOTECA** primero define el nombre interno del módulo de la DLL. Usted debe coincidir con el nombre de la DLL.
La declaración **EXPORTACIONES** le indica al vinculador que funciona en el archivo DLL se exportan, es decir, se puede llamar desde otros programas. En el ejemplo, queremos que los otros módulos para poder llamar a TestFunction, por lo que poner su nombre en la declaración **EXPORTACIONES.**
Otro cambio está en el interruptor de enlazador. Usted debe poner **/ DLL** interruptor y **/ DEF: nombre de archivo <su definición** de los interruptores del vinculador como esta:

enlace / DLL / SUBSYSTEM: WINDOWS / DEF: DLLSkeleton.def / LIBPATH: c: \ masm32 \ lib DLLSkeleton.obj

Los interruptores de ensamblador son los mismos, a saber / c / coff / Cp. Así que después de enlazar el archivo objeto, va a conseguir. Dll y lib.. El lib. Es la biblioteca de importación que se puede utilizar para acceder a otros programas que utilizan las funciones de la DLL.
A continuación te mostraré cómo usar LoadLibrary para cargar un archivo DLL.

```
; ---------------------------------------------- ----------------------------------------
; UseDLL.asm
; ---------------------------------------------- ----------------------------------------
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ user32.lib

. Datos
LibName db "DLLSkeleton.dll", 0
FunctionName db "TestHello", 0
DllNotFound db "No se puede cargar la biblioteca", 0
AppName db "cargar la biblioteca", 0
FunctionNotFound db "función TestHello no encontrado", 0

. Datos?
hLib dd? ; El mango de la biblioteca (DLL)
TestHelloAddr dd? ; La dirección de la función TestHello

. Código
empezar:
invocar LoadLibrary, addr libName

, Llamar a LoadLibrary con el nombre de la DLL que desee. Si la llamada es exitosa
, Sino que se devuelva la palanca a la biblioteca (DLL). Si no, se devuelven NULL
; Puede pasar el identificador de la biblioteca a GetProcAddress o cualquier función que requiera
; Una biblioteca manejar como un parámetro.

. Si eax == NULL
invocar el cuadro de mensaje, NULL, DllNotFound addr, addr AppName, MB_OK
. Más
mov hLib, eax
invocar GetProcAddress, hLib, addr FunctionName

, Cuando llegue el mango biblioteca, se le pasa a GetProcAddress con la dirección
, Del nombre de la función en esa DLL al que desea llamar. Devuelve la dirección
; De la función en caso de éxito. De lo contrario, devuelve NULL
; Las direcciones de las funciones no cambian a menos que descargar y cargar la biblioteca.
, Así que usted puede poner en las variables globales para su uso futuro.

. Si eax == NULL
invocar el cuadro de mensaje, NULL, FunctionNotFound addr, addr AppName, MB_OK
. Más
mov TestHelloAddr, eax
llamar al [TestHelloAddr]
```

; A continuación, puede llamar a la función con una simple llamada con la variable que contiene
; La dirección de la función como el operando.

. Endif
invocar FreeLibrary, hLib

Cuando ya no necesita la biblioteca más, se descarga con FreeLibrary.

. Endif
invocar ExitProcess, NULL
poner fin a empezar a

Así se puede ver que el uso de LoadLibrary es un poco más complicado pero también es más flexible.

TUTORIAL 19: CONTROL DE VISTA DE ÁRBOL

En este tutorial, vamos a aprender cómo utilizar el control de vista de árbol. Por otra parte, también vamos a aprender a hacer de arrastrar y soltar bajo el control de vista de árbol y cómo utilizar una lista de imágenes con ella.

TEORÍA:

Un control de vista de árbol es una especie de ventana especial que representa objetos en orden jerárquico. Un ejemplo de ello es el panel izquierdo del Explorador de Windows. Puede utilizar este control para mostrar las relaciones entre los objetos.
Usted puede crear un control de vista de árbol llamando CreateWindowEx, pasando por "SysTreeView32" como el nombre de la clase o se puede incorporar en un cuadro de diálogo. No se olvide de poner InitCommonControls llamada en el código.
Hay varios estilos específicos para el control de vista de árbol. Estos tres son los más utilizados.

> **TVS_HASBUTTONS == Muestra positivo (+) y negativo (-) botones al lado de los elementos principales. El usuario hace clic en los botones para expandir o contraer la lista de un elemento primario de elementos secundarios. Para incluir botones con los elementos en la raíz de la vista de árbol, TVS_LINESATROOT también se debe especificar.**
> **TVS_HASLINES == Usa líneas para mostrar la jerarquía de los elementos.**
> **TVS_LINESATROOT == Usa líneas para unir los elementos en la raíz del control tree-view. Este valor se ignora si TVS_HASLINES no se especifica también.**

El control de vista de árbol, al igual que otros controles comunes, se comunica con la ventana principal a través de mensajes. La ventana padre puede enviar varios mensajes a la misma y el control de vista de árbol puede enviar mensajes de "notificación" a su ventana padre. En este sentido, el control de vista de árbol no es diferente que cualquier ventana.
Cuando ocurre algo interesante para él, envía un mensaje **WM_NOTIFY** a la ventana de los padres con información que lo acompaña.

WM_NOTIFY
wParam == ID de control, este valor no se garantiza que sea único por lo que no lo use.
En cambio, usamos los miembros hwndFrom o idFrom de la estructura NMHDR a la que apunta lParam
lParam == Puntero a la estructura NMHDR. Algunos controles pueden pasar un puntero a una mayor
estructura, sino que debe tener una estructura NMHDR como su primer miembro.
Es decir, cuando usted tiene lParam, usted puede estar seguro de que apunta a una
NMHDR estructura por lo menos.

A continuación vamos a examinar NMHDR estructura.

NMHDR estructura DWORD
hwndFrom DWORD?
idFrom DWORD?
código de DWORD?
NMHDR termina

hwndFrom es el identificador de ventana del control que le envía este mensaje **WM_NOTIFY.**
idFrom es el identificador del control del control que le envía este mensaje WM_NOTIFY.
código es el mensaje real del control quiere enviar a la ventana padre.
Notificaciones de árbol vista son aquellos con TVN_ al comienzo del nombre. Mensajes de la vista de árbol son los que tienen TVM_, como **TVM_CREATEDRAGIMAGE.** El control de vista de árbol envía TVN_xxxx en el elemento de código de NMHDR. La ventana padre puede enviar TVM_xxxx para controlarlo.

Para añadir elementos a un control de vista de árbol
Después de crear un control de vista de árbol, puede agregar elementos a la misma. Usted puede hacer esto mediante el envío de **TVM_INSERTITEM** a ella.

TVM_INSERTITEM
wParam = 0;
lParam = puntero a una TV_INSERTSTRUCT;

Usted debe saber algo de terminología en este punto sobre la relación entre los elementos del control de vista de árbol.
Un elemento puede ser padre, hijo, o ambas cosas a la vez. Un elemento principal es el elemento que tiene algún otro subpunto (s) asociado con él. Al mismo tiempo, el elemento primario puede ser un niño de algún otro elemento. Un artículo sin un padre que se llama un elemento raíz. No puede haber muchos elementos de raíz en un control de vista de árbol.
Ahora vamos a examinar la estructura **TV_INSERTSTRUCT**

TV_INSERTSTRUCT STRUCT DWORD
hParent DWORD?
hInsertAfter DWORD?
ItemType <>
TV_INSERTSTRUCT TERMINA

hParent = identifica el elemento principal. Si este miembro es el valor **TVI_ROOT** o NULL, el elemento se inserta en la raíz del control tree-view.
hInsertAfter = identifica el artículo después de que el nuevo elemento se va a insertar o uno de los valores siguientes:

- TVI_FIRST ==> Inserta el elemento en el principio de la lista.
- TVI_LAST ==> Inserta el elemento en la final de la lista.
- TVI_SORT ==> Inserta el elemento en la lista por orden alfabético.

```
ItemType UNIÓN
itemex TVITEMEX <>
elemento TVITEM <>
ItemType TERMINA
```

Vamos a utilizar sólo TVITEM aquí.

```
TV_ITEM STRUCT DWORD
imask DWORD?
hItem DWORD?
Estado DWORD?
stateMask DWORD?
pszText DWORD?
cchTextMax DWORD?
iImage DWORD?
iSelectedImage DWORD?
cChildren DWORD?
lParam DWORD?
TV_ITEM TERMINA
```

Esta estructura se utiliza para enviar y recibir información sobre un elemento de vista de árbol, en función de los mensajes. Por ejemplo, con **TVM_INSERTITEM,** se utiliza para especificar el atributo del elemento a ser insertado en el control de vista de árbol. Con **TVM_GETITEM,** que va a estar lleno de información sobre el elemento de vista de árbol seleccionado.

imask se utiliza para especificar qué miembro (s) de la estructura **TV_ITEM** es (son) válido. Por ejemplo, si el valor en imask es **TVIF_TEXT,** significa solamente el miembro pszText es válida. Puede combinar varias banderas juntas.

hItem es el mango hasta el punto de vista de árbol. Cada elemento tiene su propio mango, como un identificador de ventana. Si quieres hacer algo con un artículo, usted debe seleccionar por el mango.

pszText es el puntero a una cadena terminada en cero que es la etiqueta del elemento de vista de árbol.

cchTextMax sólo se utiliza cuando se desea recuperar la etiqueta del elemento de vista de árbol. Dado que va a suministrar el puntero a la memoria intermedia en el pszText, Windows tiene que saber el tamaño del búfer proporcionado. Hay que darle el tamaño del buffer en este miembro.

iImage y **iSelectedImage** se refiere al índice en una lista de imágenes que contiene las imágenes que se mostrarán cuando el elemento no está seleccionado y cuando se selecciona. Si usted recuerda panel Explorador de Windows izquierda, las imágenes de las carpetas se especifican por estos dos miembros.

Para insertar un elemento en el control de vista de árbol, se debe llenar al menos en el hParent, hInsertAfter y usted debe llenar los miembros imask y pszText así.

Adición de imágenes para el control de vista de árbol

Si quieres poner una imagen a la izquierda de la etiqueta del elemento de vista de árbol, tienes que crear una lista de imágenes y asociarlo con el control de vista de árbol. Puede crear una lista de imágenes llamando **ImageList_Create.**

```
ImageList_Create PROTO CX: DWORD, CY: DWORD, banderas: DWORD, \
cInitial: DWORD, cGrow: DWORD
```

Esta función devuelve el identificador de una lista de imágenes vacía si tiene éxito.
cx == ancho de cada imagen en esta lista de imágenes en píxeles.
cy == altura de cada imagen en esta lista de imágenes en píxeles. Cada imagen en una lista de imágenes debe ser iguales entre sí en tamaño. Si se especifica un mapa de bits grande, Windows usará cx y cy de corte * * en varios pedazos. Así que usted debe preparar su propia imagen como una tira de imágenes con dimensiones idénticas.
banderas == especificar el tipo de imágenes en esta lista de imágenes ya sean en color o blanco y negro y su profundidad de color. Consulte a su referencia de la api de win32 para obtener más detalles
cInitial == El número de imágenes que esta lista de imágenes inicialmente contienen. Windows utilizará esta información para asignar memoria para las imágenes.
cGrow == Cantidad de imágenes por las que la lista de imágenes pueden crecer cuando el sistema tiene que cambiar el tamaño de la lista para hacer espacio para nuevas imágenes. Este parámetro representa el número de nuevas imágenes que la lista puede contener la imagen cambia de tamaño.
Una lista de imágenes no es una ventana! Es sólo un depósito de imágenes para su uso por otras ventanas.
Después de una lista de imágenes se crea, puede agregar imágenes para que al llamar **ImageList_Add**

ImageList_Add himl PROTO: DWORD, hbmImage: DWORD, hbmMask: DWORD

Esta función devuelve -1 si no tiene éxito.
himl == el mango de la lista de imágenes que desea agregar imágenes a. Es el valor devuelto por una llamada con éxito a **ImageList_Create**
hbmImage == el mango para el mapa de bits que se añade a la lista de imágenes. Por lo general, almacenar el mapa de bits en el recurso y lo cargan con la llamada **LoadBitmap.** Tenga en cuenta que no tiene que especificar el número de imágenes que figuran en este mapa de bits ya que esta información se infiere de cx y cy parámetros pasado a **ImageList_Create** llamada.
hbmMask == identifica el mapa de bits que contiene la máscara. Si no se usa una máscara con la lista de imágenes, este parámetro se ignora.
Normalmente, vamos a añadir sólo dos imágenes a la lista de imágenes para su uso con el control de vista de árbol: una que se utiliza cuando el elemento de vista de árbol no está seleccionada, y el otro cuando el elemento está seleccionado.
Cuando la lista de imágenes está listo, se asocia con el control de vista de árbol mediante el envío de **TVM_SETIMAGELIST** a la vista de árbol de control.

TVM_SETIMAGELIST
wParam = tipo de lista de imagen para definir. Hay dos opciones:

- **TVSIL_NORMAL Establece la lista de imagen normal, que contiene las imágenes seleccionadas y no seleccionadas para el elemento de vista de árbol.**
- **TVSIL_STATE Establecer la lista de imágenes del estado, que contiene las imágenes para ver elementos de árboles que se encuentran en un estado definido por el usuario.**

lParam = Handle de la lista de imágenes

Recuperar la información sobre el punto de vista de árbol
Puede recuperar la información sobre un elemento de vista de árbol mediante el envío de mensajes **TVM_GETITEM.**

TVM_GETITEM

TUTORIAL 20: VENTANA DE SUBCLASIFICACIÓN

En este tutorial, vamos a aprender acerca de la creación de subclases ventana, lo que es y cómo usarlo a tu favor.
Bajar el ejemplo aquí .

TEORÍA:

Si el programa en Windows desde hace algún tiempo, usted encontrará algunos casos en los que una ventana tiene *cerca de* los atributos que necesita en su programa pero no del todo. ¿Se ha encontrado una situación en la que desea algún tipo especial de control de edición que puede filtrar un poco de texto no deseados? Lo fácil que hacer es codificar su propia ventana. Pero en realidad es un trabajo duro y requiere mucho tiempo. Ventana a la creación de subclases de rescate.
En pocas palabras, la subclasificación de la ventana le permite "hacerse cargo" de la ventana de subclase. Usted tendrá el control absoluto sobre ella. Tomemos un ejemplo para aclarar este punto. Supongamos que usted necesita un cuadro de texto que sólo acepta números hexadecimales. Si utiliza un simple control de edición, que no tienen voz alguna cuando el usuario escribe algo distinto de números hexadecimales en el cuadro de texto, es decir. si el usuario escribe "zb + q *" en el cuadro de texto, no se puede hacer nada con ella, salvo rechazo de la cadena de texto en su conjunto. Esto es *poco profesional* por lo menos. En esencia, lo que necesita la capacidad de examinar cada personaje que el usuario escriba en el cuadro de texto a la derecha en el momento en que se escribió.
Vamos a estudiar la manera de hacer eso ahora. Cuando el usuario escribe algo en un cuadro de texto, Windows envía el mensaje WM_CHAR al procedimiento de ventana del control de edición. Este procedimiento se encuentra dentro de la ventana de Windows en sí lo que no podemos modificarlo. **Sin embargo, podemos redirigir el flujo de mensajes a nuestro procedimiento de ventana.** Así que nuestro procedimiento de ventana recibirá el primer disparo en cualquier mensaje de Windows envía al control de edición. Si nuestro procedimiento de ventana decide actuar sobre el mensaje, puede hacerlo. Pero si no quiere manejar el mensaje, se puede pasar al procedimiento de ventana original. De esta manera, los insertos de procedimiento de la ventana en sí entre Windows y el control de edición. Mira el siguiente flujo:

Antes de subclases
Procedimiento de Windows ==> control de edición de la ventana

Después de subclases
Procedimiento de Windows ==> nuestro procedimiento de ventana -----> control de edición de la ventana

Ahora ponemos nuestra atención en cómo crear subclases de una ventana. Nótese que subclases no se limita a los controles, se puede utilizar con cualquier ventana.
Vamos a pensar en cómo Windows sabe dónde procedimiento de control de edición de la ventana reside. Una conjetura? ..... Miembro lpfnWndProc de estructura WNDCLASSEX. Si

podemos reemplazar este miembro con la dirección de nuestro procedimiento de ventana, Windows enviará mensajes a nuestro proc ventana de su lugar.
Podemos hacerlo llamando a SetWindowLong.

SetWindowLong hWnd PROTO: DWORD, nIndex: DWORD, dwNewLong: DWORD

hWnd = manejador de la ventana para cambiar el valor en la estructura WNDCLASSEX
nIndex == valor para cambiar.

GWL_EXSTYLE Establece un nuevo estilo de ventana extendida.
Define un estilo **GWL_STYLE** nueva ventana.
GWL_WNDPROC Establece una nueva dirección para el procedimiento de ventana.
GWL_HINSTANCE Establece un contexto de aplicación nueva instancia.
GWL_ID Establece un nuevo identificador de la ventana.
GWL_USERDATA Establece el valor de 32 bits asociado a la ventana. Cada ventana tiene su correspondiente valor de 32 bits diseñado para su uso por la aplicación que creó la ventana.

dwNewLong = el valor de reposición.
Así que nuestro trabajo es fácil: Nos codificar un proc ventana que se encargará de los mensajes para el control de edición y luego llamar a SetWindowLong con la bandera GWL_WNDPROC, pasando por la dirección de nuestra ventana de procesos como el tercer parámetro. Si la función tiene éxito, el valor de retorno es el valor anterior de la de 32 bits entero, en nuestro caso, la dirección del procedimiento de ventana original. Tenemos que almacenar este valor para su uso dentro de nuestro procedimiento de ventana.
Recuerde que habrá algunos mensajes que no quieren manejar, vamos a pasar al procedimiento de ventana original. Podemos hacerlo llamando a la función CallWindowProc.

**CallWindowProc lpPrevWndFunc PROTO: DWORD, **
**hWnd: DWORD, **
**Mensaje de error: DWORD, **
**wParam: DWORD, **
lParam: DWORD

lpPrevWndFunc = la dirección del procedimiento de ventana original.
Los otros cuatro son los parámetros pasados a nuestro procedimiento de ventana. Acabamos de pasar junto a la CallWindowProc.

Código de ejemplo:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comctl32.inc
includelib \ masm32 \ lib \ comctl32.lib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WinMain PROTO: DWORD,: DWORD,: DWORD,: DWORD
EditWndProc PROTO: DWORD,: DWORD,: DWORD,: DWORD

. Datos
ClassName db "SubclassWinClass", 0
```

```
AppName db "Demo subclases", 0
EditClass db "EDIT", 0
Db mensaje "Usted oprimió Intro en el cuadro de texto", 0

. Datos?
HINSTANCE hInstance?
hwndEdit dd?
OldWndProc dd?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
WinMain invoca, hInstance, NULL, NULL, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_APPWORKSPACE
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPED + + WS_CAPTION WS_SYSMENU + + WS_MINIMIZEBOX
WS_MAXIMIZEBOX + WS_VISIBLE, CW_USEDEFAULT, \
CW_USEDEFAULT, 350.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
. Mientras VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. Endw
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
. Si uMsg == WM_CREATE
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ADDR EditClass, NULL, \
WS_CHILD + WS_VISIBLE + WS_BORDER, 20, \
20,300,25, hWnd, NULL, \
hInstance, NULL
mov hwndEdit, eax
```

```
invocar SetFocus, eax
; ----------------------------------------
; Subclase ella!
; ----------------------------------------
invocar SetWindowLong, hwndEdit, GWL_WNDPROC, EditWndProc addr
mov OldWndProc, eax
. Elseif uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. Más
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. Endif
xor eax, eax
ret
WndProc ENDP

EditWndProc PROC hEdit: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
. Si uMsg == WM_CHAR
mov eax, wParam
. Si (al> = "0" al && <= "9") | | (al> = "A" al && <= "F") | | (al> = "A" al && <= "f" ) | | == al
VK_BACK
. Si al> = "A" al && <= "f"
sub al, 20h
. Endif
invocar CallWindowProc, OldWndProc, hEdit, uMsg, eax, lParam
ret
. Endif
. UMsg elseif == WM_KEYDOWN
mov eax, wParam
. Si al == VK_RETURN
invocar el cuadro de mensajes, hEdit, Mensaje addr, addr AppName, MB_OK +
MB_ICONINFORMATION
invocar SetFocus, hEdit
. Más
invocar CallWindowProc, OldWndProc, hEdit, uMsg, wParam, lParam
ret
. Endif
. Más
invocar CallWindowProc, OldWndProc, hEdit, uMsg, wParam, lParam
ret
. Endif
xor eax, eax
ret
EditWndProc endp
poner fin a empezar a
```

Análisis:

```
invocar SetWindowLong, hwndEdit, GWL_WNDPROC, EditWndProc addr
mov OldWndProc, eax
```

Después de que el control de edición se ha creado, lo subclase llamando a SetWindowLong, reemplazando a la dirección del procedimiento de ventana original con nuestro procedimiento de ventana. Tenga en cuenta que almacenar la dirección del procedimiento de ventana original para su uso con CallWindowProc. Tenga en cuenta la EditWndProc es un procedimiento de ventana normal.

```
. Si uMsg == WM_CHAR
mov eax, wParam
```

```
. Si (al> = "0" al && <= "9") | | (al> = "A" al && <= "F") | | (al> = "A" al && <= "f" ) | |
== al VK_BACK
. Si al> = "A" al && <= "f"
sub al, 20h
. Endif
invocar CallWindowProc, OldWndProc, hEdit, uMsg, eax, lParam
ret
. Endif
```

Dentro de EditWndProc, vamos a filtrar los mensajes WM_CHAR. Si el personaje es de entre 0-9 o af, lo aceptamos por la que pasa a lo largo del mensaje al procedimiento de ventana original. Si se trata de un carácter en minúscula, que se convierten a mayúsculas añadiendo con 20h. Tenga en cuenta que, si el personaje no es el que esperamos, la descartamos. No pasar a la ventana original proc. Así que cuando el usuario escribe algo que no sea 0-9 o af, el carácter justo no aparece en el control de edición.

```
. UMsg elseif == WM_KEYDOWN
mov eax, wParam
. Si al == VK_RETURN
invocar el cuadro de mensajes, hEdit, Mensaje addr, addr AppName, MB_OK +
MB_ICONINFORMATION
invocar SetFocus, hEdit
. Más
invocar CallWindowProc, OldWndProc, hEdit, uMsg, wParam, lParam
ret
. Final
```

Quiero demostrar el poder de la subclase más al atrapar la tecla Enter. EditWndProc comprueba mensaje WM_KEYDOWN si es VK_RETURN (la tecla Enter). Si es así, se muestra un cuadro de mensaje que dice "Se ha pulsado la tecla Enter en el cuadro de texto". Si no es una tecla Enter, pasa el mensaje al procedimiento de ventana original.
Puede utilizar la subclasificación ventana para tomar el control de otras ventanas. Es una de las técnicas de gran alcance que debe tener en su arsenal

TUTORIAL 21: PIPE

En este tutorial, vamos a explorar la tubería, lo que es y lo que podemos utilizarlo. Para hacerlo más interesante, me tiro en la técnica sobre cómo cambiar el fondo y color del texto de un control de edición.

Teoría:

La tubería es un conducto de comunicación o vía con dos extremos. Usted puede utilizar tubería para intercambiar los datos entre dos procesos diferentes, o dentro del mismo proceso. Es como un walkie-talkie. Usted da a la otra parte un conjunto y que se puede utilizar para comunicarnos con usted.
Hay dos tipos de tuberías: tuberías anónimas y con nombre. Anónimo tubería es, bueno, anónimo, es decir, se puede usar sin conocer su nombre. Una tubería con nombre es todo lo contrario: usted tiene que saber su nombre antes de poder usarlo.

También puede clasificar las tuberías de acuerdo a su propiedad: de un solo sentido o de dos vías. En un tubo de un solo sentido, los datos pueden fluir en una sola dirección: desde un extremo al otro. Mientras que en un tubo de dos vías, los datos pueden ser intercambiados entre ambos extremos.

Una tubería anónima es siempre de un solo sentido, mientras que una tubería con nombre puede ser unidireccional o bidireccional. Una tubería con nombre se utiliza generalmente en un entorno de red donde un servidor se puede conectar a varios clientes.

En este tutorial, vamos a examinar la tubería anónima con cierto detalle. Propósito principal tubo de Anónimo es para ser utilizado como una vía de communcation entre una matriz y procesos secundarios o entre procesos secundarios.

Anónimo tubería es realmente útil cuando se trabaja con una aplicación de consola. Una aplicación de consola es una especie de programa de win32 que usa la consola para su entrada y salida. Una consola es como una ventana de DOS. Sin embargo, una aplicación de consola es un programa completamente 32-bit. Se puede utilizar cualquier función de interfaz gráfica de usuario, al igual que otros programas de interfaz gráfica de usuario. Se le pasa a tener una consola para su uso.

Una aplicación de consola cuenta con tres mangos se puede utilizar para la entrada y salida. Se llaman asas estándar. Hay tres de ellos: la entrada estándar, salida estándar y error estándar. Mango de entrada estándar se utiliza para leer / recuperar la información de la consola y palanca de salida estándar se utiliza para dar salida a / imprimir la información a la consola. Mango error estándar se utiliza para informar condición de error ya que su salida no puede ser redirigida.

Una aplicación de consola puede recuperar esos tres asas estándar llamando a la función GetStdHandle, especificando el mango se quiere obtener. Un interfaz de la aplicación no tiene una consola. Si usted llama GetStdHandle, se devuelve un error. Si realmente quieres usar una consola, puede llamar a AllocConsole para asignar una nueva consola. Sin embargo, no se olvide de llamar a FreeConsole cuando haya terminado con la consola.

Anónimo tubería se utiliza más frecuentemente para redirigir la entrada y / o salida de una aplicación de consola niño. El proceso de los padres puede ser una consola o una aplicación de interfaz gráfica de usuario, pero el niño debe ser una aplicación de consola. para este trabajo. Como usted sabe, una aplicación de consola utiliza asas estándar para su entrada y salida. Si queremos redirigir la entrada y / o salida de una aplicación de consola, se puede sustituir el mango con un mango para un extremo de un tubo. Una aplicación de consola no sabrá que está utilizando un identificador a un extremo de un tubo. No voy a utilizar como un mango estándar. Se trata de un tipo de polimorfismo, en la jerga de programación orientada a objetos. Este enfoque es de gran alcance ya que no es necesario modificar el proceso hijo en cualquier caso.

Otra cosa que usted debe saber acerca de una aplicación de consola es donde se pone las normas de maneja. Cuando una aplicación de consola se crea, el proceso padre tiene dos opciones: se puede crear una nueva consola para el niño o puede dejar que el niño herede su propia consola. Para la segunda aproximación al trabajo, el proceso de los padres debe ser una aplicación de consola o si se trata de una aplicación con interfaz gráfica, debe llamar primero AllocConsole asignar una consola.

Vamos a comenzar el trabajo. Con el fin de crear una canalización anónima es necesario llamar CreatePipe. CreatePipe tiene el siguiente prototipo:

```
CreatePipe proto pReadHandle: DWORD, \
pWriteHandle: DWORD, \
pPipeAttributes: DWORD, \
nBufferSize: DWORD
```

- pReadHandle es un puntero a una variable dword que recibirá el mango hasta el final de lectura de la tubería

- pWriteHandle es un puntero a una variable dword que recibirá el mango hasta el extremo de escritura de la tubería.
- puntos pPipeAttributes a una estructura SECURITY_ATTRIBUTES que determina si la devuelve leer y escribir asas son heredables por los procesos de los niños
- nBufferSize es el tamaño recomendado de la memoria intermedia de la tubería se reserva para su uso. Este es un tamaño sugerido solamente. Usted puede utilizar NULL para indicar a la función de utilizar el tamaño predeterminado.

Si la llamada es exitosa, el valor de retorno es distinto de cero. Si no, el valor de retorno es cero.

Después de la llamada tiene éxito, obtendrá dos mangos, uno para leer extremo del tubo y el otro hasta el final de escritura. Ahora voy a esbozar los pasos necesarios para redirigir la salida estándar de un programa de consola niño a su propia process.Note que mi método es diferente de la de win32 Referencia API de Borland. El método de la API Win32 de referencia asume el proceso de los padres es una aplicación de consola y así el niño puede heredar la norma se encarga de ello. Pero la mayor parte del tiempo, tendremos que redirigir la salida de una aplicación de consola a una interfaz gráfica de usuario una.

1. Crear una tubería anónima con CreatePipe. No se olvide de establecer el miembro de la bInheritable SECURITY_ATTRIBUTES a TRUE así que las asas son heredables.
2. Ahora debemos preparar a los parámetros que se pasan a CreateProcess ya lo vamos a utilizar para cargar la aplicación de consola niño. Una estructura importante es la estructura STARTUPINFO. Esta estructura determina la apariencia de la ventana principal del proceso hijo cuando aparece por primera vez. Esta estructura es vital para nuestro propósito. Puede ocultar la ventana principal y pasar el identificador de tubería para el proceso de la consola hijo con ella. A continuación se muestra los miembros que deben llenar:
    - CB: el tamaño de la estructura STARTUPINFO
    - dwFlags: los indicadores de bits binarios que determinan qué miembros de la estructura son válidos también gobierna el mostrar / ocultar el estado de la ventana principal. Para nuestro propósito, debe utilizar una combinación de STARTF_USESHOWWINDOW y STARTF_USESTDHANDLES
    - hStdOutput y hStdError: los mangos desea que el proceso hijo para utilizar como estándar de salida / error asas. Para nuestro propósito, vamos a pasar a escribir el mango de la tubería en la salida estándar y el error del niño. Así que cuando el niño emite algo que el error estándar de salida de /, lo que realmente pasa la información a través de la tubería para que el proceso padre.
    - wShowWindow gobierna el estado Mostrar / Ocultar de la ventana principal. Para nuestro propósito, no queremos que la ventana de la consola del niño para demostrar así que pusimos SW_HIDE a este miembro.
3. Llame a CreateProcess para cargar la aplicación de niño. Después de CreateProcess tiene éxito, el niño todavía está latente. Se cargan en la memoria pero no se ejecuta de inmediato
4. Cierre el mango de tubo de escritura. Esto es necesario. Debido a que el proceso padre no tiene ningún uso para el mango de tubo de escritura, y el tubo no va a funcionar si hay más de uno de los extremos de escritura, hay que cerrar antes de la lectura de los datos de la tubería. Sin embargo, no cierra el identificador de escritura antes de llamar a CreateProcess, la pipa se romperá. Usted debe cerrar justo después de que CreateProcess y antes de leer los datos de lectura al final de la tubería.
5. Ahora usted puede leer los datos de lectura al final de la tubería con ReadFile. Con ReadFile, que comenzará el proceso de niño en modo de ejecución. Se iniciará la ejecución y cuando se escribe algo en la manija de salida estándar (que es en realidad el mango hasta el extremo de escritura de la tubería), los datos se envían a través de la tubería hasta el final leer. Usted puede pensar en ReadFile como chupar los datos de lectura al final de la tubería. Usted debe llamar a ReadFile repetidamente hasta que devuelva 0, lo que significa que hay no hay más datos para ser leídos. Usted puede hacer cualquier cosa con los datos que lee de la tubería. En nuestro ejemplo, las puse en un control de edición.

6. Cierre la manija del tubo de lectura.

Ejemplo:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ gdi32.inc
includelib \ masm32 \ lib \ gdi32.lib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WinMain PROTO: DWORD,: DWORD,: DWORD,: DWORD

Const.
IDR_MAINMENU equ 101; el identificador del menú principal
IDM_ASSEMBLE equ 40001

. Datos
ClassName db "PipeWinClass", 0
AppName db "One-way Ejemplo de tuberías", 0 db EditClass "EDIT", 0
CreatePipeError db "Error durante la creación de la tubería", 0
Db CreateProcessError "Error durante el proceso de creación", 0
CommandLine db "ml / c / coff / Cp test.asm", 0

. Datos?
HINSTANCE hInstance?
hwndEdit dd?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
WinMain invoca, hInstance, NULL, NULL, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: DWORD, hPrevInst: DWORD, CmdLine: DWORD, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_APPWORKSPACE
mov wc.lpszMenuName, IDR_MAINMENU
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
```

```
 wc.hCursor mov, eax
 invocar RegisterClassEx, addr wc
 invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPEDWINDOW + WS_VISIBLE, CW_USEDEFAULT, \ CW_USEDEFAULT,
400.200, NULL, NULL, \ hInst, NULL
 mov hwnd, eax
 . Mientras VERDADERO
 invocar GetMessage, msg ADDR, NULL, 0,0
 . INTER. If (! Eax)
 invocar TranslateMessage, ADDR msg
 invocar DispatchMessage, ADDR msg
 . Endw
 mov eax, msg.wParam
 ret
 WinMain endp

 WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
 LOCAL rect: RECT
 LOCAL HRead: DWORD
 LOCAL hWrite: DWORD
 LOCAL STARTUPINFO: STARTUPINFO
 LOCAL pinfo: PROCESS_INFORMATION
 Búfer local [1024]: bytes
 LOCAL BytesRead: DWORD
 LOCAL hdc: DWORD
 LOCAL se sentó: SECURITY_ATTRIBUTES
 . Si uMsg == WM_CREATE
 invocar CreateWindowEx, NULL, EditClass addr, NULL, WS_CHILD + + WS_VISIBLE
ES_MULTILINE + + ES_AUTOHSCROLL ES_AUTOVSCROLL, 0, 0, 0, 0, hWnd, NULL,
hInstance, NULL
 mov hwndEdit, eax
 . UMsg elseif == WM_CTLCOLOREDIT
 invocar SetTextColor, wParam, Amarillo
 invocar SetBkColor, wParam, Negro
 invocar GetStockObject, BLACK_BRUSH
 ret
 . Elseif uMsg == WM_SIZE
 mov edx, lParam
 mov ecx, edx
 shr ecx, 16
 y edx, 0FFFFh
 invocar MoveWindow, hwndEdit, 0,0, EDX, ECX, TRUE
 . Elseif uMsg == WM_COMMAND
 . Si lParam == 0
 mov eax, wParam
 . Si ax == IDM_ASSEMBLE
 sat.nLength mov, SECURITY_ATTRIBUTES sizeof
 mov sat.lpSecurityDescriptor, NULL
 mov sat.bInheritHandle, TRUE
 invocar CreatePipe, HRead addr, hWrite addr, addr sáb, NULL
 . Si eax == NULL
 invocar el cuadro de mensajes, hWnd, CreatePipeError addr, addr AppName,
MB_ICONERROR + MB_OK
 . Más
 mov startupinfo.cb, sizeof STARTUPINFO
 invocar GetStartupInfo, addr STARTUPINFO
 mov eax, hWrite
 mov startupinfo.hStdOutput, eax
 mov startupinfo.hStdError, eax
 mov startupinfo.dwFlags, STARTF_USESHOWWINDOW + STARTF_USESTDHANDLES
```

```
 mov startupinfo.wShowWindow, SW_HIDE
 invocar CreateProcess, NULL, addr CommandLine, NULL, NULL, TRUE, NULL NULL,
NULL, STARTUPINFO addr, addr pinfo
 . Si eax == NULL
 invocar el cuadro de mensajes, hWnd, CreateProcessError addr, addr AppName,
MB_ICONERROR + MB_OK
 . Más
 invocar CloseHandle, hWrite
 . Mientras VERDADERO
 invocar RtlZeroMemory, addr buffer, 1024
 invocar ReadFile, HRead, addr buffer, 1023, dir BytesRead, NULL
 . Si eax == NULL
 . Descanso
 . Endif
 invoca SendMessage, hwndEdit, EM_SETSEL, -1,0
 invoca SendMessage, hwndEdit, EM_REPLACESEL, FALSO, addr tampón
 . Endw
 . Endif
 invocar CloseHandle, HRead
 . Endif
 . Endif
 . Endif
 . Elseif uMsg == WM_DESTROY
 invocar PostQuitMessage, NULL
 . Más
 invocar DefWindowProc, hWnd, uMsg, wParam, lParam ret
 . Endif
 xor eax, eax
 ret
 WndProc ENDP
 poner fin a empezar a
```

Análisis:

El ejemplo se llama ml.exe para montar un archivo llamado test.asm y redirigir la salida de la ml.exe al control de edición en su área de cliente.

Cuando se carga el programa, se registra la clase de ventana y crea la ventana principal como de costumbre. Lo primero que hace durante la creación de la ventana principal es crear un control de edición que se utiliza para mostrar el resultado de ml.exe.

Ahora la parte interesante, vamos a cambiar el texto y color de fondo del control de edición. Cuando un control de edición se va a pintar su área cliente, que envía el mensaje WM_CTLCOLOREDIT a su padre.

wParam contiene el manejador del contexto de dispositivo que el control de edición se utiliza para escribir su propia área cliente. Podemos aprovechar esta oportunidad para modificar las características de la HDC.

```
 . UMsg elseif == WM_CTLCOLOREDIT
 invocar SetTextColor, wParam, Amarillo
 invocar SetTextColor, wParam, Negro
 invocar GetStockObject, BLACK_BRUSH
 ret
```

SetTextColor cambia el color del texto a amarillo. SetTextColor cambia el color de fondo del texto a negro. Y, por último, se obtiene el mango del cepillo negro que volver a Windows. Con el mensaje de WM_CTLCOLOREDIT, debe devolver un identificador a un pincel que Windows usará para pintar el fondo del control de edición. En nuestro ejemplo, quiero que el fondo sea

negro, así que devuelva la palanca a la brocha negro para Windows.
Ahora cuando el usuario selecciona Monte menuitem, crea una canalización anónima.

```
. Si ax == IDM_ASSEMBLE
sat.nLength mov, SECURITY_ATTRIBUTES sizeof
mov sat.lpSecurityDescriptor, NULL
mov sat.bInheritHandle, TRUE
```

Antes de llamar a CreatePipe, debemos llenar la estructura SECURITY_ATTRIBUTES primero. Tenga en cuenta que podemos usar NULL en miembro lpSecurityDescriptor si no se preocupan por la seguridad. Y el miembro bInheritHandle debe ser TRUE para que los mangos de tubo se pueden heredar al proceso hijo.

```
invocar CreatePipe, HRead addr, hWrite addr, addr sáb, NULL
```

Después de eso, hacemos un llamamiento CreatePipe que, si tiene éxito, va a llenar HRead y las variables hWrite con las manijas para leer y escribir extremos de la tubería, respectivamente.

```
mov startupinfo.cb, sizeof STARTUPINFO
invocar GetStartupInfo, addr STARTUPINFO
mov eax, hWrite
mov startupinfo.hStdOutput, eax
mov startupinfo.hStdError, eax
mov startupinfo.dwFlags, STARTF_USESHOWWINDOW + STARTF_USESTDHANDLES
mov startupinfo.wShowWindow, SW_HIDE
```

A continuación se debe rellenar la estructura STARTUPINFO. Hacemos un llamado GetStartupInfo para llenar la estructura STARTUPINFO con valores por defecto del proceso padre. Usted debe llenar la estructura STARTUPINFO con esta llamada, si usted tiene la intención de su código para que funcione en ambas win9x y NT. Después devuelve la llamada GetStartupInfo, puede modificar los miembros que son importantes. Copiamos el mango hasta el extremo de escritura de la tubería en hStdOutput y hStdError ya que queremos que el proceso hijo para utilizarlo en lugar de los predeterminados estándar de salida / error asas. También queremos ocultar la ventana de la consola del proceso hijo, así que pusimos un valor en el miembro SW_HIDE wShowWidow. Y, por último, debemos señalar que los miembros hStdOutput, hStdError y wShowWindow son válidos y deben ser utilizados, especificando el STARTF_USESHOWWINDOW banderas y STARTF_USESTDHANDLES en miembro dwFlags.

```
invocar CreateProcess, NULL, addr CommandLine, NULL, NULL, TRUE, NULL NULL,
NULL, STARTUPINFO addr, addr pinfo
```

Ahora crear el proceso hijo con la llamada de CreateProcess. Tenga en cuenta que el parámetro bInheritHandles se debe establecer en TRUE para el tubo de la manija para trabajar.

```
invocar CloseHandle, hWrite
```

Después de crear con éxito el proceso hijo, tenemos que cerrar el extremo de escritura de la tubería. Recuerde que pasamos por el mango en el proceso de escritura infantil a través de la estructura STARTUPINFO. Si no cierra el identificador de escritura de nuestra parte, habrá dos extremos de escritura. Y que el tubo no va a funcionar. Debemos cerrar el identificador de escritura después de CreateProcess pero antes de leer los datos desde el extremo de lectura de la tubería.

```
. Mientras VERDADERO
invocar RtlZeroMemory, addr buffer, 1024
invocar ReadFile, HRead, addr buffer, 1023, dir BytesRead, NULL
```

```
. Si eax == NULL
. Descanso
. Endif
invoca SendMessage, hwndEdit, EM_SETSEL, -1,0
invoca SendMessage, hwndEdit, EM_REPLACESEL, FALSO, addr tampón
. Endw
```

Ahora estamos listos para leer los datos de la salida estándar del proceso hijo. Nos quedaremos en un bucle infinito hasta que haya datos sin más a la izquierda para leer de la tubería. Hacemos un llamado RtlZeroMemory para llenar el búfer con ceros a continuación, llamar a ReadFile, pasando por el mango de lectura de la tubería en lugar de un identificador de archivo. Tenga en cuenta que sólo leemos un máximo de 1023 bytes, ya que necesitamos los datos para ser una cadena ASCIIZ que podemos pasar al control de edición.
Cuando ReadFile regresa con los datos en el búfer, llenamos los datos en el control de edición. Sin embargo, hay un pequeño problema aquí. Si usamos SetWindowText para poner los datos en el control de edición, los nuevos datos se sobreponen a los datos existentes! Queremos que los datos para añadir al final de los datos existentes.
Para lograr ese objetivo, lo primero que mover el cursor hasta el final del texto en el control de edición mediante el envío de mensajes EM_SETSEL con wParam == -1. A continuación, añadir los datos en ese punto con el mensaje de EM_REPLACESEL.

```
invocar CloseHandle, HRead
```

Cuando ReadFile devuelve NULL, salimos del bucle y cerrar el identificador de lectura.

TUTORIAL 22: SUPERCLASSING

En este tutorial, vamos a aprender acerca superclassing, lo que es y para qué sirve. También aprenderá a proporcionar una navegación tecla de tabulación para los controles en su propia ventana.

TEORÍA:

En su carrera de programación, seguramente se encontrará con una situación donde usted necesita varios controles con * levemente * comportamiento diferente. Por ejemplo, es posible que tenga 10 controles de edición que acepten sólo el número. Hay varias maneras de lograr ese objetivo:

- Crea tu propia clase y una instancia de los controles
- Crear los control de edición y, a continuación subclase todos ellos
- Superclase el control de edición

El primer método es demasiado tedioso. Usted tiene que poner en práctica todas las funcionalidades de la edición de controlar a ti mismo. Apenas una tarea que debe tomarse a la ligera. El segundo método es mejor que el primero pero todavía demasiado trabajo. No es buena idea si usted subclase sólo unos pocos controles, sino que va a ser una pesadilla para la subclase de una docena de controles menos. Superclassing es la técnica que puedes usar para esta ocasión.
La subclasificación es el método que se utiliza para tomar * * el control de una clase de ventana en particular. Por * control * de tomar, me refiero a que puede modificar las

propiedades de la clase de ventana para adaptarse a su propósito, entonces a continuación, crear el grupo de controles.
Los pasos en superclassing se detalla a continuación:

- llame GetClassInfoEx para obtener la información sobre la clase de ventana que desea superclase. GetClassInfoEx requiere un puntero a una estructura WNDCLASSEX que será llenada con la información si la llamada se devuelve correctamente.
- Modificar los miembros WNDCLASSEX que desee. Sin embargo, hay dos miembros que se debe modificar:
    - hInstance Debes poner el identificador de instancia de su programa en este miembro.
    - lpszClassName Usted debe proporcionar un puntero a un nuevo nombre de clase.
      No es necesario modificar el miembro lpfnWndProc pero la mayoría de las veces, es necesario hacerlo. Sólo recuerde guardar el miembro original lpfnWndProc si quieres llamarlo con CallWindowProc.
- Registrar el modifed estructura WNDCLASSEX. Usted tendrá una nueva clase de ventana que tiene varias características de la clase de ventana antigua.
- Crear ventanas de la nueva clase

Superclassing es mejor que la creación de subclases, si desea crear muchos controles con las mismas características.

Ejemplo:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WM_SUPERCLASS equ WM_USER 5
WinMain PROTO: DWORD,: DWORD,: DWORD,: DWORD
EditWndProc PROTO: DWORD,: DWORD,: DWORD,: DWORD

. Datos
ClassName db "SuperclassWinClass", 0
AppName db "Demo Superclassing", 0
EditClass db "EDIT", 0
OurClass db "SUPEREDITCLASS", 0
Db mensaje "Se ha pulsado la tecla Enter en el cuadro de texto", 0

. Datos?
hInstance dd?
hwndEdit dd 6 dup (?)
OldWndProc dd?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
```

```
 WinMain invoca, hInstance, NULL, NULL, SW_SHOWDEFAULT
 invocar ExitProcess, eax

 WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
 LOCAL wc: WNDCLASSEX
 Msg LOCAL: MSG
 LOCAL hwnd: HWND

 mov wc.cbSize, sizeof WNDCLASSEX
 mov wc.style, CS_HREDRAW o CS_VREDRAW
 mov wc.lpfnWndProc, OFFSET WndProc
 mov wc.cbClsExtra, NULL
 mov wc.cbWndExtra, NULL
 impulsar hInst
 pop wc.hInstance
 mov wc.hbrBackground, COLOR_APPWORKSPACE
 mov wc.lpszMenuName, NULL
 wc.lpszClassName mov, ClassName OFFSET
 invocar LoadIcon, NULL, IDI_APPLICATION
 mov wc.hIcon, eax
 mov wc.hIconSm, eax
 invocar LoadCursor, NULL, IDC_ARROW
 wc.hCursor mov, eax
 invocar RegisterClassEx, addr wc
 invocar CreateWindowEx, WS_EX_CLIENTEDGE + WS_EX_CONTROLPARENT,
ClassName ADDR, ADDR AppName, \
 WS_OVERLAPPED + + WS_CAPTION WS_SYSMENU + + WS_MINIMIZEBOX
WS_MAXIMIZEBOX + WS_VISIBLE, CW_USEDEFAULT, \
 CW_USEDEFAULT, 350.220, NULL, NULL, \
 hInst, NULL
 mov hwnd, eax

 . Mientras VERDADERO
 invocar GetMessage, msg ADDR, NULL, 0,0
 . INTER. If (! Eax)
 invocar TranslateMessage, ADDR msg
 invocar DispatchMessage, ADDR msg
 . Endw
 mov eax, msg.wParam
 ret
 WinMain endp

 WndProc proc utiliza ebx edición hWnd: HWND, uMsg: UINT, wParam: wParam, lParam:
LPARAM
 LOCAL wc: WNDCLASSEX
 . Si uMsg == WM_CREATE
 mov wc.cbSize, sizeof WNDCLASSEX
 invocar GetClassInfoEx, NULL, EditClass addr, addr wc
 impulsar wc.lpfnWndProc
 pop OldWndProc
 mov wc.lpfnWndProc, OFFSET EditWndProc
 impulsar hInstance
 pop wc.hInstance
 wc.lpszClassName mov, OFFSET OurClass
 invocar RegisterClassEx, addr wc
 xor ebx, ebx
 MOV EDI, 20
 . Mientras ebx <6
 invocar CreateWindowEx, WS_EX_CLIENTEDGE, ADDR OurClass, NULL, \
```

```
WS_CHILD + WS_VISIBLE + WS_BORDER, 20, \
edición, 300,25, hWnd, ebx, \
hInstance, NULL
mov dword ptr [hwndEdit 4 * ebx], eax
añadir edición, 25
inc ebx
. Endw
invocar SetFocus, hwndEdit
. Elseif uMsg == WM_DESTROY
invocar PostQuitMessage, NULL
. Más
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. Endif
xor eax, eax
ret
WndProc ENDP

EditWndProc PROC hEdit: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
. Si uMsg == WM_CHAR
mov eax, wParam
. Si (al> = "0" al && <= "9") | | (al> = "A" al && <= "F") | | (al> = "A" al && <= "f" ) | | == al
VK_BACK
. Si al> = "A" al && <= "f"
sub al, 20h
. Endif
invocar CallWindowProc, OldWndProc, hEdit, uMsg, eax, lParam
ret
. Endif
. UMsg elseif == WM_KEYDOWN
mov eax, wParam
. Si al == VK_RETURN
invocar el cuadro de mensajes, hEdit, Mensaje addr, addr AppName, MB_OK +
MB_ICONINFORMATION
invocar SetFocus, hEdit
. Al elseif == VK_TAB
invocar VK_SHIFT GetKeyState,
test eax, 80000000
. Si es cero?
invocar GetWindow, hEdit, GW_HWNDNEXT
. Si eax == NULL
invocar GetWindow, hEdit, GW_HWNDFIRST
. Endif
. Más
invocar GetWindow, hEdit, GW_HWNDPREV
. Si eax == NULL
invocar GetWindow, hEdit, GW_HWNDLAST
. Endif
. Endif
invocar SetFocus, eax
xor eax, eax
ret
. Más
invocar CallWindowProc, OldWndProc, hEdit, uMsg, wParam, lParam
ret
. Endif
. Más
invocar CallWindowProc, OldWndProc, hEdit, uMsg, wParam, lParam
ret
. Endif
```

```
xor eax, eax
ret
EditWndProc endp
poner fin a empezar a
```

Análisis:

El programa creará una simple ventana con 6 "modificado" los controles de edición en su área de cliente. Los controles de edición sólo aceptará dígitos hexadecimales. En realidad, he modificado el ejemplo de la creación de subclases de hacer superclassing. El programa se inicia con normalidad y la parte interesante es cuando la ventana principal se crea:

```
. Si uMsg == WM_CREATE
mov wc.cbSize, sizeof WNDCLASSEX
invocar GetClassInfoEx, NULL, EditClass addr, addr wc
```

En primer lugar, debe rellenar la estructura WNDCLASSEX con los datos de la clase que queremos superclase, en este caso, es la clase EDIT. Recuerde que debe establecer el miembro de la estructura cbSize WNDCLASSEX antes de llamar a otra persona GetClassInfoEx la estructura WNDCLASSEX no se llena correctamente. Después vuelve GetClassInfoEx, wc está lleno de toda la información que necesitamos para crear una nueva clase de ventana.

```
impulsar wc.lpfnWndProc
pop OldWndProc
mov wc.lpfnWndProc, OFFSET EditWndProc
impulsar hInstance
pop wc.hInstance
wc.lpszClassName mov, OFFSET OurClass
```

Ahora tenemos que modificar algunos de los miembros de wc. El primero es el puntero al procedimiento de ventana. Puesto que necesitamos a la cadena de nuestro procedimiento de ventana con el original, hay que guardarlo en una variable por lo que podemos llamar con CallWindowProc. Esta técnica es idéntica a la herencia, salvo que se modifique la estructura WNDCLASSEX directamente sin tener que llamar a SetWindowLong. Los próximos dos miembros deben ser cambiados de lo contrario no será capaz de registrar su nueva clase de ventana, hInstance y lpsClassName. Debe reemplazar el valor original hInstance con hInstance de su propio programa. Y usted debe elegir un nuevo nombre para la nueva clase.

```
invocar RegisterClassEx, addr wc
```

Cuando todo está listo, el registro de la nueva clase. Usted recibirá una nueva clase con algunas características de la clase de edad.

```
xor ebx, ebx
MOV EDI, 20
. Mientras ebx <6
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ADDR OurClass, NULL, \
WS_CHILD + WS_VISIBLE + WS_BORDER, 20, \
edición, 300,25, hWnd, ebx, \
hInstance, NULL
mov dword ptr [hwndEdit 4 * ebx], eax
añadir edición, 25
inc ebx
. Endw
invocar SetFocus, hwndEdit
```

Ahora que se registró la clase, podemos crear ventanas basadas en ella. En el código anterior, yo uso ebx como el contador del número de ventanas creadas. EDI se utiliza como la coordenada y de la esquina superior izquierda de la ventana. Cuando se crea una ventana, su asa se almacena en la matriz de dwords. Cuando todas las ventanas se crean, se establece el foco de entrada a la primera ventana.
En este punto, usted tiene 6 controles de edición que acepten sólo dígitos hexadecimales. El proc ventana sustituido maneja el filtro. En realidad, es idéntica a la ventana de procesos en la subclasificación ejemplo. Como puede ver, usted no tiene que hacer trabajo extra de la subclasificación ellos.

Me tiro en un fragmento de código para manejar el control de la navegación con pestañas para hacer este ejemplo más jugosa. Normalmente, si pones los controles de un cuadro de diálogo, el cuadro de diálogo Administrador se encarga de las teclas de navegación para que usted pueda pestaña para ir al siguiente control, o Shift + Tabulador para volver al control anterior. Por desgracia, dicha función no está disponible si usted pone los controles en una ventana simple. Usted tiene que subclase ellos por lo que puede manejar las teclas de tabulación a ti mismo. En nuestro ejemplo, no tenemos por qué subclase los controles, uno por uno porque ya superclase ellos, para que podamos ofrecer un "gerente de control central de navegación" para ellos.

```
. Al elseif == VK_TAB
invocar VK_SHIFT GetKeyState,
test eax, 80000000
. Si es cero?
invocar GetWindow, hEdit, GW_HWNDNEXT
. Si eax == NULL
invocar GetWindow, hEdit, GW_HWNDFIRST
. Endif
. Más
invocar GetWindow, hEdit, GW_HWNDPREV
. Si eax == NULL
invocar GetWindow, hEdit, GW_HWNDLAST
. Endif
. Endif
invocar SetFocus, eax
xor eax, eax
ret
```

El fragmento de código de arriba es de procedimiento EditWndClass. Se comprueba si el usuario presiona tecla Tab, si es así, llame a GetKeyState para comprobar si la tecla SHIFT también es presionado. GetKeyState devuelve un valor en eax que determina si la clave especificada está presionado o no. Si se pulsa la tecla, el bit de EAX se establece. Si no, el bit alto es clara. Por lo tanto, comprobar el valor de retorno contra 80000000H. Si el bit está establecido, significa que el usuario pulsa SHIFT + TAB, que debemos manejar por separado.
Si el usuario presiona tecla Tab solos, que nosotros llamamos GetWindow para recuperar el identificador del control siguiente. Usamos la bandera GW_HWNDNEXT decir GetWindow para obtener el handle de la ventana que está al lado en la línea de hEdit actual. Si esta función devuelve NULL, lo interpretamos como el mango no más para obtener lo que el hEdit actual es el último control en la línea. Vamos a "envolver" al primer control llamando GetWindow con la bandera GW_HWNDFIRST. Al igual que en el caso del Tab, Shift-Tab simplemente funciona a la inversa.

TUTORIAL 23: ICONO DE LA BANDEJA

En este tutorial, vamos a aprender a poner los iconos en la bandeja del sistema y cómo crear / usar un menú emergente.

TEORÍA:

La bandeja del sistema es la región rectangular en la barra de tareas, donde residen varios iconos. Normalmente, usted verá por lo menos un reloj digital en el mismo. También puede poner los iconos en la bandeja del sistema también. A continuación se muestran los pasos que tienen que realizar para poner un icono en la bandeja del sistema:

1. Llene una estructura NOTIFYICONDATA que tiene los siguientes miembros:
   - cbSize El tamaño de esta estructura.
   - hwnd Handle de la ventana que va a recibir una notificación cuando se produce un evento de ratón sobre el icono de la bandeja.
   - UID Una constante que se utiliza como identificador del icono. Tú eres el que decide sobre este valor. En caso de tener más de un iconos de la bandeja, usted será capaz de verificar de qué icono de la bandeja de la notificación del ratón es de.
   - uFlags especificar qué miembros de esta estructura son válidos
     - **NIF_ICON** El miembro hIcon es válido.
     - El miembro **NIF_MESSAGE** uCallbackMessage es válida.
     - El miembro **NIF_TIP** szTip es válida.
   - uCallbackMessage El mensaje personalizado que Windows enviará a la ventana especificada por el miembro hwnd, cuando se producen los eventos del ratón sobre el icono de la bandeja. Se crea este mensaje a ti mismo.
   - hIcon El mango del icono que desea poner en la bandeja del sistema
   - szTip una matriz de 64 bytes que contiene la cadena que se utiliza como texto de información sobre herramientas cuando el puntero del ratón sobre el icono de la bandeja.
2. Llama Shell_NotifyIcon que se define en shell32.inc. Esta función tiene el siguiente prototipo:

Shell_NotifyIcon dwMessage PROTO: DWORD, pnid: DWORD

dwMessage es el tipo de mensaje para enviar a la cáscara.
NIM_ADD Agrega un icono para el área de estado.
Elimina **NIM_DELETE** un icono del área de estado.
NIM_Modify Modifica un icono en el área de estado.
pnid es el puntero a una estructura NOTIFYICONDATA lleno de valores propios
Si quiere añadir un icono a la bandeja, utilice NIM_ADD mensaje, si desea eliminar el icono, utilice NIM_DELETE.

Eso es todo lo que hay que hacer. Pero la mayoría de las veces, usted no está contenido en un solo poner un icono allí. ¡Tienes que ser capaz de responder a los eventos del ratón sobre el icono de la bandeja. Usted puede hacer esto mediante el procesamiento del mensaje que se especifica en el elemento de la estructura uCallbackMessage NOTIFYICONDATA. Este mensaje tiene los siguientes valores en wParam y lParam (Gracias especiales a s__d de la información):

- wParam contiene el identificador del icono. Este es el mismo valor que usted pone en miembro de la UID de la estructura NOTIFYICONDATA.
- lParam La palabra baja contiene el mensaje de ratón. Por ejemplo, si el usuario hace clic derecho en el icono, lParam contendrá WM_RBUTTONDOWN.

La mayoría de icono de la bandeja, sin embargo, muestra un menú emergente cuando el usuario haga clic en él. Podemos implementar esta función mediante la creación de un menú emergente y luego llamar a TrackPopupMenu para que aparezca. Los pasos se describen a continuación:

1. Crear un menú desplegable llamando CreatePopupMenu. Esta función crea un menú vacío. Se devuelve el identificador del menú en eax si tiene éxito.
2. Agregar elementos de menú con AppendMenu, InsertMenu o InsertMenuItem.
3. Cuando se desea que aparezca el menú emergente donde el cursor del ratón es, llame GetCursorPos para obtener la coordenada de pantalla del cursor y luego llamar a TrackPopupMenu para visualizar el menú. Cuando el usuario selecciona un elemento de menú en el menú emergente, Windows envía el mensaje WM_COMMAND a su procedimiento de ventana al igual que la selección del menú normal.

Nota: Tenga cuidado con los dos comportamientos molestos cuando se utiliza un menú desplegable con un icono de la bandeja:

1. Cuando aparezca el menú emergente se muestra, si hace clic en cualquier parte fuera del menú, el menú que aparece no va a desaparecer inmediatamente, ya que debe ser. Este comportamiento se produce porque la ventana que va a recibir las notificaciones en el menú emergente debe ser la ventana en primer plano. Sólo tiene que llamar SetForegroundWindow lo corregiremos.
2. Después de llamar a SetForegroundWindow, usted encontrará que la primera vez que se visualiza el menú emergente, que funciona bien, pero en los tiempos posteriores, en el menú emergente se mostrará y cerrar inmediatamente. Este comportamiento es "intencional", para citar de MSDN. El cambio de tarea para el programa que es el propietario del icono de la bandeja en un futuro próximo es necesario. Usted puede forzar este cambio de tarea al publicar cualquier mensaje a la ventana del programa. Sólo tiene que utilizar PostMessage no SendMessage!

Ejemplo:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ shell32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ shell32.lib

WM_SHELLNOTIFY equ WM_USER 5
IDI_TRAY equ 0
IDM_RESTORE equ 1000
IDM_EXIT equ 1010
WinMain PROTO: DWORD,: DWORD,: DWORD,: DWORD

. Datos
ClassName db "TrayIconWinClass", 0
AppName db "Demo TrayIcon", 0
RestoreString db "y restauración", 0
ExitString db "Programa E & Salir", 0

. Datos?
hInstance dd?
tenga en cuenta NOTIFYICONDATA <>
hPopupMenu dd?
```

```
. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
WinMain invoca, hInstance, NULL, NULL, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR, CmdShow:
DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW o CS_DBLCLKS
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInst
pop wc.hInstance
mov wc.hbrBackground, COLOR_APPWORKSPACE
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
invocar CreateWindowEx, WS_EX_CLIENTEDGE, ClassName ADDR, ADDR AppName, \
WS_OVERLAPPED + + WS_CAPTION WS_SYSMENU + + WS_MINIMIZEBOX
WS_MAXIMIZEBOX + WS_VISIBLE, CW_USEDEFAULT, \
CW_USEDEFAULT, 350.200, NULL, NULL, \
hInst, NULL
mov hwnd, eax
. Mientras VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. INTER. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. Endw
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
Locales del PT: PUNTO
. Si uMsg == WM_CREATE
invocar CreatePopupMenu
mov hPopupMenu, eax
invocar AppendMenu, hPopupMenu, MF_STRING, IDM_RESTORE, addr RestoreString
invocar AppendMenu, hPopupMenu, MF_STRING, IDM_EXIT, addr ExitString
. Elseif uMsg == WM_DESTROY
invocar DestroyMenu, hPopupMenu
invocar PostQuitMessage, NULL
. Elseif uMsg == WM_SIZE
. Si wParam == SIZE_MINIMIZED
mov note.cbSize, sizeof NOTIFYICONDATA
impulsar hWnd
pop note.hwnd
mov note.uID, IDI_TRAY
```

```
mov note.uFlags, NIF_ICON + + NIF_MESSAGE NIF_TIP
mov note.uCallbackMessage, WM_SHELLNOTIFY
invocar LoadIcon, NULL, IDI_WINLOGO
mov note.hIcon, eax
invocar lstrcpy, addr note.szTip, addr AppName
invocar ShowWindow, hWnd, SW_HIDE
invocar Shell_NotifyIcon, NIM_ADD, nota addr
. Endif
. Elseif uMsg == WM_COMMAND
. Si lParam == 0
invocar Shell_NotifyIcon, NIM_DELETE, nota addr
mov eax, wParam
. Si ax == IDM_RESTORE
invocar ShowWindow, hWnd, SW_RESTORE
. Más
invocar DestroyWindow, hWnd
. Endif
. Endif
. UMsg elseif == WM_SHELLNOTIFY
. Si wParam == IDI_TRAY
. Si lParam == WM_RBUTTONDOWN
invocar GetCursorPos, addr pt
invocar SetForegroundWindow, hWnd
invocar TrackPopupMenu, hPopupMenu, TPM_RIGHTALIGN, pt.x, pt.y, NULL, hWnd,
NULL
invocar PostMessage, hWnd, WM_NULL, 0,0
. Elseif lParam == WM_LBUTTONDBLCLK
invoca SendMessage, hWnd, WM_COMMAND, IDM_RESTORE, 0
. Endif
. Endif
. Más
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. Endif
xor eax, eax
ret
WndProc ENDP

poner fin a empezar a
```

Análisis:

El programa mostrará una ventana simple. Cuando se presiona el botón de minimizar, se va a ocultar y poner un icono en la bandeja de sistema. Al hacer doble clic en el icono, el programa se restablecerá y eliminar el icono de la bandeja del sistema. Al hacer clic en él, un menú desplegable en la pantalla. Usted puede elegir para restaurar el programa o salir de ella.

```
. Si uMsg == WM_CREATE
invocar CreatePopupMenu
mov hPopupMenu, eax
invocar AppendMenu, hPopupMenu, MF_STRING, IDM_RESTORE, addr RestoreString
invocar AppendMenu, hPopupMenu, MF_STRING, IDM_EXIT, addr ExitString
```

Cuando aparezca la ventana principal se crea, se crea un menú emergente y añadir dos elementos de menú. AppendMenu tiene la siguiente sintaxis:

AppendMenu hMenu PROTO: DWORD, uFlags: DWORD, uIDNewItem: DWORD, lpNewItem: DWORD

- hMenu es el identificador del menú que desea añadir el artículo a
- uFlags le dice a Windows sobre el elemento de menú que se añade al menú si se trata de un mapa de bits o una cadena o un elemento de propietario del dibujo, habilitado, gris o desactivar, etc Usted puede obtener la lista completa de referencia de la api de win32. En nuestro ejemplo, usamos MF_STRING lo que significa que el elemento de menú es una cadena.
- uIDNewItem es el identificador del elemento de menú. Este es un valor definido por el usuario que se utiliza para representar el elemento de menú.
- lpNewItem especifica el contenido del elemento de menú, dependiendo de lo que se especifica en el miembro uFlags. Como se especifica en el miembro MF_STRING uFlags, lpNewItem debe contener el puntero a la cadena que se mostrará en el menú emergente.

Después en el menú emergente se crea la ventana principal espera pacientemente a que el usuario presione botón de minimizar.
Cuando una ventana está minimizada, recibe el mensaje con el valor WM_SIZE SIZE_MINIMIZED en wParam.

```
. Elseif uMsg == WM_SIZE
. Si wParam == SIZE_MINIMIZED
mov note.cbSize, sizeof NOTIFYICONDATA
impulsar hWnd
pop note.hwnd
mov note.uID, IDI_TRAY
mov note.uFlags, NIF_ICON + + NIF_MESSAGE NIF_TIP
mov note.uCallbackMessage, WM_SHELLNOTIFY
invocar LoadIcon, NULL, IDI_WINLOGO
mov note.hIcon, eax
invocar lstrcpy, addr note.szTip, addr AppName
invocar ShowWindow, hWnd, SW_HIDE
invocar Shell_NotifyIcon, NIM_ADD, nota addr
. Endif
```

Aprovechamos esta oportunidad para llenar la estructura NOTIFYICONDATA. IDI_TRAY es sólo una constante definida en el comienzo del código fuente. Se puede ajustar a cualquier valor que desee. No es importante, porque sólo tiene un icono de la bandeja. Pero si va a poner varios iconos en la bandeja de sistema, es necesario un ID único para cada icono de la bandeja. Se especifican todas las banderas en los países miembros uFlags porque especificar un icono (NIF_ICON), se especifica un mensaje personalizado (NIF_MESSAGE) y especificar el texto tooltip (NIF_TIP). WM_SHELLNOTIFY es sólo un mensaje personalizado definido como WM_USER 5. El valor real no es importante siempre que es único. Puedo usar el icono WinLogo como el icono de la bandeja aquí, pero usted puede utilizar cualquier icono en su programa. Sólo tienes que cargar desde el recurso con LoadIcon y poner el identificador devuelto en el miembro hIcon. Por último, llenamos el szTip con el texto que queremos que el shell va a mostrar cuando el ratón está sobre el icono.
Nos escondemos la ventana principal para dar la ilusión de "minimizar a la bandeja-icono de" apariencia.
A continuación llamamos Shell_NotifyIcon con el mensaje de NIM_ADD para añadir el icono a la bandeja del sistema.

Ahora la ventana principal está oculto y es el icono en la bandeja del sistema. Si usted mueve el puntero del ratón sobre él, usted verá una información sobre herramientas que muestra el texto que ponemos en miembro de szTip. A continuación, si hace doble clic en el icono, la ventana principal volverá a aparecer y el icono de la bandeja se ha ido.

```
. UMsg elseif == WM_SHELLNOTIFY
. Si wParam == IDI_TRAY
. Si lParam == WM_RBUTTONDOWN
```

```
invocar GetCursorPos, addr pt
invocar SetForegroundWindow, hWnd
invocar TrackPopupMenu, hPopupMenu, TPM_RIGHTALIGN, pt.x, pt.y, NULL, hWnd,
NULL
invocar PostMessage, hWnd, WM_NULL, 0,0
. Elseif lParam == WM_LBUTTONDBLCLK
invoca SendMessage, hWnd, WM_COMMAND, IDM_RESTORE, 0
. Endif
. Endif
```

Cuando se produce un evento de ratón sobre el icono de la bandeja, la ventana recibe el mensaje WM_SHELLNOTIFY que es el mensaje personalizado que se especifica en el miembro uCallbackMessage. Recordemos que al recibir este mensaje, wParam contiene el identificador del icono de la bandeja y lParam contiene el mensaje del ratón. En el código anterior, se comprueba primero si este mensaje llega desde el icono de la bandeja que está interesado pulg Si lo hace, compruebe el mensaje de ratón real. Puesto que sólo están interesados en clic con el botón derecho del ratón y haga doble clic con el botón izquierdo, que procesa los mensajes sólo WM_RBUTTONDOWN y WM_LBUTTONDBLCLK.

Si el mensaje del ratón es WM_RBUTTONDOWN, que llamamos GetCursorPos para obtener la coordenada de pantalla actual del cursor del ratón. Cuando la función retorna, la estructura POINT se llena con la coordenada pantalla del cursor del ratón. Por las coordenadas de la pantalla, me refiero a las coordenadas de la pantalla completa sin relación a ningún límite de la ventana. Por ejemplo, si la resolución de pantalla es 640 * 480, la esquina inferior derecha de la pantalla es x == y == 639 y 479. Si desea convertir la pantalla de coordinación para coordinar la ventana, utilice la función ScreenToClient.

Sin embargo, para nuestro propósito, queremos mostrar el menú emergente en la posición actual del cursor con el llamado de TrackPopupMenu y requiere de coordenadas de la pantalla, podemos utilizar las coordenadas ocupados por GetCursorPos directamente.

TrackPopupMenu tiene la siguiente sintaxis:

TrackPopupMenu hMenu PROTO: DWORD, uFlags: DWORD, x: DWORD, y: DWORD, nReserved: DWORD hWnd: DWORD, prcRect: DWORD

- hMenu es el identificador del menú emergente que se mostrará
- uFlags especifica las opciones de la función. Al igual que dónde colocar el menú en relación con las coordenadas especificadas más adelante y que botón del ratón se puede utilizar para rastrear el menú. En nuestro ejemplo, usamos TPM_RIGHTALIGN para colocar el menú que aparece a la izquierda de las coordenadas.
- xey especificar la ubicación del menú en coordenadas de pantalla.
- nReserved debe ser NULL
- hWnd es el handle de la ventana que recibirá los mensajes en el menú.
- prcRect es el rectángulo en la pantalla donde es posible hacer clic sin dejar de lado el menú. Normalmente ponemos NULL aquí, así que cuando el usuario hace clic en cualquier parte fuera del menú emergente, el menú se cierra.

Cuando el usuario hace doble clic en el icono de la bandeja, enviamos mensaje WM_COMMAND a nuestra propia ventana especificando IDM_RESTORE para emular el usuario selecciona del menú Restaurar en el menú emergente restaurando así la ventana principal y eliminar el icono de la bandeja del sistema. Con el fin de poder recibir el mensaje de doble clic, la ventana principal debe tener estilo CS_DBLCLKS.

```
invocar Shell_NotifyIcon, NIM_DELETE, nota addr
mov eax, wParam
. Si ax == IDM_RESTORE
invocar ShowWindow, hWnd, SW_RESTORE
. Más
invocar DestroyWindow, hWnd
. Endif
```

Cuando el usuario selecciona el punto de restauración del menú, se elimina el icono de la bandeja llamando Shell_NotifyIcon nuevo, esta vez se especifica NIM_DELETE como el mensaje. A continuación, restaurar la ventana principal a su estado original. Si el usuario selecciona el punto de salida del menú, también eliminar el icono de la bandeja y destruir a la ventana principal llamando DestroyWindow.

TUTORIAL 24: GANCHOS DE WINDOWS

Vamos a aprender acerca de Windows ganchos en este tutorial. Ganchos de Windows son muy poderosos. Con ellos, usted puede meter dentro de otros procesos y, a veces modificar sus comportamientos.

.

TEORÍA:

Ganchos de Windows puede ser considerada como una de las características más poderosas de Windows. Con ellos, puede interceptar los eventos que tendrán lugar, ya sea en su propio proceso o en procesos distintos. Por "enganche", le dice a Windows acerca de una función, la función de filtro también se llama procedimiento de enlace, que será llamado cada vez un evento que le interesa se produce. Hay dos tipos de ellos: ganchos locales y remotas.

- Locales ganchos eventos de captura que tendrán lugar en su propio proceso.
- Remoto ganchos eventos de captura que se producirán en otro proceso (es). Hay dos tipos de ganchos remotos
    - hilo de eventos específicos de las trampas que se producirán en un hilo específico en otro proceso. En definitiva, se quiere observar los acontecimientos en un hilo específico en un proceso específico.
    - todo el sistema de trampas de todos los eventos destinados a todos los temas en todos los procesos en el sistema.

Al instalar los ganchos, recuerde que afectan el rendimiento del sistema. Todo el sistema de ganchos son los más notorios. Como todos los eventos relacionados se dirigirán a través de su función de filtro, el sistema puede reducir la velocidad notablemente. Así que si usted utiliza un gancho de todo el sistema, que debe administrarse con prudencia y desenganche que tan pronto como usted no lo necesita. Además, tienen una mayor probabilidad de chocar los otros procesos, ya que puede interferir con otros procesos y si algo anda mal en su función de filtro, se puede tirar de los otros procesos hasta el olvido de la misma. Recuerde: El poder viene la responsabilidad.

Usted tiene que entender cómo funciona un gancho antes de que pueda utilizarlo de manera eficiente. Cuando se crea un gancho, Windows crea una estructura de datos en la memoria, que contiene información sobre el gancho, y se agrega a una lista enlazada de ganchos existentes. Gancho Nueva se añade delante de ganchos viejos. Cuando ocurre un evento, si instala un gancho local, la función de filtro en el proceso se llama así que es bastante sencillo. Pero si es un gancho remoto, el sistema debe inyectar el código para el procedimiento de gancho en el espacio de direcciones (s) del proceso de otro (s). Y el sistema puede hacer que sólo si la función se encuentra en un archivo DLL. Por lo tanto, si desea utilizar un gancho remoto, el procedimiento de enlace debe residir en una DLL. Hay dos excepciones a esta regla: Diario de grabación y reproducción de los ganchos de revistas. Los procedimientos de gancho para los dos ganchos deben residir en el hilo que instala los ganchos. La razón por la que debe ser así es que: tanto los ganchos acuerdo con la interceptación de bajo nivel de los eventos de entrada de hardware. Los eventos de entrada debe estar registrado / reproduzcan en el orden en que aparecen. Si el código de esos dos ganchos se encuentra en un archivo

DLL, los eventos de entrada pueden dispersarse entre los varios hilos y es imposible saber el orden de ellos. Así que la solución: el procedimiento de enlace de los dos ganchos deben estar en un solo hilo es decir, sólo el hilo que instala los ganchos.
Hay 14 tipos de anzuelos:

- **WH_CALLWNDPROC** llama cuando se llama a SendMessage
- **WH_CALLWNDPROCRET** llama cuando SendMessage devuelve
- **WH_GETMESSAGE** llamado cuando GetMessage o PeekMessage se llama
- **WH_KEYBOARD** llamado cuando GetMessage o PeekMessage recupera WM_KEYUP WM_KEYDOWN o de la cola de mensajes
- **WH_MOUSE** llamado cuando GetMessage o PeekMessage recupera un mensaje de ratón de la cola de mensajes
- **WH_HARDWARE** llamado cuando GetMessage o PeekMessage recupera un mensaje de hardware que no está relacionado con el teclado o el ratón.
- **WH_MSGFILTER** llama cuando un cuadro de diálogo, menú o barra de desplazamiento está a punto de procesar un mensaje. Este gancho es local. Es específicamente para aquellos objetos que tienen sus propios bucles de mensajes internos.
- **WH_SYSMSGFILTER** mismo WH_MSGFILTER, sino de todo el sistema
- **WH_JOURNALRECORD** llamado cuando Windows recupera mensaje de la cola de entrada de hardware
- **WH_JOURNALPLAYBACK** llamada cuando un evento se solicita a la cola del hardware del sistema de entrada.
- **WH_SHELL** llama cuando algo interesante sobre la cáscara se produce por ejemplo cuando la barra de tareas tiene que volver a dibujar el botón.
- **WH_CBT** utilizado específicamente para formación por ordenador (CBT).
- **WH_FOREGROUNDIDLE** utilizado internamente por Windows. Escasa utilización para aplicaciones generales
- **WH_DEBUG** utiliza para depurar el procedimiento de enganche

Ahora que sabemos algo de teoría, podemos pasar a la forma de instalar / desinstalar los ganchos.
Para instalar un gancho, se llama a SetWindowsHookEx que tiene la siguiente sintaxis:
SetWindowsHookEx proto HookType: DWORD, pHookProc: DWORD, hInstance: DWORD, ThreadID: DWORD

- HookType es uno de los valores antes mencionados, por ejemplo, **WH_MOUSE, WH_KEYBOARD**
- pHookProc es la dirección del procedimiento de enlace que se llamará para procesar los mensajes para el gancho especificado. Si el gancho es un mando a distancia, debe residir en una DLL. Si no, debe ser en su proceso.
- hInstance es el identificador de instancia de la DLL en el que el procedimiento de enlace reside. Si el gancho es local, este valor debe ser NULL
- ThreadID es el ID del hilo que desea instalar el gancho para espiar. Este parámetro es el que determina si un gancho es local o remoto. Si este parámetro es NULL, Windows interpreta el gancho como un gancho a distancia en todo el sistema que afecta a todas las discusiones en el sistema. Si se especifica el ID del hilo de un hilo en su propio proceso, este libro es local. Si se especifica el identificador del subproceso de otro proceso, el gancho es un hilo específico de un mando a distancia. Hay dos excepciones a esta regla: **WH_JOURNALRECORD** y **WH_JOURNALPLAYBACK** son siempre locales en todo el sistema ganchos que no están obligados a estar en un archivo DLL. Y **WH_SYSMSGFILTER** es siempre un gancho a distancia en todo el sistema. De hecho, es idéntico al gancho **WH_MSGFILTER** con ThreadID == 0.

Si la llamada tiene éxito, se devuelve el manejador del gancho en eax. Si no, se devuelve NULL. Debe guardar el asa con gancho para desenroscar la tarde.

Puede desinstalar un gancho llamando **UnhookWindowsHookEx** que acepta un solo parámetro, el mango del gancho que desee desinstalar. Si la llamada tiene éxito, devuelve un valor distinto de cero en eax. De lo contrario, devuelve NULL.

Ahora que sabes cómo instalar / desinstalar los ganchos, podemos examinar el procedimiento de enlace.

El procedimiento de enlace será llamado cada vez que un evento que está asociado con el tipo de gancho que ha instalado se produce. Por ejemplo, si instala el gancho **WH_MOUSE,** cuando se produce un evento de ratón, el procedimiento de enlace será llamado. Independientemente del tipo de gancho que ha instalado, el procedimiento de enlace siempre tiene el siguiente prototipo:

HookProc proto nCode: DWORD, wParam: DWORD, lParam: DWORD

- nCode especifica el código de gancho.
- wParam y lParam contienen información adicional sobre el evento

HookProc es en realidad un marcador de posición para el nombre de la función. Puede que sea lo que quiera, siempre y cuando se tiene el prototipo anterior. La interpretación de nCode, wParam y lParam depende del tipo de gancho de instalar. Así como el valor devuelto desde el procedimiento de enlace. Por ejemplo:

WH_CALLWNDPROC

- nCode puede ser sólo HC_ACTION que significa que hay un mensaje enviado a una ventana
- wParam contiene el mensaje que se envía, si no es cero
- lParam apunta a una estructura CWPSTRUCT
- Valor de retorno: no se utiliza, devolver cero

WH_MOUSE

- nCode puede ser HC_ACTION o HC_NOREMOVE
- wParam contiene el mensaje del ratón
- lParam apunta a una estructura MOUSEHOOKSTRUCT
- Valor de retorno: cero si el mensaje debe ser procesada. 1 si el mensaje debe ser desechada.

La conclusión es: usted debe consultar a su referencia de la api de win32 para obtener más información acerca de los significados de los parámetros y el valor de retorno del gancho que desea instalar.

Ahora hay una trampa poco sobre el procedimiento de enlace. Recuerde que los ganchos se encadenan en una lista enlazada con el gancho más recientemente instalado a la cabeza de la lista. Cuando ocurre un evento, Windows llamará sólo el primer gancho de la cadena. Es la responsabilidad de su procedimiento de gancho para llamar al siguiente gancho en la cadena. Usted puede optar por no llamar al siguiente gancho, pero es mejor que sepas lo que estás haciendo. La mayor parte del tiempo, es una buena práctica para llamar al procedimiento siguiente para que otros ganchos pueden tener una oportunidad en el evento. Usted puede llamar al siguiente gancho llamando **CallNextHookEx** que tiene el siguiente prototipo:

CallNextHookEx proto hHook: DWORD, nCode: DWORD, wParam: DWORD, lParam: DWORD

- hHook es el mango de su propio gancho. La función utiliza este identificador para recorrer la lista enlazada y busque el procedimiento de enlace se debe llamar al siguiente.

- nCode, wParam y lParam puedes pasar esos tres valores que recibe de Windows para CallNextHookEx.

Una nota importante sobre los ganchos a distancia: el procedimiento de gancho debe residir en una DLL que se asignan a otros procesos. Cuando Windows se asigna el archivo DLL en otros procesos, no va a asignar la sección de datos (s) en los otros procesos. En resumen, todos los procesos de compartir una sola copia del código, pero ellos tendrán su propia copia privada de la sección del archivo DLL de datos! Esto puede ser una gran sorpresa a los incautos. Usted puede pensar que cuando se almacena un valor en una variable en la sección de datos de un archivo DLL, ese valor será compartido entre todos los procesos que cargan el archivo DLL en su espacio de direcciones del proceso. Simplemente no es cierto. En situación normal, este comportamiento es deseable, ya que proporciona la ilusión de que cada proceso tiene su propia copia de la DLL. Pero no cuando Windows gancho es que se trate. Queremos que el archivo DLL que ser idénticos en todos los procesos, incluidos los datos. La solución: hay que marcar la sección de datos compartido. Usted puede hacer esto mediante la especificación de la sección (s) atributo en el interruptor de enlace. Para MASM, es necesario utilizar este modificador:
/ SECCIÓN: <nombre <section, S

El nombre de la sección de datos es inicializada. De datos y los datos no inicializada es. Bss. Por ejemplo, si quieres montar un archivo DLL que contiene un procedimiento de enlace y desea que la sección de datos sin inicializar a compartir amoung procesos, debe utilizar la siguiente línea:

enlace / sección:. bss, S / DLL / SUBSYSTEM: WINDOWS .....

Atributo de S marca la sección como compartida.

Ejemplo:

Hay dos módulos: uno es el programa principal, que va a hacer la parte de interfaz gráfica de usuario y la otra es la DLL que instalar / desinstalar el gancho.

; ---------- Esta es la fuente el código del programa principal ----------

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
incluyen mousehook.inc
mousehook.lib includelib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

wsprintfA proto C: DWORD,: DWORD,: vararg
wsprintf <wsprintfA> TEXTEQU

Const.
IDD_MAINDLG equ 101
IDC_CLASSNAME equ 1000
IDC_HANDLE equ 1001
IDC_WNDPROC equ 1002
IDC_HOOK equ 1004
```

```
IDC_EXIT equ 1005
WM_MOUSEHOOK equ WM_USER 6

DlgFunc PROTO: DWORD,: DWORD,: DWORD,: DWORD

. Datos
HookFlag dd FALSO
HookText db "y el gancho", 0
UnhookText db "y Desenganche", 0
plantilla de DB "% lx", 0

. Datos?
hInstance dd?
hHook dd?
. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar DialogBoxParam, hInstance, IDD_MAINDLG, NULL, addr DlgFunc, NULL
invocar ExitProcess, NULL

DlgFunc proc hDlg: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
LOCAL hLib: DWORD
Búfer local [128]: bytes
LOCAL buffer1 [128]: bytes
LOCAL rect: RECT
. Si uMsg == WM_CLOSE
. Si HookFlag == TRUE
invocar UninstallHook
. Endif
invocar EndDialog, hDlg, NULL
. UMsg elseif == WM_INITDIALOG
invocar GetWindowRect, hDlg, addr rect
invocar SetWindowPos, hDlg, HWND_TOPMOST, rect.left, rect.top, rect.right,
rect.bottom, SWP_SHOWWINDOW
. Elseif uMsg == WM_MOUSEHOOK
invocar GetDlgItemText, hDlg, IDC_HANDLE, addr buffer1, 128
invocar wsprintf, addr buffer, addr plantilla, wParam
invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_HANDLE, tampón addr
. Endif
invocar GetDlgItemText, hDlg, IDC_CLASSNAME, addr buffer1, 128
invocar GetClassName, wParam, addr buffer, 128
invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_CLASSNAME, tampón addr
. Endif
invocar GetDlgItemText, hDlg, IDC_WNDPROC, addr buffer1, 128
invocar GetClassLong, wParam, GCL_WNDPROC
invocar wsprintf, addr buffer, addr plantilla, eax
invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_WNDPROC, tampón addr
. Endif
. Elseif uMsg == WM_COMMAND
. Si lParam! = 0
mov eax, wParam
mov edx, eax
shr edx, 16
```

```
. Si dx == BN_CLICKED
. Si ax == IDC_EXIT
invoca SendMessage, hDlg, WM_CLOSE, 0,0
. Más
. Si HookFlag == FALSO
invocar InstallHook, hDlg
. Si eax! = NULL
mov HookFlag, TRUE
invocar SetDlgItemText, hDlg, IDC_HOOK, UnhookText addr
. Endif
. Más
invocar UninstallHook
invocar SetDlgItemText, hDlg, IDC_HOOK, HookText addr
mov HookFlag, FALSO
invocar SetDlgItemText, hDlg, IDC_CLASSNAME, NULL
invocar SetDlgItemText, hDlg, IDC_HANDLE, NULL
invocar SetDlgItemText, hDlg, IDC_WNDPROC, NULL
. Endif
. Endif
. Endif
. Endif
. Más
mov eax, FALSO
ret
. Endif
mov eax, TRUE
ret
DlgFunc endp

poner fin a empezar a

; ------------------------------------------------ ---- Este es el código fuente de la DLL ----------------------
---------------
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ kernel32.lib
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ user32.lib

Const.
WM_MOUSEHOOK equ WM_USER 6

. Datos
hInstance dd 0

. Datos?
hHook dd?
hWnd dd?

. Código
DllEntry proc hInst: HINSTANCE, la razón: DWORD, reserved1: DWORD
. Si la razón == DLL_PROCESS_ATTACH
impulsar hInst
pop hInstance
. Endif
mov eax, TRUE
```

```
ret
DllEntry Endp

MouseProc proc nCode: DWORD, wParam: DWORD, lParam: DWORD
invocar CallNextHookEx, hHook, nCode, wParam, lParam
mov edx, lParam
asumir edx: PTR MOUSEHOOKSTRUCT
invocar WindowFromPoint, [edx]. pt.x, [edx]. pt.y
invocar PostMessage, hWnd, WM_MOUSEHOOK, eax, 0
asumir edx: nada
xor eax, eax
ret
MouseProc endp

InstallHook proc hwnd: DWORD
impulsar hwnd
pop hWnd
invocar SetWindowsHookEx, WH_MOUSE, MouseProc addr, hInstance, NULL
mov hHook, eax
ret
InstallHook endp

UninstallHook proc
invocar UnhookWindowsHookEx, hHook
ret
UninstallHook endp

Fin DllEntry

; ---------- Este es el makefile de la DLL ----------

NAME = mousehook
. $ (NOMBRE) dll: $ (NOMBRE) obj.
Enlace / SECCIÓN:. Bss, S / DLL / DEF:. $ (NOMBRE) def / SUBSYSTEM: WINDOWS /
LIBPATH: c:. \ MASM \ lib $ (NOMBRE) obj
. $ (NOMBRE) obj: $ (NOMBRE) asm.
ml / c / coff / Cp $ (NAME). asm
```

Análisis:

El ejemplo mostrará un cuadro de diálogo con tres controles de edición que será llenado con el nombre de la clase, la manija de la ventana y la dirección del procedimiento de ventana asociado a la ventana bajo el cursor del ratón. Hay dos botones, gancho y Salir. Cuando se presiona el botón Hook, el programa engancha la entrada del ratón y el texto del botón cambia para desenganchar. Al mover el cursor del ratón sobre una ventana, la información sobre esa ventana se mostrará en la ventana principal del ejemplo. Cuando se presiona el botón Desenganche, el programa elimina el gancho del ratón.

El programa principal utiliza un cuadro de diálogo como la ventana principal. Se define un mensaje personalizado WM_MOUSEHOOK, que se utilizará entre el programa principal y el archivo DLL de gancho. Cuando el programa principal recibe este mensaje, wParam contiene el manejador de la ventana que el cursor del mouse está encendido. Por supuesto, esto es una disposición arbitraria. Decido enviar el mango en wParam en aras de la simplicidad. Usted puede elegir su propio método de comunicación entre el programa principal y el archivo DLL de gancho.

```
. Si HookFlag == FALSO
invocar InstallHook, hDlg
. Si eax! = NULL
mov HookFlag, TRUE
invocar SetDlgItemText, hDlg, IDC_HOOK, UnhookText addr
. Endif
```

El programa mantiene una bandera, HookFlag, para vigilar el estado del gancho. Es falso cuando el gancho no está instalado y TRUE si el gancho se instala.

Cuando el usuario presiona el botón Hook, el programa comprueba si el gancho ya está instalado. Si no es así, llame a la función InstallHook en la DLL de gancho para instalarlo. Tenga en cuenta que pasamos la manija de la ventana principal como el parámetro de la función para la DLL de gancho puede enviar los mensajes WM_MOUSEHOOK a la ventana de la derecha, es decir la nuestra.

Cuando se carga el programa, el archivo DLL se carga el gancho también. En realidad, los archivos DLL se cargan inmediatamente después de que el programa es en la memoria. La función de punto de entrada DLL se llama antes de la primera instrucción en el programa principal es ejecutar aun. Así que cuando el programa principal ejecuta el archivo DLL (s) es / son inicializado. Nosotros ponemos el siguiente código en la función DLL punto de entrada de la DLL de gancho:

```
. Si la razón == DLL_PROCESS_ATTACH
impulsar hInst
pop hInstance
. Endif
```

El código sólo guarda el identificador de instancia de la DLL de gancho de sí a una variable global llamada hInstance para su uso dentro de la función InstallHook. Dado que la función del punto de entrada DLL se llama antes de otras funciones en el archivo DLL se llaman, hInstance es siempre válido. Ponemos en hInstance. Sección de datos de manera que este valor se mantiene en función de cada proceso. Dado que cuando se pasa el cursor del ratón sobre una ventana, la DLL de gancho se asigna en el proceso. Imagine que ya existe un archivo DLL que ocupa la dirección de carga previsto de la DLL de gancho, la DLL de gancho se reasigna a otra dirección. El valor de hInstance será actualizado a las de la dirección de carga nueva. Cuando el usuario presiona el botón Desenganche y luego el botón Hook, SetWindowsHookEx será llamada de nuevo. Sin embargo, en esta ocasión, se utilizará la dirección de carga nueva, como el identificador de instancia que será malo porque en el proceso de ejemplo, la dirección de la DLL de gancho de carga no se ha modificado. El gancho será un local donde se puede conectar sólo los eventos de ratón que se producen en su propia ventana. Apenas deseable.

```
InstallHook proc hwnd: DWORD
impulsar hwnd
pop hWnd
invocar SetWindowsHookEx, WH_MOUSE, MouseProc addr, hInstance, NULL
mov hHook, eax
ret
InstallHook endp
```

La función InstallHook en sí es muy simple. Guarda el identificador de ventana pasado como parámetro a una variable global llamada hWnd para su uso futuro. A continuación, llama SetWindowsHookEx para instalar un gancho de ratón. El valor de retorno de SetWindowsHookEx se almacena en una variable llamada hHook mundial para su uso con UnhookWindowsHookEx.

Después de SetWindowsHookEx se llama, el gancho del ratón es funcional. Cada vez que un evento de ratón se produce en el sistema, MouseProc (el procedimiento de enlace) se llama.

```
MouseProc proc nCode: DWORD, wParam: DWORD, lParam: DWORD
invocar CallNextHookEx, hHook, nCode, wParam, lParam
mov edx, lParam
```

```
asumir edx: PTR MOUSEHOOKSTRUCT
invocar WindowFromPoint, [edx]. pt.x, [edx]. pt.y
invocar PostMessage, hWnd, WM_MOUSEHOOK, eax, 0
asumir edx: nada
xor eax, eax
ret
MouseProc endp
```

Lo primero que hace es llamar a CallNextHookEx para dar a los ganchos de otros la oportunidad de procesar el evento de ratón. Después de eso, llama a la función WindowFromPoint para recuperar el identificador de la ventana en la coordenada de pantalla especificada. Tenga en cuenta que se utiliza la estructura de punto de la estructura MOUSEHOOKSTRUCT a la que apunta lParam como la coordenada actual del ratón. Después de que enviamos el identificador de ventana para el programa principal a través de PostMessage con el mensaje de WM_MOUSEHOOK. Una cosa que debes recordar es que: no se debe utilizar SendMessage dentro del procedimiento de enlace, que puede causar estancamiento mensaje. PostMessage se recomienda. La estructura MOUSEHOOKSTRUCT se definen a continuación:

```
MOUSEHOOKSTRUCT STRUCT DWORD
pt PUNTO <>
hwnd DWORD?
wHitTestCode DWORD?
dwExtraInfo DWORD?
MOUSEHOOKSTRUCT EXTREMOS
```

- PT es la coordenada de pantalla actual del cursor del ratón
- hwnd es el handle de la ventana que se recibe el mensaje de ratón. Por lo general es la ventana bajo el cursor del ratón, pero no siempre. Si una ventana se llama SetCapture, la entrada del ratón será redirigido a la ventana en su lugar. Por esta razón, yo no uso el miembro hwnd de esta estructura, pero decide llamar a WindowFromPoint lugar.
- wHitTestCode especifica el valor del éxito de la prueba. El valor hit-test da más información acerca de la posición actual del cursor. Se especifica en qué parte de la ventana del cursor del ratón. Para una lista completa, consulte su referencia de la api de win32 en WM_NCHITTEST mensaje.
- dwExtraInfo contiene la información adicional asociada con el mensaje. Normalmente, este valor se establece mediante una llamada mouse_event y recuperados llamando GetMessageExtraInfo.

Cuando la ventana principal recibe el mensaje WM_MOUSEHOOK, utiliza el identificador de ventana en wParam para recuperar la información sobre la ventana.

```
. Elseif uMsg == WM_MOUSEHOOK
invocar GetDlgItemText, hDlg, IDC_HANDLE, addr buffer1, 128
invocar wsprintf, addr buffer, addr plantilla, wParam
invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_HANDLE, tampón addr
. Endif
invocar GetDlgItemText, hDlg, IDC_CLASSNAME, addr buffer1, 128
invocar GetClassName, wParam, addr buffer, 128
invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_CLASSNAME, tampón addr
. Endif
invocar GetDlgItemText, hDlg, IDC_WNDPROC, addr buffer1, 128
invocar GetClassLong, wParam, GCL_WNDPROC
invocar wsprintf, addr buffer, addr plantilla, eax
```

invocar lstrcmpi, addr buffer, addr buffer1
. Si eax! = 0
invocar SetDlgItemText, hDlg, IDC_WNDPROC, tampón addr
. Endif

Para evitar parpadeos, comprobamos que el texto ya en los controles de edición y el texto que será puesto en ellos si son idénticos. Si es así, que les pase.
Nos recuperar el nombre de la clase llamando GetClassName, la dirección del procedimiento de ventana llamando GetClassLong con GCL_WNDPROC y luego darles formato en cadenas y ponerlas en los correspondientes controles de edición.

invocar UninstallHook
invocar SetDlgItemText, hDlg, IDC_HOOK, HookText addr
mov HookFlag, FALSO
invocar SetDlgItemText, hDlg, IDC_CLASSNAME, NULL
invocar SetDlgItemText, hDlg, IDC_HANDLE, NULL
invocar SetDlgItemText, hDlg, IDC_WNDPROC, NULL

Cuando el usuario presiona el botón Desenganche, el programa llama a la función UninstallHook en la DLL de gancho. UninstallHook sólo llama a UnhookWindowsHookEx. Después de eso, cambia el texto del botón de nuevo a "Hook", HookFlag a FALSE y borra el contenido de los controles de edición.
Tenga en cuenta el cambio del vinculador en el makefile.

Enlace / SECCIÓN:. Bss, S DLL / / DEF: $ (NOMBRE) def / SUBSYSTEM: Windows

En él se especifica. La sección bss como una sección compartida para que todos los procesos comparten la misma sección de inicializar los datos de la DLL de gancho. Sin este cambio, el archivo DLL de gancho no funcionará correctamente.

TUTORIAL 25: SIMPLE MAPA DE BITS

En este tutorial, vamos a aprender a usar mapas de bits en nuestro programa. Para ser exactos, vamos a aprender cómo se muestra un mapa de bits en el área de cliente de nuestra ventana.

TEORÍA

Los mapas de bits se pueden considerar como las imágenes almacenadas en el ordenador. Hay muchos formatos de imagen utilizados con ordenadores, pero sólo para Windows de forma nativa archivos de mapa de bits compatible con Windows de gráficos (. Bmp). Los mapas de bits que me referiré en este tutorial son gráficos de Windows los archivos de mapa de bits. La manera más fácil de usar un mapa de bits es para utilizarlo como un recurso. Hay dos maneras de hacerlo. Puede incluir el mapa de bits en el archivo de definición de recursos (RC). De la siguiente manera:
Define IDB_MYBITMAP 100
BITMAP IDB_MYBITMAP "c: \ proyecto \ example.bmp"
Este método utiliza una constante para representar el mapa de bits. La primera línea crea un IDB_MYBITMAP constante con nombre que tiene el valor de 100. Vamos a utilizar esta etiqueta para referirse al mapa de bits en el programa. La siguiente línea declara un recurso de mapa de bits. Le dice al compilador de recursos donde encontrar la verdadera bmp.
El otro método usa un nombre para representar el mapa de bits como sigue:
BITMAP myBitmap "c: \ proyecto \ example.bmp"
Este método requiere que usted se refiere al mapa de bits en su programa por la cadena "myBitmap" en lugar de un valor.
Cualquiera de estos métodos funciona bien siempre y cuando usted sabe que el método que

está utilizando.
Ahora que ponemos el mapa de bits en el archivo de recursos, podemos seguir con los pasos de su exposición en el área de cliente de nuestra ventana.

1. llame LoadBitmap para obtener el identificador de mapa de bits. LoadBitmap tiene la siguiente definición:

LoadBitmap proto hInstance: HINSTANCE, lpBitmapName: LPSTR

Esta función devuelve un identificador de mapa de bits. hInstance es el identificador de instancia de nuestro programa. lpBitmapName es un puntero a la cadena que es el nombre del mapa de bits (en caso que usted utilice el segundo método para referirse al mapa de bits). Si usted usa una constante para referirse al mapa de bits (como IDB_MYBITMAP), usted puede poner su valor aquí. (En el ejemplo anterior sería 100). Un pequeño ejemplo es con el fin de:

Primer Método:

```
0.386
. Modelo plano, stdcall
.....
Const.
IDB_MYBITMAP equ 100
.....
. Datos?
hInstance dd?
.....
. Código
.....
invocar GetModuleHandle, NULL
mov hInstance, eax
.....
invocar LoadBitmap, hInstance, IDB_MYBITMAP
.....
```

Segundo método:

```
0.386
. Modelo plano, stdcall
.....
. Datos
BitmapName db "myBitmap", 0
.....
. Datos?
hInstance dd?
.....
. Código
.....
invocar GetModuleHandle, NULL
mov hInstance, eax
.....
invocar LoadBitmap, hInstance, addr BitmapName
.....
```

1. Obtener un manejador al contexto de dispositivo (DC). Usted puede obtener este identificador llamando a BeginPaint en respuesta al mensaje WM_PAINT o llamando a GetDC en cualquier lugar.
2. Crear un contexto de dispositivo de memoria que tiene el mismo atributo que el contexto de dispositivo que acaba de obtener. La idea aquí es crear un tipo de superficie "oculto" de dibujo que podemos dibujar el mapa de bits en. Cuando hayamos terminado con la operación, sólo tienes que copiar el contenido de la superficie de dibujo oculto en el contexto real del dispositivo en una llamada a la función. Es un ejemplo de la técnica de doble búfer utilizado para la visualización rápida de imágenes en la pantalla. Puede crear este "oculto" superficie de dibujo llamando CreateCompatibleDC.

```
CreateCompatibleDC proto hdc: HDC
```

Si esta función tiene éxito, regresa el manejador del contexto de dispositivo de memoria en eax. hdc es el manejador del contexto de dispositivo que desea que el DC de memoria que sea compatible con.

1. Ahora que usted tiene una superficie de dibujo oculto, se puede dibujar en él mediante la selección de mapa de bits en el mismo. Esto se hace llamando a SelectObject con el mango a la DC de memoria como el primer parámetro y el mango de mapa de bits como el segundo parámetro. SelectObject tiene la siguiente definición:

```
SelectObject proto hdc: HDC, hGdiObject: DWORD
```

1. El mapa de bits se dibuja en el contexto de dispositivo de memoria ahora. Todo lo que necesitamos hacer aquí es para copiarlo en el dispositivo de visualización real, es decir, el contexto de dispositivo real. Hay varias funciones que pueden realizar esta operación, tales como BitBlt y StretchBlt. BitBlt simplemente copia el contenido de un DC a otro por lo que es rápido, mientras que StretchBlt puede estirar o comprimir el mapa de bits para adaptarse a la zona de salida. Vamos a utilizar BitBlt aquí para simplificar. BitBlt tiene la siguiente definición:

```
BitBlt proto hdcDest: DWORD, nxDest: DWORD, nyDest: DWORD, nWidth:
DWORD, nHeight: DWORD, hdcSrc: DWORD, nxSrc: DWORD, nySrc:
DWORD, dwRop: DWORD
```

hdcDest es el identificador del contexto de dispositivo que sirve como el destino de la operación de transferencia de mapa de bits
nxDest nyDest, son las coordenadas de la esquina superior izquierda del área de salida
nWidth, nHeight son la anchura y la altura del área de salida
hdcSrc es el identificador del contexto de dispositivo que sirve como la fuente de operación de transferencia de mapa de bits
nxSrc, nySrc son la coordenada de la esquina superior izquierda del rectángulo fuente.
dwRop es el código de trama de la operación (de ahí el acrónimo ROP), que regula la forma de combinar los datos de color del mapa de bits de los datos de color existentes en el área de salida para lograr el resultado final. La mayoría de las veces, sólo desea sobrescribir los datos de color existentes con el nuevo.

1. Cuando haya terminado con el mapa de bits, elimínelo con DeleteObject llamada a la API.

¡Eso es! En resumen, es necesario poner el mapa de bits en el scipt recursos. A continuación, se carga desde el recurso con LoadBitmap. Vas a obtener el identificador de mapa de bits. A continuación, obtener el identificador del contexto de dispositivo de la zona

que quieres pintar el mapa de bits en. A continuación, se crea un contexto de dispositivo de memoria que sea compatible con el contexto de dispositivo que acaba de obtener. Seleccione el mapa de bits en la memoria DC copie el contenido de la memoria DC a la toma DC real.

EJEMPLO DE CÓDIGO:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ gdi32.inc
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ gdi32.lib

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD
IDB_MAIN un equ

. Datos
ClassName db "SimpleWin32ASMBitmapClass", 0
AppName db "Ejemplo de mapa de bits Win32ASM simple", 0

. Datos?
HINSTANCE hInstance?
LPSTR de línea de comandos?
hBitmap dd?

. Código
empezar:
invocar GetModuleHandle, NULL
mov hInstance, eax
invocar GetCommandLine
CommandLine mov, eax
WinMain invoca, hInstance, NULL, de línea de comandos, SW_SHOWDEFAULT
invocar ExitProcess, eax

WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR,
CmdShow: DWORD
LOCAL wc: WNDCLASSEX
Msg LOCAL: MSG
LOCAL hwnd: HWND
mov wc.cbSize, sizeof WNDCLASSEX
mov wc.style, CS_HREDRAW o CS_VREDRAW
mov wc.lpfnWndProc, OFFSET WndProc
mov wc.cbClsExtra, NULL
mov wc.cbWndExtra, NULL
impulsar hInstance
pop wc.hInstance
mov wc.hbrBackground, COLOR_WINDOW un
mov wc.lpszMenuName, NULL
wc.lpszClassName mov, ClassName OFFSET
invocar LoadIcon, NULL, IDI_APPLICATION
mov wc.hIcon, eax
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
```

```
INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
hInst, NULL
mov hwnd, eax
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. Mientras VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. Descanso. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. Endw
mov eax, msg.wParam
ret
WinMain endp

WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
LOCAL ps: PAINTSTRUCT
LOCAL hdc: HDC
LOCAL hMemDC: HDC
LOCAL rect: RECT
. Si uMsg == WM_CREATE
invocar LoadBitmap, hInstance, IDB_MAIN
hBitmap mov, eax
. UMsg elseif == WM_PAINT
invocar BeginPaint, hWnd, addr ps
mov hdc, eax
invocar CreateCompatibleDC, hdc
mov hMemDC, eax
invocar SelectObject, hMemDC, hBitmap
invocar GetClientRect, hWnd, addr rect
invocar BitBlt, hdc, 0,0, rect.right, rect.bottom, hMemDC, 0,0,
SRCCOPY
invocar DeleteDC, hMemDC
invocar EndPaint, hWnd, addr ps
. Elseif uMsg == WM_DESTROY
invocar DeleteObject, hBitmap
invocar PostQuitMessage, NULL
. MÁS
invocar DefWindowProc, hWnd, uMsg, wParam, lParam
ret
. ENDIF
xor eax, eax
ret
WndProc ENDP
poner fin a empezar a

; ----------------------------------------------- ------------------
--
; La secuencia de comandos de recursos
; ----------------------------------------------- ------------------
--
# Define IDB_MAIN 1
BITMAP IDB_MAIN "tweety78.bmp"
```

ANÁLISIS:

No hay mucho para analizar en este tutorial ;)

```
# Define IDB_MAIN 1
BITMAP IDB_MAIN "tweety78.bmp"
```

Definir una constante llamada IDB_MAIN, asignar una como su valor. Y luego usar esa constante como el identificador de recurso de mapa de bits. El archivo de mapa de bits que se incluirán en el recurso es "tweety78.bmp" que reside en la misma carpeta que la secuencia de comandos de recursos.

```
. Si uMsg == WM_CREATE
invocar LoadBitmap, hInstance, IDB_MAIN
hBitmap mov, eax
```

En respuesta a WM_CREATE, llamamos LoadBitmap para cargar el mapa de bits de los recursos, pasando el identificador del mapa de bits de recursos como el segundo parámetro de la API. Se tiene la manija para el mapa de bits, cuando la función devuelve. Ahora que el mapa de bits se carga, se puede pintar en el área de cliente de la ventana principal.

```
. UMsg elseif == WM_PAINT
invocar BeginPaint, hWnd, addr ps
mov hdc, eax
invocar CreateCompatibleDC, hdc
mov hMemDC, eax
invocar SelectObject, hMemDC, hBitmap
invocar GetClientRect, hWnd, addr rect
invocar BitBlt, hdc, 0,0, rect.right, rect.bottom, hMemDC, 0,0,
SRCCOPY
invocar DeleteDC, hMemDC
invocar EndPaint, hWnd, addr ps
```

Hemos escogido para pintar el mapa de bits en respuesta al mensaje WM_PAINT. En primer lugar, llamar a BeginPaint para obtener el identificador del contexto de dispositivo. Luego creamos un DC de memoria compatible con CreateCompatibleDC. A continuación, seleccione el mapa de bits en la memoria DC con SelectObject. Determinar la dimensión del área de cliente con GetClientRect. Ahora podemos ver el mapa de bits en el área de cliente llamando a BitBlt que copia el mapa de bits de la memoria DC a la toma DC real. Cuando la pintura se hace, no tenemos más necesidad de la DC de memoria, así que lo elimine con DeleteDC. Fin de sesión de pintura con EndPaint.

```
. Elseif uMsg == WM_DESTROY
invocar DeleteObject, hBitmap
invocar PostQuitMessage, NULL
```

Cuando no tenemos el mapa de bits más, lo borramos con DeleteObject

---

TUTORIAL 28: DEPURACIÓN DE LA API WIN32 PARTE 1

En este tutorial, usted aprenderá lo que ofrece a los desarrolladores Win32 con respecto a las primitivas de depuración. Usted sabrá cómo depurar un proceso cuando haya terminado con este tutorial.

.

TEORÍA:

Win32 tiene varias APIs que permiten a los programadores a utilizar algunos de los poderes de un depurador. Se les llama API de Win32 Debug o primitivas. Con ellos, usted puede:

- Carga de un programa o adherirse a un programa en ejecución para la depuración
- Obtener información de bajo nivel sobre el programa que se está depurando, como la ID del proceso, la dirección del punto de entrada, la base de la imagen y así sucesivamente.
- Recibir notificación de eventos relacionados con la depuración, como cuando un proceso / hilo se inicia y salidas, archivos DLL se carga / descarga, etc
- Modificar el proceso / hilo que se está depurando

En resumen, usted puede codificar un depurador sencillo con las API. Dado que este tema es muy amplio, lo divide en varias partes manejables: este tutorial es la primera parte. Voy a explicar los conceptos básicos y el marco general para el uso de las API de Win32 Debug en este tutorial.
Los pasos en el uso de las API de Win32 Debug son los siguientes:

1. **Crear un proceso o adjuntar el programa en un proceso en ejecución.** Este es el primer paso en el uso de las API de Win32 Debug. Desde su programa actuará como un depurador, se necesita un programa para depurar. El programa que se depura se llama depuración. Puede adquirir una depuración de dos maneras:
    - Puede crear el proceso de depuración a ti mismo con **CreateProcess.** Con el fin de crear un proceso para la depuración, debe especificar la bandera **DEBUG_PROCESS.** Esta bandera le dice a Windows que queremos depurar el proceso. Windows enviará notificaciones de importantes acontecimientos relacionados con la depuración (depuración de eventos) que se producen en la depuración de su programa. El proceso de depuración será inmediatamente suspendido hasta que el programa está listo. Si la depuración también crea procesos hijos, Windows enviará también eventos de depuración que se producen en todos los procesos hijo a su programa también. Este comportamiento es generalmente indeseable. Puede desactivar este comportamiento especificando la bandera **DEBUG_ONLY_THIS_PROCESS** en combinación de la bandera **DEBUG_PROCESS.**
    - Puede conectar el programa a un proceso que se ejecuta con **DebugActiveProcess.**
2. **Espere a que la depuración de los acontecimientos.** Después de su programa adquirió un código depurado, el hilo principal de la depuración se suspende, y seguirá suspendido hasta que el programa llama a **WaitForDebugEvent.** Esta función se activa al igual que otras funciones, es decir, WaitForXXX. bloquea el subproceso de llamada hasta que el evento esperado-de se produce. En este caso, espera a que eventos de depuración que se enviarán por Windows. Veamos su definición:

    WaitForDebugEvent proto lpDebugEvent: DWORD, dwMilliseconds: DWORD

    lpDebugEvent es la dirección de una estructura **DEBUG_EVENT** que será llenado con la información sobre el evento de depuración que se produce dentro de la depuración.

    dwMilliseconds es la longitud de tiempo en milisegundos esta función se esperan para el evento de depuración que se produzca. Si transcurre este plazo y no hay evento de depuración ocurre, **WaitForDebugEvent** vuelve a la persona que llama. Por otro lado, si se especifica **constante** infinita en este argumento, la función no volverá hasta que se produce un evento de depuración.

    Ahora vamos a examinar la estructura DEBUG_EVENT con más detalle.

    DEBUG_EVENT STRUCT
    dwDebugEventCode dd?
    dwProcessId dd?
    dwThreadId dd?
    u DEBUGSTRUCT <>
    DEBUG_EVENT TERMINA

dwDebugEventCode contiene el valor que especifica el tipo de evento de depuración se produce. En pocas palabras, no puede haber muchos tipos de eventos, el programa necesita para comprobar el valor en este campo para que sepa qué tipo de evento se produce y responde apropiadamente. Los valores posibles son:

Valor	Significados
CREATE_PROCESS_DEBUG_EVENT	Se crea un proceso. Este evento será enviado cuando el proceso de depuración que se acaba de crear (y aún no en funcionamiento) o cuando el programa sólo se une a un proceso que se ejecuta con **DebugActiveProcess.** Este es el primer evento de su programa va a recibir.
EXIT_PROCESS_DEBUG_EVENT	Un proceso de las salidas.
CREATE_THEAD_DEBUG_EVENT	Un nuevo hilo se crea en el proceso de depuración o cuando su primer programa se une a un proceso en ejecución. Tenga en cuenta que usted no recibirá esta notificación cuando el hilo principal de la depuración se crea.
EXIT_THREAD_DEBUG_EVENT	Un hilo en el proceso sale depurado. Su programa no recibirán este evento por el hilo principal. En resumen, usted puede pensar en el hilo principal de la depuración como el equivalente del proceso de depuración en sí. Por lo tanto, cuando el programa ve **CREATE_PROCESS_DEBUG_EVENT,** en realidad es el **CREATE_THREAD_DEBUG_EVENT** para el hilo principal.
LOAD_DLL_DEBUG_EVENT	La depuración se carga un archivo DLL. Usted recibirá este evento cuando el cargador PE primero resuelve los enlaces a los archivos DLL (se llama a **CreateProcess** para cargar el depurado) y cuando la depuración llama a LoadLibrary.
UNLOAD_DLL_DEBUG_EVENT	Un DLL se descarga en el proceso de depuración.
EXCEPTION_DEBUG_EVENT	Se produce una excepción en el proceso de depuración. **Importante:** Este evento se producirá una vez justo antes de la depuración comienza a ejecutar su primera instrucción. La excepción es en realidad una interrupción de depuración (int 3h). Cuando desee reanudar la depuración, llame **a ContinueDebugEvent** con la bandera **DBG_CONTINUE.** No utilizar el indicador de **DBG_EXCEPTION_NOT_HANDLED** de lo contrario el código depurado se negará a ejecutarse en NT (en Windows 98, que funciona bien).
OUTPUT_DEBUG_STRING_EVENT	Este evento se genera cuando la depuración llama a la función **DebugOutputString** para enviar una cadena de mensaje a su programa.
RIP_EVENT	Error del sistema de depuración se produce

dwProcessId y **dwThreadId** son el proceso y los identificadores de hilo del proceso que se produce el evento de depuración. Puede utilizar estos valores como identificadores del proceso / hilo usted está interesado pulg Recuerde que si usted utiliza **CreateProcess** para cargar el código depurado, usted también consigue el proceso y los identificadores de hilo de la depuración en la estructura **PROCESS_INFO.** Puede utilizar estos valores para diferenciar entre los eventos de

depuración que ocurren en la depuración y sus procesos secundarios (en caso de que no se ha especificado la bandera **DEBUG_ONLY_THIS_PROCESS).**

u es una unión que contiene más información sobre el evento de depuración. Puede ser una de las siguientes estructuras, dependiendo del valor de **dwDebugEventCode** anteriormente.

valor en dwDebugEventCode	Interpretación de u
CREATE_PROCESS_DEBUG_EVENT	Una estructura **CREATE_PROCESS_DEBUG_INFO** llamado **CreateProcessInfo**
EXIT_PROCESS_DEBUG_EVENT	Un **EXIT_PROCESS_DEBU G_INFO estructura llamada** ExitProcess
CREATE_THREAD_DEBUG_EVENT	Una estructura **CREATE_THREAD_DEBUG_INFO** llamado **CreateThread**
EXIT_THREAD_DEBUG_EVENT	Una estructura **EXIT_THREAD_DEBUG_EVENT** llamado **ExitThread**
LOAD_DLL_DEBUG_EVENT	Una estructura **LOAD_DLL_DEBUG_INFO** llamado **LoadDll**
UNLOAD_DLL_DEBUG_EVENT	Una estructura **UNLOAD_DLL_DEBUG_INFO** llamado **UnloadDll**
EXCEPTION_DEBUG_EVENT	Una estructura **EXCEPTION_DEBUG_INFO** llamado **Excepción**
OUTPUT_DEBUG_STRING_EVENT	Una estructura **OUTPUT_DEBUG_STRING_INFO** llamado **DebugString**
RIP_EVENT	Una estructura **RIP_INFO** llamado **RipInfo**

No voy a entrar en detalles acerca de todas las estructuras de este tutorial, sólo la estructura **CREATE_PROCESS_DEBUG_INFO** serán cubiertos aquí.
Suponiendo que el programa llama a **WaitForDebugEvent** y lo devuelve. Lo primero que debemos hacer es examinar el valor de **dwDebugEventCode** para ver qué tipo de evento de depuración se produjo en el proceso de depuración. Por ejemplo, si el valor de **dwDebugEventCode** es **CREATE_PROCESS_DEBUG_EVENT,** se puede interpretar el miembro **u** como **CreateProcessInfo** y acceder a ella con **u.CreateProcessInfo.**

3. **Haced lo que el programa quiere hacer en respuesta al evento de depuración.** Cuando regresa **WaitForDebugEvent,** que significa un evento de depuración acaba de ocurrir en el proceso de depuración o se produce un tiempo de espera. Su programa tiene que examinar el valor de **dwDebugEventCode** con el fin de reaccionar ante los eventos apropiadamente. En este sentido, es como de procesamiento de mensajes de Windows: usted elige para manejar algunos e ignorar algunos.
4. **Deje que la depuración continúa la ejecución.** Cuando se produce un evento de depuración, Windows suspende el depurado. Cuando haya terminado con el manejo de eventos, es necesario poner en el código depurado en marcha otra vez. Esto se hace llamando a la función **ContinueDebugEvent.**

ContinueDebugEvent proto dwProcessId: DWORD, dwThreadId: DWORD, dwContinueStatus: DWORD

Esta función se reanuda el hilo que fue suspendido con anterioridad, porque se produjo un evento de depuración.

dwProcessId y **dwThreadId** son el proceso y los identificadores de hilo de la rosca que se reanudará. Por lo general, tomar estos dos valores de los miembros **dwProcessId** y **dwThreadId** de la estructura **DEBUG_EVENT.**

dwContinueStatus especifica la manera de seguir el hilo que reportó el evento de depuración. Hay dos valores posibles: **DBG_CONTINUE** y **DBG_EXCEPTION_NOT_HANDLED.** Para todos los eventos de depuración de otro modo, esos dos valores hacen lo mismo: retomar el hilo. La excepción es el **EXCEPTION_DEBUG_EVENT.** Si el hilo informa de un evento de depuración excepción, significa que se produjo una excepción en el subproceso de depuración. Si se especifica **DBG_CONTINUE,** el hilo se ignora su manejo de excepciones y continúe con la ejecución. En este escenario, el programa debe examinar y resolver la misma excepción antes de reanudar el hilo con **DBG_CONTINUE** más la excepción que se produzca de nuevo una y otra vez .... Si se especifica **DBG_EXCEPTION_NOT_HANDLED,** el programa le dice a Windows que no controla la excepción: Windows debe utilizar el controlador de excepciones por omisión de la depuración para controlar la excepción.

En conclusión, si el evento de depuración se refiere a una excepción en el proceso de depuración, debe llamar **a ContinueDebugEvent** con la bandera **DBG_CONTINUE** si el programa ya se ha quitado la causa de la excepción. De lo contrario, el programa debe llamar a **ContinueDebugEvent** con la bandera **DBG_EXCEPTION_NOT_HANDLED.** Excepto en un caso que siempre se debe usar la bandera **DBG_CONTINUE:** el **EXCEPTION_DEBUG_EVENT** primero que tiene el **EXCEPTION_BREAKPOINT** valor en el miembro ExceptionCode. Cuando la depuración se va a ejecutar su primera instrucción, el programa recibirá el evento de depuración excepción. En realidad es una interrupción de depuración (int 3h). Si usted responde llamando **a ContinueDebugEvent** con la bandera **DBG_EXCEPTION_NOT_HANDLED,** Windows NT se negará a ejecutar el código depurado (porque nadie se preocupa por él). Siempre se debe usar la bandera **DBG_CONTINUE** en este caso para decirle a Windows que desea que el hilo para seguir adelante.

5. **Continuar este ciclo en un bucle infinito hasta que el proceso depurado.** Su programa debe estar en un bucle infinito, muy parecido a un bucle de mensajes hasta que se salga depurado. El bucle es el siguiente:

```
. Mientras VERDADERO
invocar WaitForDebugEvent, addr DebugEvent, INFINITO
. Descanso. Si DebugEvent.dwDebugEventCode == EXIT_PROCESS_DEBUG_EVENT
<Handle Events> la depuración
invocar ContinueDebugEvent, DebugEvent.dwProcessId, DebugEvent.dwThreadId,
DBG_EXCEPTION_NOT_HANDLED
. Endw
```

Aquí está el truco: Una vez que iniciar la depuración de un programa, usted no puede desprenderse de la depuración hasta que sale.

Vamos a resumir los pasos de nuevo:

1. **Crear un proceso o adjuntar el programa en un proceso en ejecución.**
2. **Espere a que la depuración de los eventos**
3. **Haced lo que el programa quiere hacer en respuesta al evento de depuración.**
4. **Deje que la depuración continúa la ejecución.**
5. **Continuar este ciclo en un bucle infinito hasta que el proceso depurado**

Ejemplo:

En este ejemplo se depura un programa win32 y muestra información importante como el identificador del proceso, identificación de procesos, base de la imagen y así sucesivamente.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
```

```
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib
. Datos
AppName db "Win32 Debug Ejemplo n º 1", 0
OFN OPENFILENAME <>
FilterString db "Archivos ejecutables", 0, "*. Exe", 0
db "Todos los archivos", 0, "*. *", 0,0
ExitProc db "Las salidas de depuración", 0
Db NewThread "El hilo se crea uno nuevo", 0
EndThread db "El hilo se destruye", 0
ProcessInfo db "del manejador de archivos:% lx", 0DH, 0AH
db "identificador de proceso:% lx", 0Dh, 0Ah
db "Mango Tema:% lx", 0Dh, 0Ah
db "Banco de Imágenes:% lx", 0Dh, 0Ah
db "Inicio Dirección:% lx", 0
. Datos?
tampón db 512 dup (?)
StartInfo STARTUPINFO <>
pi PROCESS_INFORMATION <>
DBEvent DEBUG_EVENT <>
. Código
empezar:
mov ofn.lStructSize, sizeof OFN
mov ofn.lpstrFilter, FilterString desplazamiento
mov ofn.lpstrFile, desplazamiento de búfer
mov ofn.nMaxFile, 512
mov ofn.Flags, OFN_FILEMUSTEXIST o OFN_PATHMUSTEXIST o OFN_LONGNAMES o
OFN_EXPLORER o OFN_HIDEREADONLY
invocar GetOpenFileName, ADDR OFN
. Si eax == TRUE
invocar GetStartupInfo, addr StartInfo
invocar CreateProcess, tampón addr, NULL, NULL, NULL, FALSE, DEBUG_PROCESS
DEBUG_ONLY_THIS_PROCESS +, NULL, NULL, StartInfo addr, addr pi
. Mientras VERDADERO
invocar WaitForDebugEvent, addr DBEvent, INFINITO
. Si DBEvent.dwDebugEventCode == EXIT_PROCESS_DEBUG_EVENT
invocar el cuadro de mensajes, 0, ExitProc addr, addr AppName, MB_OK +
MB_ICONINFORMATION
. Descanso
. Elseif DBEvent.dwDebugEventCode == CREATE_PROCESS_DEBUG_EVENT
invocar wsprintf, addr buffer, ProcessInfo addr, DBEvent.u.CreateProcessInfo.hFile,
DBEvent.u.CreateProcessInfo.hProcess, DBEvent.u.CreateProcessInfo.hThread,
DBEvent.u.CreateProcessInfo.lpBaseOfImage, DBEvent.u.CreateProcessInfo.lpStartAddress
invocar el cuadro de mensajes, 0, buffer de addr, addr AppName, MB_OK +
MB_ICONINFORMATION
. Elseif DBEvent.dwDebugEventCode == EXCEPTION_DEBUG_EVENT
. Si DBEvent.u.Exception.pExceptionRecord.ExceptionCode == EXCEPTION_BREAKPOINT
invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId, DBG_CONTINUE
. Continuar
. Endif
. Elseif DBEvent.dwDebugEventCode == CREATE_THREAD_DEBUG_EVENT
invocar el cuadro de mensajes, 0, NewThread addr, addr AppName, MB_OK +
MB_ICONINFORMATION
. Elseif DBEvent.dwDebugEventCode == EXIT_THREAD_DEBUG_EVENT
invocar el cuadro de mensajes, 0, EndThread addr, addr AppName, MB_OK +
```

```
MB_ICONINFORMATION
 . Endif
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_EXCEPTION_NOT_HANDLED
 . Endw
 invocar CloseHandle, pi.hProcess
 invocar CloseHandle, pi.hThread
 . Endif
 invocar ExitProcess, 0
 poner fin a empezar a
```

Análisis:

El programa rellena la estructura OPENFILENAME y luego llama a GetOpenFileName para permitir al usuario elegir un programa para ser depurado.

```
 invocar GetStartupInfo, addr StartInfo
 invocar CreateProcess, tampón addr, NULL, NULL, NULL, FALSE, DEBUG_PROCESS
DEBUG_ONLY_THIS_PROCESS +, NULL, NULL, StartInfo addr, addr pi
```

Cuando el usuario elige una, llama a **CreateProcess** para cargar el programa. Hace un llamamiento **GetStartupInfo** para llenar la estructura **STARTUPINFO** con sus valores por defecto. Tenga en cuenta que se utiliza en combinación con banderas **DEBUG_PROCESS DEBUG_ONLY_THIS_PROCESS** con el fin de depurar sólo este programa, sin incluir los procesos de sus hijos.

```
 . Mientras VERDADERO
 invocar WaitForDebugEvent, addr DBEvent, INFINITO
```

Cuando el código depurado se carga, entramos en el bucle infinito de depuración, llamando **WaitForDebugEvent. WaitForDebugEvent** no regresará hasta que un evento de depuración se produce en la depuración, ya que especificamos **INFINITE** como segundo parámetro. Cuando un evento de depuración se produjeron, devoluciones y **WaitForDebugEvent** DBEvent está lleno de información sobre el evento de depuración.

```
 . Si DBEvent.dwDebugEventCode == EXIT_PROCESS_DEBUG_EVENT
 invocar el cuadro de mensajes, 0, ExitProc addr, addr AppName, MB_OK +
MB_ICONINFORMATION
 . Descanso
```

En primer lugar, comprobar el valor de **dwDebugEventCode.** Si se trata de **EXIT_PROCESS_DEBUG_EVENT,** nos mostrará un cuadro de mensaje que dice "Las salidas de depuración" y luego salir del bucle de depuración.

```
 . Elseif DBEvent.dwDebugEventCode == CREATE_PROCESS_DEBUG_EVENT
 invocar wsprintf, addr buffer, ProcessInfo addr, DBEvent.u.CreateProcessInfo.hFile,
DBEvent.u.CreateProcessInfo.hProcess, DBEvent.u.CreateProcessInfo.hThread,
DBEvent.u.CreateProcessInfo.lpBaseOfImage, DBEvent.u.CreateProcessInfo.lpStartAddress
 invocar el cuadro de mensajes, 0, buffer de addr, addr AppName, MB_OK +
MB_ICONINFORMATION
```

Si el valor es **dwDebugEventCode CREATE_PROCESS_DEBUG_EVENT,** entonces mostrar la información de varias interesante acerca de la depuración en un cuadro de mensaje. Se obtiene información de los **u.CreateProcessInfo.** CreateProcessInfo es una estructura de tipo **CREATE_PROCESS_DEBUG_INFO.** Puede obtener más información sobre esta estructura de la API de Win32 referencia.

```
 . Elseif DBEvent.dwDebugEventCode == EXCEPTION_DEBUG_EVENT
 . Si DBEvent.u.Exception.pExceptionRecord.ExceptionCode == EXCEPTION_BREAKPOINT
```

```
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId, DBG_CONTINUE
 . Continuar
 . Endif
```

Si el valor es **dwDebugEventCode EXCEPTION_DEBUG_EVENT,** debemos de buscar más el tipo exacto de excepción. Es una larga línea de referencia de estructura anidada, pero puede obtener el tipo de excepción de un miembro de **ExceptionCode.** Si el valor es **ExceptionCode EXCEPTION_BREAKPOINT** y se produce por primera vez (o si estamos seguros de que la depuración no se ha incorporado int 3h), podemos asumir con seguridad que esta excepción se produjo cuando la depuración se va a ejecutar su primera instrucción. Cuando hayamos terminado con el proceso, debemos llamar **a ContinueDebugEvent** con la bandera **DBG_CONTINUE** para dejar correr la depuración. A continuación, volvemos a esperar a que el evento de depuración que viene.

```
 . Elseif DBEvent.dwDebugEventCode == CREATE_THREAD_DEBUG_EVENT
 invocar el cuadro de mensajes, 0, NewThread addr, addr AppName, MB_OK +
MB_ICONINFORMATION
 . Elseif DBEvent.dwDebugEventCode == EXIT_THREAD_DEBUG_EVENT
 invocar el cuadro de mensajes, 0, EndThread addr, addr AppName, MB_OK +
MB_ICONINFORMATION
 . Endif
```

Si el valor es **dwDebugEventCode CREATE_THREAD_DEBUG_EVENT** o **EXIT_THREAD_DEBUG_EVENT,** se muestra un cuadro de mensaje que lo diga.

```
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_EXCEPTION_NOT_HANDLED
 . Endw
```

Salvo en el caso **EXCEPTION_DEBUG_EVENT** anterior, llamamos **a ContinueDebugEvent** con la bandera **DBG_EXCEPTION_NOT_HANDLED** para reanudar la depuración.

```
 invocar CloseHandle, pi.hProcess
 invocar CloseHandle, pi.hThread
```

Cuando se sale depurado, estamos fuera del circuito de depuración y debe cerrar el proceso y el hilo se encarga de la depuración. Cierre de los mangos no quiere decir que están matando el proceso / hilo. Sólo quiere decir que no queremos utilizar esos mangos para referirse al proceso / hilo más.

TUTORIAL 29: DEPURACIÓN DE LA API WIN32 PARTE 2

Seguimos con el tema de la API de depuración de Win32. En este tutorial, vamos a aprender cómo modificar el proceso de depuración.

TEORÍA:

En el tutorial anterior, sabemos cómo cargar el código depurado y manejar eventos de depuración que se producen en su proceso. Con el fin de ser útil, nuestro programa debe ser capaz de modificar el proceso de depuración. Hay varias APIs para este fin.

- **ReadProcessMemory** Esta función le permite leer la memoria en el proceso especificado. El prototipo de función es como sigue:

ReadProcessMemory proto hProcess: DWORD, lpBaseAddress: DWORD, lpBuffer: DWORD, nSize: DWORD, lpNumberOfBytesRead: DWORD

hProcess es el identificador para el proceso que desea leer.
lpBaseAddress es la dirección en el proceso de destino que desea comenzar a leer. Por ejemplo, si usted quiere leer 4 bytes del proceso de depuración a partir de 401000h, el valor de este parámetro debe ser 401000h.
lpBuffer es la dirección de la memoria intermedia para recibir los bytes leídos del proceso.
nSize es el número de bytes que desea leer
lpNumberOfBytesRead es la dirección de la variable de tamaño dword que recibe el número de bytes leídos en realidad. Si no se preocupan por él, usted puede usar NULL.

- **WriteProcessMemory** es la contraparte de **ReadProcessMemory.** Le permite escribir en la memoria del proceso de destino. Sus parámetros son exactamente los mismos que los de **ReadProcessMemory**

Las siguientes dos funciones de la API necesita un poco de antecedentes sobre el contexto. Bajo un sistema operativo multitarea como Windows, no puede haber varios programas que se ejecutan al mismo tiempo. Windows da a cada hilo de una porción de tiempo. Cuando esa porción de tiempo, Windows se congela el hilo de la actualidad y se pasa a la siguiente conversación que tiene la más alta prioridad. Justo antes de pasar al otro hilo, Windows guarda los valores en los registros de la rosca de la actualidad de manera que cuando llegue el momento de retomar el hilo, Windows puede restaurar la última * entorno * de ese hilo. Los valores guardados de los registros se denominan colectivamente un contexto.
Volviendo a nuestro tema. Cuando se produce un evento de depuración, Windows suspende el depurado. El contexto de la depuración se guarda. Puesto que la depuración se suspende, se puede estar seguro de que los valores en el contexto no cambiará. Podemos obtener los valores en el contexto de **GetThreadContext** y podemos cambiar con **SetThreadContext.**
Estas API dos son muy poderosos. Con ellos, usted tiene en sus manos el poder VxD-al igual que en el depurado: puede modificar los valores de los registros guardados y justo antes de la ejecución depurado hojas de vida, los valores en el contexto va a ser escrito de nuevo en los registros. Cualquier cambio realizado en el contexto se refleja de vuelta a la depuración. Piense en esto: incluso se puede alterar el valor de la EIP registrarse y desviar el flujo de ejecución a cualquier lugar que te gusta! Usted no será capaz de hacer eso en circunstancias normales.

GetThreadContext proto hThread: DWORD, lpContext: DWORD

hThread es el identificador del subproceso que desea obtener el contexto de la
lpContext es la dirección de la estructura del **contexto** que se llena cuando la función devuelve correctamente.

SetThreadContext tiene exactamente los mismos parámetros. Vamos a ver lo que es una estructura CONTEXT se ve así:

- **CONTEXTO STRUCT**
- **ContextFlags dd?**

; En esta sección se devuelve si ContextFlags contiene el valor CONTEXT_DEBUG_REGISTERS

- **iDr0 dd?**
  iDr1 dd?
  iDr2 dd?
  EAF3 dd?
  iDr6 dd?
  iDr7 dd?

- **; En esta sección se devuelve si ContextFlags contiene el valor de CONTEXT_FLOATING_POINT**

- **FloatSave FLOATING_SAVE_AREA <>**
- **; En esta sección se devuelve si ContextFlags contiene el valor CONTEXT_SEGMENTS**
- **Reggs dd?**
  regFs dd?
  Reges dd?
  regds dd?
- **; En esta sección se devuelve si ContextFlags contiene el valor de CONTEXT_INTEGER**
- **regEdi dd?**
  regEsi dd?
  regEbx dd?
  regEdx dd?
  regEcx dd?
  regEax dd?
- **; En esta sección se devuelve si ContextFlags contiene el valor de CONTEXT_CONTROL**
- **regEbp dd?**
  regEip dd?
  regCs dd?
  regFlag dd?
  regEsp dd?
  regSs dd?
- **; En esta sección se devuelve si ContextFlags contiene el valor CONTEXT_EXTENDED_REGISTERS**
- **ExtendedRegisters db MAXIMUM_SUPPORTED_EXTENSION dup (?) CONTEXTO TERMINA**

Como se puede observar, los miembros de estas estructuras son imitadores de los registros del procesador real. Antes de poder utilizar esta estructura, es necesario especificar qué grupos de registros que desea leer / escribir en el miembro **ContextFlags.** Por ejemplo, si usted quiere leer / escribir todos los registros, se debe especificar en el **CONTEXT_FULL ContextFlags.** Si sólo desea leer / escribir regEbp, regEip, regCs, regFlag, regEsp o regSs, debe especificar en el **CONTEXT_CONTROL ContextFlags.**

Una cosa que usted debe recordar cuando se utiliza la estructura del **contexto:** debe estar alineado en un límite DWORD otra cosa que conseguiría resultados extraños en NT. Usted debe poner "align dword" justo por encima de la línea que se declara, de esta manera:

align dword
Micontexto CONTEXTO <>

EJEMPLO:

El primer ejemplo muestra el uso de **DebugActiveProcess.** En primer lugar, es necesario ejecutar un destino denominado win.exe que va en un bucle infinito, justo antes de que la ventana se muestra en la pantalla. Luego de ejecutar el ejemplo, se adhiere a win.exe y modificar el código de win.exe de manera que las salidas de win.exe el bucle infinito y muestra su propia ventana.

0.386
. Modelo plano, stdcall

```
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib

. Datos
AppName db "Win32 Debug Ejemplo n º 2", 0
ClassName db "SimpleWinClass", 0
SearchFail db "No se encuentra el proceso de destino", 0
Db TargetPatched "Target parcheado", 0
tampón DW 9090h

. Datos?
DBEvent DEBUG_EVENT <>
ProcessId dd?
ThreadId dd?
align dword
contexto CONTEXTO <>

. Código
empezar:
invocar FindWindow, ClassName addr, NULL
. Si eax! = NULL
invocar GetWindowThreadProcessId, eax, addr ProcessId
mov ThreadId, eax
invocar DebugActiveProcess, ProcessId
. Mientras VERDADERO
invocar WaitForDebugEvent, addr DBEvent, INFINITO
. Descanso. Si DBEvent.dwDebugEventCode == EXIT_PROCESS_DEBUG_EVENT
. Si DBEvent.dwDebugEventCode == CREATE_PROCESS_DEBUG_EVENT
mov context.ContextFlags, CONTEXT_CONTROL
invocar GetThreadContext, DBEvent.u.CreateProcessInfo.hThread, el contexto addr
invocar WriteProcessMemory, DBEvent.u.CreateProcessInfo.hProcess, context.regEip, tampón addr, 2,
NULL
invocar el cuadro de mensajes, 0, addr TargetPatched, addr AppName, MB_OK +
MB_ICONINFORMATION
. Elseif DBEvent.dwDebugEventCode == EXCEPTION_DEBUG_EVENT
. Si DBEvent.u.Exception.pExceptionRecord.ExceptionCode == EXCEPTION_BREAKPOINT
invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId, DBG_CONTINUE
. Continuar
. Endif
. Endif
invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_EXCEPTION_NOT_HANDLED
. Endw
. Más
invocar el cuadro de mensajes, 0, SearchFail addr, addr AppName, MB_OK + MB_ICONERROR. endif
invocar ExitProcess, 0
poner fin a empezar a

; ------------------------------------------------ ------------------
; El código fuente parcial de win.asm, nuestro depurado. De hecho, es
; El ejemplo simple ventana en el tutorial 2, con un bucle infinito insertada
, Justo antes de entrar en el bucle de mensajes.
; ------------------------------------------------ ----------------------
```

```
.....
mov wc.hIconSm, eax
invocar LoadCursor, NULL, IDC_ARROW
wc.hCursor mov, eax
invocar RegisterClassEx, addr wc
INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \ CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \ hInst, NULL
mov hwnd, eax
jmp $ <---- Este es nuestro bucle infinito.  Reúne a EB FE
invocar ShowWindow, hwnd, SW_SHOWNORMAL
invocar UpdateWindow, hwnd
. Mientras VERDADERO
invocar GetMessage, msg ADDR, NULL, 0,0
. Descanso. If (! Eax)
invocar TranslateMessage, ADDR msg
invocar DispatchMessage, ADDR msg
. Endw
mov eax, msg.wParam
ret
WinMain endp
```

Análisis:

```
invocar FindWindow, ClassName addr, NULL
```

Nuestro programa tiene que unirse a la depuración con **DebugActiveProcess** que requiere el ID del proceso de la depuración. Podemos obtener el ID del proceso llamando a **GetWindowThreadProcessId** que a su vez necesita el identificador de ventana como parámetro. Así que tenemos que obtener el handle de la ventana en primer lugar.

Con **FindWindow,** podemos especificar el nombre de la clase de ventana que necesitamos. Devuelve el identificador de la ventana creada por esa clase de ventana. Si devuelve **NULL,** no hay ninguna ventana de la clase está presente.

```
. Si eax! = NULL
invocar GetWindowThreadProcessId, eax, addr ProcessId
mov ThreadId, eax
invocar DebugActiveProcess, ProcessId
```

Después de obtener el ID del proceso, que podemos llamar **DebugActiveProcess.** Luego entramos en el bucle de depuración de espera para los eventos de depuración.

```
. Si DBEvent.dwDebugEventCode == CREATE_PROCESS_DEBUG_EVENT
mov context.ContextFlags, CONTEXT_CONTROL
invocar GetThreadContext, DBEvent.u.CreateProcessInfo.hThread, el contexto addr
```

Cuando lleguemos **CREATE_PROCESS_DEBUG_INFO,** significa que la depuración se ha suspendido, listo para que hagamos la cirugía de su proceso. En este ejemplo, se sobreponen a la instrucción de bucle infinito en el código depurado (0EBh 0FEh) con NOP (90h 90h).

En primer lugar, tenemos que obtener la dirección de la instrucción. Dado que la depuración ya está en el bucle en el momento en nuestro programa que se le atribuye, eip siempre apuntará a la instrucción. Todo lo que necesitamos hacer es obtener el valor de eip. Usamos **GetThreadContext** para lograr ese objetivo. Hemos establecido el miembro de **ContextFlags CONTEXT_CONTROL** con el fin de

decirle **GetThreadContext** que queremos que se llene el "control" registrar a los miembros de la estructura del **contexto.**

invocar WriteProcessMemory, DBEvent.u.CreateProcessInfo.hProcess, context.regEip, tampón addr, 2, NULL

Ahora que tenemos el valor de eip, podemos llamar a **WriteProcessMemory** para sobrescribir el "jmp $" instrucción con NOP, lo que efectivamente ayudar a la salida de depurado el bucle infinito. Después de que aparezca el mensaje para el usuario y luego llamar **a ContinueDebugEvent** para reanudar la depuración. Dado que la instrucción "jmp $" se reemplaza por el NOP, la depuración será capaz de continuar con la muestra de su ventana y entrar en el bucle de mensajes. La evidencia es que veremos su ventana en la pantalla.

El otro ejemplo utiliza un método ligeramente diferente para romper el código depurado del bucle infinito.

.....
.....
. Si DBEvent.dwDebugEventCode == CREATE_PROCESS_DEBUG_EVENT
mov context.ContextFlags, CONTEXT_CONTROL
invocar GetThreadContext, DBEvent.u.CreateProcessInfo.hThread, el contexto addr
añadir context.regEip, 2
invocar SetThreadContext, DBEvent.u.CreateProcessInfo.hThread, el contexto addr
invocar el cuadro de mensajes, 0, addr LoopSkipped, addr AppName, MB_OK + MB_ICONINFORMATION
.....
.....

Todavía llama **GetThreadContext** para obtener el valor actual de la EIP, pero en lugar de sobrescribir el "jmp $" instrucción, se incrementa el valor de **regEip** por 2 a "saltarse" la instrucción. El resultado es que cuando el código depurado recupera el control, se reanuda la ejecución en la siguiente instrucción después de "jmp $".

Ahora usted puede ver el poder de Get / SetThreadContext. También puede modificar las imágenes de otros registros, así como sus valores se refleja de vuelta a la depuración. Usted puede incluso insertar la instrucción int 3h para poner puntos de interrupción en el proceso de depuración.

TUTORIAL 30: API WIN32 DEBUG PARTE 3

En este tutorial, continuar la exploración de la API de depuración de Win32. En concreto, vamos a aprender a seguir la depuración.

Historial de revisiones:

02/12/2000: Se olvidó de DWORD a alinear la estructura del contexto

TEORÍA:

Si ha utilizado un depurador antes, deberían estar familiarizados con el rastreo. Cuando se "traza" un programa, el programa se detiene después de ejecutar cada instrucción, que le da la oportunidad de examinar los valores de los registros / memoria. Un solo paso a paso es el nombre oficial del trazado.
La función de un solo paso es proporcionada por la propia CPU. El bit 8 del registro de la bandera se llama **la bandera trampa.** Si esta bandera (bit) se establece, la CPU se ejecuta en modo paso a paso. La CPU va a generar una excepción de depuración después de cada instrucción. Después de la excepción de depuración se genera, la bandera de la trampa se borra automáticamente.
También podemos de un solo paso el código depurado, utilizando la API de depuración de Win32. Los pasos son los siguientes:

1. Llama **GetThreadContext,** especificando **CONTEXT_CONTROL** en **ContextFlags,** para obtener el valor del registro del pabellón.
2. Establecer el bit de la trampa en el miembro **regFlag** de la estructura del **contexto**
3. llame **SetThreadContext**
4. Espere a que los eventos de depuración como de costumbre. La depuración se ejecutarán en modo paso a paso. Después de que se ejecuta cada instrucción, nos pondremos en contacto con el valor **EXCEPTION_DEBUG_EVENT EXCEPTION_SINGLE_STEP** en **u.Exception.pExceptionRecord.ExceptionCode**
5. Si usted necesita para trazar la siguiente instrucción, es necesario establecer el bit de la trampa de nuevo.

EJEMPLO:

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ kernel32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ user32.inc
includelib \ masm32 \ lib \ kernel32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib

. Datos
AppName db "Win32 Debug Ejemplo n º 4", 0
OFN OPENFILENAME <>
FilterString db "Archivos ejecutables", 0, "*. Exe", 0
db "Todos los archivos", 0, "*. *", 0,0
ExitProc db "Las salidas de depuración", 0DH, 0AH
db "Instrucciones totales ejecutadas:% lu", 0
TotalInstruction dd 0

. Datos?
tampón db 512 dup (?)
StartInfo STARTUPINFO <>
pi PROCESS_INFORMATION <>
DBEvent DEBUG_EVENT <>
align dword
contexto CONTEXTO <>

. Código
```

```
 empezar:
 mov ofn.lStructSize, sizeof OFN
 mov ofn.lpstrFilter, FilterString OFFSET
 mov ofn.lpstrFile, desplazamiento de búfer
 mov ofn.nMaxFile, 512
 mov ofn.Flags, OFN_FILEMUSTEXIST o OFN_PATHMUSTEXIST o
OFN_LONGNAMES o OFN_EXPLORER o OFN_HIDEREADONLY
 invocar GetOpenFileName, ADDR OFN
 . Si eax == TRUE
 invocar GetStartupInfo, addr StartInfo
 invocar CreateProcess, tampón addr, NULL, NULL, NULL, FALSE,
DEBUG_PROCESS DEBUG_ONLY_THIS_PROCESS +, NULL, NULL, StartInfo
addr, addr pi
 . Mientras VERDADERO
 invocar WaitForDebugEvent, addr DBEvent, INFINITO
 . Si DBEvent.dwDebugEventCode == EXIT_PROCESS_DEBUG_EVENT
 invocar wsprintf, addr buffer, addr ExitProc, TotalInstruction
 invocar el cuadro de mensajes, 0, buffer de addr, addr AppName, MB_OK +
MB_ICONINFORMATION
 . Descanso
 . Elseif DBEvent.dwDebugEventCode == EXCEPTION_DEBUG_EVENT. Si
DBEvent.u.Exception.pExceptionRecord.ExceptionCode ==
EXCEPTION_BREAKPOINT
 mov context.ContextFlags, CONTEXT_CONTROL
 invocar GetThreadContext, pi.hThread, el contexto addr
 o context.regFlag, 100h
 invocar SetThreadContext, pi.hThread, el contexto addr
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_CONTINUE
 . Continuar
 . Elseif DBEvent.u.Exception.pExceptionRecord.ExceptionCode ==
EXCEPTION_SINGLE_STEP
 inc TotalInstruction
 invocar GetThreadContext, pi.hThread, el contexto o addr context.regFlag, 100h
 invocar SetThreadContext, pi.hThread, el contexto addr
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_CONTINUE
 . Continuar
 . Endif
 . Endif
 invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId,
DBG_EXCEPTION_NOT_HANDLED
 . Endw
 . Endif
 invocar CloseHandle, pi.hProcess
 invocar CloseHandle, pi.hThread
 invocar ExitProcess, 0
 poner fin a empezar a
```

ANÁLISIS:

El programa muestra el cuadro de diálogo OpenFile. Cuando el usuario elige un archivo ejecutable, se ejecuta el programa en modo paso a paso, Couting el número de instrucciones ejecutadas hasta que se salga depurado.

. Elseif DBEvent.dwDebugEventCode == EXCEPTION_DEBUG_EVENT. Si DBEvent.u.Exception.pExceptionRecord.ExceptionCode == EXCEPTION_BREAKPOINT

Aprovechamos esta oportunidad para establecer la depuración en modo paso a paso. Recuerde que Windows envía un EXCEPTION_BREAKPOINT justo antes de que se ejecuta la primera instrucción de la depuración.

mov context.ContextFlags, CONTEXT_CONTROL
invocar GetThreadContext, pi.hThread, el contexto addr

Hacemos un llamado **GetThreadContext** para llenar la estructura del **contexto** con los valores actuales en los registros de la depuración. Más específicamente, se necesita el valor actual del registro del pabellón.

o context.regFlag, 100h

Hemos establecido el bit de trampa (8 bits) en el registro de la imagen del pabellón.

invocar SetThreadContext, pi.hThread, el contexto addr
invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId, DBG_CONTINUE
. Continuar

Entonces te llamamos para **SetThreadContext** para sobrescribir los valores en la estructura del **contexto** con el nuevo (s) y llame **a ContinueDebugEvent** con la bandera **DBG_CONTINUE** para reanudar la depuración.

. Elseif DBEvent.u.Exception.pExceptionRecord.ExceptionCode == EXCEPTION_SINGLE_STEP
inc TotalInstruction

Cuando una operación se ejecuta en la depuración, recibimos una **EXCEPTION_DEBUG_EVENT.** Debemos examinar el valor de **u.Exception.pExceptionRecord.ExceptionCode.** Si el valor es **EXCEPTION_SINGLE_STEP,** entonces este evento de depuración se genera debido a la modalidad de un solo paso. En este caso, se puede incrementar el **TotalInstruction** variable en uno, porque sabemos que exactamente una instrucción se ejecuta en el depurado.

invocar GetThreadContext, pi.hThread, el contexto o addr context.regFlag, 100h
invocar SetThreadContext, pi.hThread, el contexto addr
invocar ContinueDebugEvent, DBEvent.dwProcessId, DBEvent.dwThreadId, DBG_CONTINUE
. Continuar

Desde la bandera de la trampa se borra después de la excepción de depuración se genera, se debe establecer el indicador de la trampa de nuevo si queremos seguir en el modo paso a paso.
Advertencia: No usar el ejemplo en este tutorial con un gran programa: el rastreo es lento. Puede que tenga que esperar diez minutos antes de poder cerrar el código depurado.

TUTORIAL 32: INTERFAZ DE MÚLTIPLES DOCUMENTOS (MDI)

En este tutorial se muestra cómo crear una aplicación MDI. En realidad no es demasiado difícil de hacer.

TEORÍA:

Interfaz de documentos múltiples (MDI) es una especificación para aplicaciones que manejan documentos multple al mismo tiempo. Usted está familiarizado con el Bloc de notas: Es un ejemplo de interfaz de documento único (SDI). Bloc de notas sólo puede manejar un documento a la vez. Si desea abrir otro documento, usted tiene que cerrar la anterior en primer lugar. Como se puede imaginar, es bastante engorroso. Contrasta con Microsoft Word: Word puede abrir documentos arbitrarios, al mismo tiempo y permitir al usuario elegir qué documento para su uso. Microsoft Word es un ejemplo de interfaz de múltiples documentos (MDI).

Aplicación MDI tiene varias características que son distintivos. Voy a enumerar algunas de ellas:

- Dentro de la ventana principal, puede haber varias ventanas de los niños en el área de cliente. Todas las ventanas secundarias se recortan al área de cliente.
- Al minimizar una ventana secundaria, se reduce al mínimo a la esquina inferior izquierda del área de cliente de la ventana principal.
- Al maximizar la ventana achild, su título se combina con la de la ventana principal.
- Se puede cerrar una ventana secundaria pulsando Ctrl + F4 y cambiar el foco entre las ventanas secundarias pulsando Ctrl + Tab

La ventana principal que contiene el niño las ventanas que se llama **una ventana de marco.** Su área de cliente es donde viven las ventanas secundarias, de ahí el nombre de "marco". Su trabajo es un poco más elaborado que una ventana normal, ya que tiene que manejar algún tipo de coordinación para el MDI.

Para el control de un número arbitrario de ventanas secundarias en el área de cliente, usted necesita una ventana especial que se llama **ventana del cliente.** Usted puede pensar de esta ventana del cliente como una ventana transparente que cubre el área de cliente de la ventana de marco. Es esta ventana del cliente que es el padre real de las ventanas secundarias MDI. La ventana del cliente es el supervisor real de las ventanas MDI secundarias.

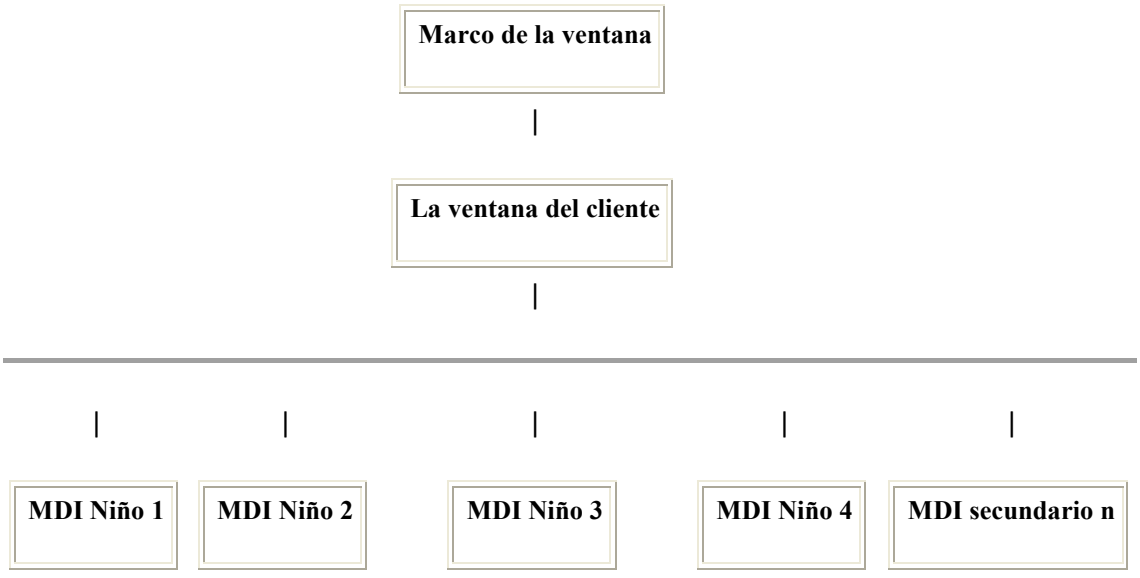


Figura 1. La jerarquía de una aplicación MDI

CREACIÓN DEL MARCO DE VENTANA

Ahora podemos dirigir nuestra atención a los detalles. En primer lugar usted necesita para crear una ventana de marco. Se crea la misma forma que la ventana normal: llamando CreateWindowEx. Hay dos grandes diferencias de una ventana normal.

La primera diferencia es que usted **DEBE** llamar **DefFrameProc** lugar de **DefWindowProc** para procesar los mensajes de Windows la ventana no quieren manejar. Esta es una manera de dejar que Windows haga el trabajo sucio de mantener la aplicación MDI para usted. Si olvidó usar **DefFrameProc,** su aplicación no tendrá la función MDI. . Período **DefFrameProc** tiene la siguiente sintaxis:

```
DefFrameProc proc hwndFrame: DWORD,
  hwndClient: DWORD,
  uMsg: DWORD,
  wParam: DWORD,
  lParam: DWORD
```

Si se compara con **DefFrameProc DefWindowProc,** te darás cuenta de que la única diferencia entre ellos es que **DefFrameProc** tiene 5 parámetros, mientras que **DefWindowProc** tiene sólo 4. El parámetro extra es el handle de la ventana del cliente. Este identificador es necesario para que Windows pueda enviar a inhaladores de dosis medidas relacionadas con los mensajes a la ventana del cliente.

La segunda diferencia es que, usted debe llamar a **TranslateMDISysAccel** en el bucle de mensajes de la ventana del marco. Esto es necesario si desea que Windows para manejar MDI relacionados con combinaciones de teclas aceleradoras como Ctrl + F4, Ctrl + Tab para usted. Se tiene la siguiente sintaxis:

```
TranslateMDISysAccel proc hwndClient: DWORD,
  lpMsg: DWORD
```

El primer parámetro es el identificador de la ventana del cliente. Esto no debería ser una sorpresa para usted, porque es la ventana del cliente que es el padre de todas las ventanas secundarias MDI. El segundo parámetro es la dirección de la estructura MSG que llenó llamando **GetMessage.** La idea es pasar de la estructura MSG a la ventana del cliente por lo que podría examinar si la estructura MSG contiene las pulsaciones de teclas MDI-relacionados. Si es así, procesa el mensaje en sí mismo y devuelve un valor distinto de cero, de lo contrario, devuelve FALSE.

Los pasos en la creación de la ventana de marco se pueden resumir como sigue:

1. Rellene la estructura WNDCLASSEX como de costumbre
2. Registrar la clase de marco de la ventana llamando al **RegisterClassEx**
3. Crear el marco de la ventana llamando **CreateWindowEx**
4. Dentro del bucle de mensajes, llame **TranslateMDISysAccel.**
5. En el procedimiento de ventana, pasar los mensajes no procesados a **DefFrameProc** lugar de **DefWindowProc.**

CREACIÓN DE LA VENTANA DEL CLIENTE

Ahora que tenemos la ventana de marco, podemos crear la ventana del cliente. La clase de la ventana del cliente está pre-registrado por Windows. El nombre de la clase es "MDICLIENT". También es necesario pasar la dirección de una estructura **CLIENTCREATESTRUCT** a **CreateWindowEx.** Esta estructura tiene la siguiente definición:

```
CLIENTCREATESTRUCT estructura
  hWindowMenu dd?
  idFirstChild dd?
CLIENTCREATESTRUCT termina
```

hWindowMenu es la palanca para el submenú que Windows se anexará la lista de nombres de ventana hija MDI. Esta función requiere una pequeña explicación. Si alguna vez utiliza una aplicación MDI como Microsoft Word antes, te darás cuenta de que hay un submenú llamado "ventana" que, al activarse, muestra MenuItems diversos relacionados con la gestión de ventanas y en el fondo, la lista de la ventana secundaria MDI en la actualidad abierto. Dicha lista está mantenida internamente por el propio Windows: usted no tiene que hacer nada especial para ello. Simplemente pasar el identificador del submenú que desee de la lista para aparecer en **hWindowMenu** y Windows se encargará del resto. Observe que el submenú puede ser **cualquier** submenú: no tiene que ser la que se denomina una "ventana". La conclusión es que, usted **debe** pasar el identificador al submenú desea que la lista de la ventana que aparezca. Si usted no desea que la lista, sólo hay que poner NULL en **hWindowMenu.** Usted consigue el mango para el submenú llamando **GetSubMenu.**

idFirstChild es el identificador de la ventana secundaria MDI **en primer lugar.** Incrementos de la ID de Windows para cada ventana secundaria MDI nueva de la aplicación creada. Por ejemplo, si se pasa de 100 a este campo, la ventana secundaria MDI primero tendrá la identificación de 100, el segundo tendrá la identificación de 101 y así sucesivamente. Este ID se envía a la ventana de marco a través de WM_COMMAND cuando la ventana secundaria MDI se selecciona de la lista de ventanas. Normalmente, usted pasará esto "no controlada" WM_COMMAND mensajes a DefFrameProc. Yo uso la palabra "no controlada", porque los objetos MenuItem en la lista de ventanas no son creados por la aplicación por lo que su aplicación no conoce sus documentos de identidad y no tiene el controlador para ellos. Este es otro caso especial de la ventana de marco MDI: si usted tiene la lista de ventanas, debe modificar el controlador de WM_COMMAND un poco como esto:

```
. Elseif uMsg == WM_COMMAND
. Si lParam == 0; este mensaje se genera a partir de un menú
mov eax, wParam
. Si ax == IDM_CASCADE
.....
. Elseif ax == IDM_TILEVERT
.....
. Más
invocar DefFrameProc, hwndFrame, hwndClient, uMsg, wParam, lParam
ret
. Endif
```

Normalmente, usted acaba de pasar por alto los mensajes de los casos no controladas. Pero en el caso de inhaladores de dosis medida, si los ignora, cuando el usuario hace clic sobre el nombre de una ventana secundaria MDI en la lista de la ventana, esa ventana no estará activo. Es necesario pasar a la **DefFrameProc** para que puedan ser manejados adecuadamente.

Una advertencia sobre el valor de **idFirstChild:** no se debe utilizar 0. Su lista de ventanas no se comportará correctamente, es decir. la marca de verificación no aparecerá en la parte delantera del nombre del niño MDI primero a pesar de que está activo. Elija un valor seguro, tal como 100 o superior.

Después de haber completado la estructura CLIENTCREATESTRUCT, puede crear la ventana cliente llamando **CreateWindowEx** con el nombre de la clase predefinida ", MDICLIENT", y pasando la dirección de la estructura CLIENTCREATESTRUCT en lParam. También debe especificar el identificador de la ventana de marco en el parámetro hwndParent por lo que Windows sabe que la relación padre-hijo entre el marco de la ventana y la ventana del cliente. Los estilos de ventana que debe utilizar son: WS_CHILD, WS_VISIBLE y WS_CLIPCHILDREN. Si se le olvida WS_VISIBLE, usted no verá las ventanas MDI secundarias, incluso si se han creado con éxito.

Los pasos en la creación de la ventana del cliente son los siguientes:

1. Obtener el mango para el submenú que desea añadir a la lista de ventanas.
2. Ponga el valor del menú de manejo, junto con el valor que desea utilizar como el identificador de la ventana secundaria MDI por primera vez en una estructura CLIENTCREATESTRUCT
3. llamar a CreateWindowEx con el nombre de la clase "MDICLIENT", pasando la dirección de la estructura CLIENTCREATESTRUCT que acaba de completar en lParam.

CREACIÓN DE LA VENTANA SECUNDARIA MDI

Ahora usted tiene tanto en el marco de la ventana y la ventana del cliente. La etapa está ahora listo para la creación de la ventana secundaria MDI. Hay dos maneras de hacerlo.

- Puede enviar un mensaje WM_MDICREATE de **la ventana del cliente,** pasando la dirección de una estructura de tipo MDICREATESTRUCT en wParam. Este es el más fácil y el método habitual de creación de ventana secundaria MDI.
- **. Datos?**
  mdicreate MDICREATESTRUCT <>
  ....
  . Código
  .....
  [Llenar los miembros de mdicreate]
  .....
  invoca SendMessage, hwndClient, WM_MDICREATE, mdicreate addr, 0

  SendMessage devuelve el handle de la ventana secundaria MDI recién creado en caso de éxito. No es necesario guardar el mango sin embargo. Se puede obtener por otros medios si se quiere. MDICREATESTRUCT tiene la siguiente definición.

  MDICREATESTRUCT STRUCT
  szClass DWORD?
  szTitle DWORD?
  hOwner DWORD?
  x DWORD?
  y DWORD?
  lx DWORD?
  DWORD mente?
  DWORD estilo?
  lParam DWORD?
  MDICREATESTRUCT TERMINA

szClass	la dirección de la clase de ventana que desea utilizar como plantilla para la ventana secundaria MDI.
szTitle	la dirección del texto que desea que aparezca en la barra de título de la ventana secundaria
hOwner	el mango ejemplo de la aplicación
x, y, Lx, Ly	la coordenada superior izquierda y el ancho y la altura de la ventana secundaria
estilo	infantil estilo de ventana. Si se crea la ventana del cliente con MDIS_ALLCHILDSTYLES, se puede usar cualquier estilo de ventana.

lParam	una aplicación definida por el valor de 32 bits. Esta es una manera de compartir valores entre las ventanas MDI. Si no hay que usarla, establecen a NULL

- Usted puede llamar a **CreateMDIWindow.** Esta función tiene la siguiente sintaxis:

  CreateMDIWindow proto lpClassName: DWORD
  lpWindowName: DWORD
  dwStyle: DWORD
  x: DWORD
  y: DWORD
  nWidth: DWORD
  nHeight: DWORD
  hwndParent: DWORD
  hInstance: DWORD
  lParam: DWORD

Si te fijas bien en los parámetros, usted encontrará que son idénticos a los miembros de la estructura MDICREATESTRUCT, a excepción de la **hwndParent.** Esencialmente es el mismo número de parámetros que se pasan con WM_MDICREATE. MDICREATESTRUCT no tiene el campo **hwndParent** porque usted tiene que pasar toda la estructura de la ventana del cliente correcto con SendMessage de todos modos.

En este punto, es posible que tenga algunas preguntas: ¿qué método se debe usar? ¿Cuál es la diferencia entre los dos? Aquí está la respuesta:

El método WM_MDICREATE sólo puede crear la ventana secundaria MDI en la misma rosca que el código de llamada. Por ejemplo, si su aplicación tiene 2 hilos, y el hilo se crea primero la ventana de marco MDI, si el segundo hilo quiere crear un formulario MDI secundario, debe hacerlo con CreateMDIChild: el envío de mensajes WM_MDICREATE que el primer hilo no va a funcionar. Si su solicitud es de un único subproceso, puede utilizar cualquiera de los métodos. (Yap Gracias por la corrección - 04/24/2002)

Un poco más de detalle las necesidades a cubrir por el procedimiento de ventana de la MDI. Como en el caso de ventana de marco, no debe llamar a **DefWindowProc** para manejar los mensajes no procesados. En su lugar, debe utilizar **DefMDIChildProc.** Esta función tiene exactamente los mismos parámetros que **DefWindowProc.**

Además de WM_MDICREATE, hay otros MDI mensajes relacionados con la ventana. Los voy a enumerar a continuación:

WM_MDIACTIVATE	Este mensaje puede ser enviado por la aplicación de la ventana del cliente para instruir a la ventana del cliente para activar el MDI secundario seleccionado. Cuando la ventana de cliente recibe el mensaje, se activa la ventana secundaria MDI seleccionado y lo envía WM_MDIACTIVATE para el niño ser desactivado y activado. El uso de este mensaje es doble: puede ser utilizado por la aplicación para activar la ventana secundaria deseada. Y puede ser utilizado por la ventana secundaria MDI sí mismo como el indicador que está siendo activado / desactivado. Por ejemplo, si cada ventana secundaria MDI tiene un menú diferente, que puede aprovechar esta oportunidad para cambiar el menú de la ventana de marco cuando se activa / desactiva.
WM_MDICASCADE	Estos mensajes de manejar la disposición de las ventanas secundarias

WM_MDITILE **WM_MDIICONARRANGE**	MDI. Por ejemplo, si desea que las ventanas secundarias MDI que se organizan en el estilo en cascada, enviar WM_MDICASCADE a la ventana del cliente.
WM_MDIDESTROY	Envía este mensaje a la ventana del cliente para destruir una ventana secundaria MDI. Usted debe utilizar este mensaje en lugar de llamar **DestroyWindow** porque si la ventana secundaria MDI está maxmized, este mensaje será restaurar el mosaico de la ventana de marco. Si utiliza **DestroyWindow,** el título de la ventana de marco no se restaurará.
WM_MDIGETACTIVE	Envía este mensaje a recuperar el identificador de la ventana secundaria MDI activa.
WM_MDIMAXIMIZE **WM_MDIRESTORE**	Enviar WM_MDIMAXIMIZE para maximizar la ventana secundaria MDI y WM_MDIRESTORE para restaurarlo al estado anterior. Siempre utilizar estos mensajes para las operaciones. Si utiliza ShowWindow con SW_MAXIMIZE, la ventana secundaria MDI maximizará bien, pero tendrá el problema al intentar restaurarlo a su tamaño anterior. Usted puede minimizar la ventana secundaria MDI con ShowWindow sin ningún problema, sin embargo.
WM_MDINEXT	Envía este mensaje a la ventana del cliente para activar la siguiente o la ventana secundaria MDI anterior, según los valores en wParam y lParam.
WM_MDIREFRESHMENU	Envía este mensaje a la ventana del cliente para actualizar el menú de la ventana de marco. Tenga en cuenta que debe llamar a **DrawMenuBar** para actualizar la barra de menú después de enviar este mensaje.
WM_MDISETMENU	Envía este mensaje a la ventana del cliente para reemplazar todo el menú de la ventana de marco o simplemente en el submenú de la ventana. Debe utilizar este mensaje en lugar de **SetMenu.** Después de enviar este mensaje, debe llamar a **DrawMenuBar** para actualizar la barra de menú. Normalmente se utiliza este mensaje cuando la ventana secundaria MDI activa tiene su propio menú y quiere que para reemplazar el menú de la ventana de marco, mientras que la ventana secundaria MDI está activo.

Voy a revisar los pasos para crear una aplicación MDI para usted una vez más a continuación.

1. Registrar las clases de ventana, tanto en el marco de la ventana de clase y la clase de la ventana secundaria MDI
2. Crear el marco de la ventana con **CreateWindowEx.**
3. Dentro del bucle de mensajes, llame **TranslateMDISysAccel** para procesar las teclas de aceleración MDI relacionados
4. En el procedimiento de ventana de la ventana de marco, llame **DefFrameProc** para manejar **todos los** mensajes no controladas por el código.
5. Crear la ventana del cliente llamando **CreateWindowEx** con el nombre de la clase de ventana predefinida, "MDICLIENT", pasando la dirección de una estructura CLIENTCREATESTRUCT

en lParam. Normalmente, se crea la ventana del cliente en el controlador WM_CREATE de la proc marco de la ventana

6. Puede crear una ventana secundaria MDI mediante el envío de WM_MDICREATE a la ventana del cliente o, alternativamente, llamando **CreateMDIWindow.**
7. Dentro de la ventana de procesos de la ventana secundaria MDI, pasar todos los mensajes no controladas **DefMDIChildProc.**
8. Utilice la versión de los mensajes de inhaladores de dosis medidas, si existe. Por ejemplo, utilice WM_MDIDESTROY en lugar de llamar **DestroyWindow**

Ejemplo:

```
  0.386
  . Modelo plano, stdcall
  casemap opción: ninguno
  include \ masm32 \ include \ windows.inc
  include \ masm32 \ include \ user32.inc
 include \ masm32 \ include \ kernel32.inc
 includelib \ masm32 \ lib \ user32.lib
 includelib \ masm32 \ lib \ kernel32.lib
 WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

 Const.
 IDR_MAINMENU equ 101
 IDR_CHILDMENU equ 102
 IDM_EXIT equ 40001
 IDM_TILEHORZ equ 40002
 IDM_TILEVERT equ 40003
 IDM_CASCADE equ 40004
 IDM_NEW equ 40005
 IDM_CLOSE equ 40006

 . Datos
 ClassName db "MDIASMClass", 0
 MDIClientName db "MDICLIENT", 0
 MDIChildClassName db "Win32asmMDIChild", 0
 MDIChildTitle db "MDI secundarios", 0
 AppName db "Win32asm MDI Demo", 0
 ClosePromptMessage db "¿Está seguro que desea cerrar esta ventana?",
0

 . Datos?
 hInstance dd?
 hMainMenu dd?
 hwndClient dd?
 hChildMenu dd?
 mdicreate MDICREATESTRUCT <>
 hwndFrame dd?

 . Código
 empezar:
         invocar GetModuleHandle, NULL
         mov hInstance, eax
         WinMain invoca, hInstance, NULL, NULL, SW_SHOWDEFAULT
         invocar ExitProcess, eax

 WinMain proc hInst: HINSTANCE, hPrevInst: HINSTANCE, CmdLine: LPSTR,
CmdShow: DWORD
         LOCAL wc: WNDCLASSEX
         Msg LOCAL: MSG
         ; ========================================
```

```
        ; Registrar la clase de ventana de marco
        ; ========================================
        mov wc.cbSize, sizeof WNDCLASSEX
        mov wc.style, CS_HREDRAW o CS_VREDRAW
        mov wc.lpfnWndProc, OFFSET WndProc
        mov wc.cbClsExtra, NULL
        mov wc.cbWndExtra, NULL
        impulsar hInstance
        pop wc.hInstance
        mov wc.hbrBackground, COLOR_APPWORKSPACE
        mov wc.lpszMenuName, IDR_MAINMENU
        wc.lpszClassName mov, ClassName OFFSET
        invocar LoadIcon, NULL, IDI_APPLICATION
        mov wc.hIcon, eax
        mov wc.hIconSm, eax
        invocar LoadCursor, NULL, IDC_ARROW
        wc.hCursor mov, eax
        invocar RegisterClassEx, addr wc
        ; ============================================
        ; Registro de la clase de ventana secundaria MDI
        ; ============================================
        mov wc.lpfnWndProc, desplazamiento ChildProc
        mov wc.hbrBackground, COLOR_WINDOW un
        wc.lpszClassName mov, MDIChildClassName desplazamiento
        invocar RegisterClassEx, addr wc
        invocar CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
                        WS_OVERLAPPEDWINDOW o WS_CLIPCHILDREN,
CW_USEDEFAULT, \
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
NULL, 0, \
                        hInst, NULL
        mov hwndFrame, eax
 invocar LoadMenu, hInstance, IDR_CHILDMENU
         mov hChildMenu, eax
        invocar ShowWindow, hwndFrame, SW_SHOWNORMAL
        invocar UpdateWindow, hwndFrame
        . Mientras VERDADERO
                invocar GetMessage, msg ADDR, NULL, 0,0
                . Descanso. If (! Eax)
                invocar TranslateMDISysAccel, hwndClient, addr msg
                . Si! Eax
                        invocar TranslateMessage, ADDR msg
                        invocar DispatchMessage, ADDR msg
                . Endif
        . Endw
        invocar DestroyMenu, hChildMenu
        mov eax, msg.wParam
        ret
 WinMain endp

 WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
        LOCAL ClientStruct: CLIENTCREATESTRUCT
        . Si uMsg == WM_CREATE
                invocar GetMenu, hWnd
                mov hMainMenu, eax
                invocar GetSubMenu, hMainMenu, 1
                mov ClientStruct.hWindowMenu, eax
                mov ClientStruct.idFirstChild, 100
                INVOKE CreateWindowEx, NULL, MDIClientName ADDR, NULL,
\
```

```
                                WS_CHILD o WS_VISIBLE o
WS_CLIPCHILDREN, CW_USEDEFAULT, \
                                CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, hWnd, NULL, \
                                hInstance, addr ClientStruct
                mov hwndClient, eax
                ; ====================================
                ; Inicializar el MDICREATESTRUCT
                ; ====================================
                mov mdicreate.szClass, MDIChildClassName
desplazamiento
                mov mdicreate.szTitle, desplazamiento MDIChildTitle
                impulsar hInstance
                pop mdicreate.hOwner
                mov mdicreate.x, CW_USEDEFAULT
                mov mdicreate.y, CW_USEDEFAULT
                mov mdicreate.lx, CW_USEDEFAULT
                mov mdicreate.ly, CW_USEDEFAULT
        . Elseif uMsg == WM_COMMAND
                . Si lParam == 0
                        mov eax, wParam
                        . Si ax == IDM_EXIT
                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                        . Elseif ax == IDM_TILEHORZ
                                invoca SendMessage, hwndClient,
WM_MDITILE, MDITILE_HORIZONTAL, 0
                        . Elseif ax == IDM_TILEVERT
                                invoca SendMessage, hwndClient,
WM_MDITILE, MDITILE_VERTICAL, 0
                        . Elseif ax == IDM_CASCADE
                                invoca SendMessage, hwndClient,
WM_MDICASCADE, MDITILE_SKIPDISABLED, 0
                        . Elseif ax == IDM_NEW
                                invoca SendMessage, hwndClient,
WM_MDICREATE, 0, addr mdicreate
                        . Elseif ax == IDM_CLOSE
                                invoca SendMessage, hwndClient,
WM_MDIGETACTIVE, 0,0
                                invoca SendMessage, eax, WM_CLOSE, 0,0
                        . Más
                                invocar DefFrameProc, hWnd, hwndClient,
uMsg, wParam, lParam
                                ret
                        . Endif
                . Endif
        . Elseif uMsg == WM_DESTROY
                invocar PostQuitMessage, NULL
        . Más
                invocar DefFrameProc, hWnd, hwndClient, uMsg, wParam,
lParam
                ret
        . Endif
        xor eax, eax
        ret
 WndProc ENDP

 ChildProc proc hChild: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
        . Si uMsg == WM_MDIACTIVATE
                mov eax, lParam
                . Si eax == hChild
```

```
                        invocar GetSubMenu, hChildMenu, 1
                        mov edx, eax
                        invoca SendMessage, hwndClient, WM_MDISETMENU,
hChildMenu, edx
                . Más
                        invocar GetSubMenu, hMainMenu, 1
                        mov edx, eax
                        invoca SendMessage, hwndClient, WM_MDISETMENU,
hMainMenu, edx
                . Endif
                invocar DrawMenuBar, hwndFrame
        . Elseif uMsg == WM_CLOSE
                invocar el cuadro de mensajes, hChild,
ClosePromptMessage addr, addr AppName, MB_YESNO
                . Si eax == IDYES
                        invoca SendMessage, hwndClient, WM_MDIDESTROY,
hChild, 0
                . Endif
        . Más
                invocar DefMDIChildProc, hChild, uMsg, wParam, lParam
                ret
        . Endif
        xor eax, eax
        ret
 ChildProc endp
 poner fin a empezar a
```

Análisis:

Lo primero que hace el programa es registrar las clases de ventana de la ventana de marco y la ventana secundaria MDI. Después de eso, llama a CreateWindowEx para crear el marco de la ventana. Dentro del controlador WM_CREATE de la ventana de marco, creamos la ventana del cliente:

```
        LOCAL ClientStruct: CLIENTCREATESTRUCT
        . Si uMsg == WM_CREATE
                invocar GetMenu, hWnd
                mov hMainMenu, eax
                invocar GetSubMenu, hMainMenu, 1
 mov ClientStruct.hWindowMenu, eax
                 mov ClientStruct.idFirstChild, 100
                invocar CreateWindowEx, NULL, MDIClientName ADDR,
NULL, \
                        WS_CHILD o WS_VISIBLE o WS_CLIPCHILDREN,
CW_USEDEFAULT, \
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
hWnd, NULL, \
                        hInstance, addr ClientStruct
                mov hwndClient, eax
```

Hace un llamamiento **GetMenu** para obtener el identificador del menú de la ventana de marco, que se utiliza en la llamada **GetSubMenu.** Tenga en cuenta que se pasa el valor 1 a **GetSubMenu** debido a que el submenú que desea que la lista de la ventana que aparece es el segundo submenú. Luego llenamos los miembros de la estructura CLIENTCREATESTRUCT.

A continuación, inicializar la estructura MDICLIENTSTRUCT. Tenga en cuenta que no necesitamos hacerlo aquí. Sólo es conveniente hacerlo en WM_CREATE.

```
  mov mdicreate.szClass, MDIChildClassName desplazamiento
        mov mdicreate.szTitle, desplazamiento MDIChildTitle
        impulsar hInstance
        pop mdicreate.hOwner
```

```
        mov mdicreate.x, CW_USEDEFAULT
        mov mdicreate.y, CW_USEDEFAULT
        mov mdicreate.lx, CW_USEDEFAULT
        mov mdicreate.ly, CW_USEDEFAULT
```

Después de que el marco de la ventana se crea (y también la ventana del cliente), que llamamos **LoadMenu** para cargar el menú de la ventana hijo a partir del recurso. Tenemos que conseguir manejar este menú para que podamos sustituir el menú de la ventana de marco con él cuando una ventana secundaria MDI está presente. No te olvides de llamar a **DestroyMenu** en el mango antes de salir de la aplicación para Windows. Normalmente Windows liberará el menú asociado con una ventana de forma automática cuando se cierra la aplicación, pero en este caso, el menú de la ventana niño no está asociada a ninguna ventana por lo que todavía ocupan la valiosa memoria incluso después de que se cierra la aplicación.

```
  invocar LoadMenu, hInstance, IDR_CHILDMENU
        mov hChildMenu, eax
        ........
        invocar DestroyMenu, hChildMenu
```

Dentro del bucle de mensajes, que llamamos **TranslateMDISysAccel.**

```
  . Mientras VERDADERO
                invocar GetMessage, msg ADDR, NULL, 0,0
                . Descanso. If (! Eax)
                invocar TranslateMDISysAccel, hwndClient, addr msg
                . Si! Eax
                        invocar TranslateMessage, ADDR msg
                        invocar DispatchMessage, ADDR msg
                . Endif
        . Endw
```

Si **TranslateMDISysAccel** devuelve un valor distinto de cero, significa que el mensaje fue manejado ya por el propio Windows para que usted no necesita hacer nada en el mensaje. Si devuelve 0, el mensaje no es MDI-relacionada y por lo tanto debe ser manejado como de costumbre.

```
  WndProc proc hWnd: HWND, uMsg: UINT, wParam: wParam, lParam: LPARAM
        .....
        . Más
 invocar DefFrameProc, hWnd, hwndClient, uMsg, wParam, lParam
                 ret
        . Endif
        xor eax, eax
        ret
 WndProc ENDP
```

Tenga en cuenta que en el procedimiento de ventana de la ventana de marco, llamamos a DefFrameProc para manejar los mensajes que no se interese

La mayor parte del procedimiento de ventana es el manejador WM_COMMAND. Cuando el usuario selecciona "Nuevo" en el menú Archivo, podemos crear una ventana secundaria MDI nuevo.

```
  . Elseif ax == IDM_NEW
                invoca SendMessage, hwndClient, WM_MDICREATE, 0, addr
mdicreate
```

En nuestro ejemplo, creamos la ventana secundaria MDI mediante el envío de WM_MDICREATE a la ventana del cliente, pasando por la dirección de la estructura MDICREATESTRUCT en lParam.

```
  ChildProc proc hChild: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
         . Si uMsg == WM_MDIACTIVATE
                mov eax, lParam
                . Si eax == hChild
                        invocar GetSubMenu, hChildMenu, 1
                        mov edx, eax
                        invoca SendMessage, hwndClient, WM_MDISETMENU,
hChildMenu, edx
                . Más
                        invocar GetSubMenu, hMainMenu, 1
                        mov edx, eax
                        invoca SendMessage, hwndClient, WM_MDISETMENU,
hMainMenu, edx
                . Endif
                invocar DrawMenuBar, hwndFrame
```

Cuando la ventana secundaria MDI se crea, supervisa WM_MDIACTIVATE para ver si es la ventana activa. Esto se hace comparando el valor de la lParam que contiene el identificador de la ventana activa niño con su propio mango. Si coinciden, es la ventana activa y el siguiente paso consiste en sustituir el menú de la ventana de marco a la suya. Desde el menú original será reemplazado, tienes que decirle a Windows de nuevo en el que al submenú de la lista de ventanas que aparecen. Es por eso que debemos llamar **GetSubMenu** nuevo para recuperar el identificador al submenú. Enviamos mensaje WM_MDISETMENU a la ventana del cliente para lograr el resultado deseado. wParam de WM_MDISETMENU contiene el identificador del menú que desea reemplazar el menú original. lParam contiene el identificador del submenú que desea la lista de ventanas que aparecen. Inmediatamente después de enviar WM_MDISETMENU, que llamamos **DrawMenuBar** para actualizar el menú de otra cosa que su menú será un desastre.

```
  . Más
                invocar DefMDIChildProc, hChild, uMsg, wParam, lParam
                ret
         . Endif
```

En el procedimiento de ventana de la ventana secundaria MDI, se debe pasar todos los mensajes no controladas **DefMDIChildProc** lugar de **DefWindowProc.**

```
  . Elseif ax == IDM_TILEHORZ
                invoca SendMessage, hwndClient, WM_MDITILE,
MDITILE_HORIZONTAL, 0
         . Elseif ax == IDM_TILEVERT
                invoca SendMessage, hwndClient, WM_MDITILE,
MDITILE_VERTICAL, 0
         . Elseif ax == IDM_CASCADE
                invoca SendMessage, hwndClient, WM_MDICASCADE,
MDITILE_SKIPDISABLED, 0
```

Cuando el usuario selecciona uno de los objetos MenuItem en el submenú ventana, enviar el mensaje correspondiente a la ventana del cliente. Si el usuario opta por las ventanas en mosaico, enviamos WM_MDITILE a la ventana del cliente, especificando en wParam qué tipo de mosaico que queremos. WM_CASCADE es similar.

```
  . Elseif ax == IDM_CLOSE
                invoca SendMessage, hwndClient, WM_MDIGETACTIVE, 0,0
                invoca SendMessage, eax, WM_CLOSE, 0,0
```

Si el usuario elige "Cerrar" menuitem, debemos obtener el handle de la ventana secundaria MDI activa por primera vez por el envío de WM_MDIGETACTIVE a la ventana del cliente. El valor de retorno en eax es el handle de la ventana secundaria MDI activa. Después de eso, enviamos WM_CLOSE a la ventana.

```
  . Elseif uMsg == WM_CLOSE
                invocar el cuadro de mensajes, hChild,
ClosePromptMessage addr, addr AppName, MB_YESNO
```

```
                . Si eax == IDYES
                        invoca SendMessage, hwndClient, WM_MDIDESTROY,
hChild, 0
                . Endif
```

Within the window procedure of the MDI child, when WM_CLOSE is received, it displays a message box asking the user if he really wants to close the window. En el procedimiento de ventana del MDI, cuando WM_CLOSE se recibe, se muestra un cuadro de mensaje preguntando al usuario si realmente desea cerrar la ventana. If the answer is yes, we send WM_MDIDESTROY to the client window. Si la respuesta es sí, enviamos WM_MDIDESTROY a la ventana del cliente. WM_MDIDESTROY closes the MDI child window and restores the title of the frame window. WM_MDIDESTROY cierra la ventana secundaria MDI y restaura el título de la ventana

TUTORIAL 33: EL CONTROL RICHEDIT: CONCEPTOS BÁSICOS

Hay un montón de tutoriales sobre la solicitud de los controles de RichEdit. Finalmente he jugado con él lo suficiente como para pensar que puedo escribir tutoriales al respecto. Así que aquí está: el primer tutorial de RichEdit. Los tutoriales se describe casi todo lo que hay que saber sobre el control RichEdit o por lo menos tanto como yo lo sé. La cantidad de información es bastante grande, así que lo dividimos en varias partes, este tutorial es la primera parte. En este tutorial, usted aprenderá lo que es un control RichEdit es, cómo crear y cómo cargar / guardar datos desde / hacia él.

TEORÍA

Un control RichEdit puede considerarse como un control trucado edición. Ofrece muchas características deseables de los que carecen del control de edición simple y sencillamente, por ejemplo, la capacidad de utilizar el tipo de fuente múltiple / tamaño, múltiples niveles de deshacer / rehacer, de búsqueda de la operación de texto, objetos incrustados OLE de arrastrar y soltar la compatibilidad de edición, etc Dado que el control RichEdit tiene tantas características, se almacena en un archivo DLL independiente. Esto también significa que, para usarla, no puedes llamar a **InitCommonControls** al igual que otros controles comunes. Usted tiene que llamar a **LoadLibrary** para cargar la DLL RichEdit.

El problema es que hay tres versiones de control RichEdit hasta ahora. Versión 1,2, y 3. La siguiente tabla muestra el nombre de la DLL para cada versión.

Nombre DLL	RichEdit versión	RichEdit Nombre de clase
Riched32.dll	1.0	RichEdit
Riched20.dll	2.0	RICHEDIT20A
Riched20.dll	3.0	RICHEDIT20A

Usted puede notar que la versión richedit 2 y 3 de utilizar el mismo nombre de DLL. También utilizan el mismo nombre de clase! Esto puede suponer un problema si usted desea utilizar las características específicas de RichEdit 3.0. Hasta ahora, no he encontrado un método oficial de diferenciar entre la versión 2.0 y 3.0. Sin embargo, hay una solución que funciona bien, te voy a mostrar más adelante.

```
. Datos
RichEditDLL db "Riched20.dll", 0
.....
. Datos?
hRichEditDLL dd?
. Código
invocar LoadLibrary, addr RichEditDLL
mov hRichEditDLL, eax
......
invocar FreeLibrary, hRichEditDLL
```

Cuando el archivo DLL se carga richedit, se registra la clase de ventana RichEdit. Por lo tanto, es imprescindible que usted cargar el archivo DLL antes de crear el control. Los nombres de las clases de control RichEdit son también diferentes. Ahora usted puede tener una pregunta: ¿cómo puedo saber qué versión de control RichEdit debo usar? Con la última versión no siempre es adecuada si no se requieren las características adicionales. Así que a continuación es la tabla que muestra las características proporcionadas por cada versión de control RichEdit.

Característica	Versión 1.0	La versión 2.0	La versión 3.0
la barra de selección	x	x	x
unicode edición		x	x
carácter / formato de párrafo	x	x	x
búsqueda de texto	adelante	adelante / atrás	adelante / atrás
Incrustación OLE	x	x	x
Arrastrar y soltar de edición	x	x	x
Deshacer / Rehacer	de un solo nivel	multi-nivel	multi-nivel
reconocimiento automático de URL		x	x
Acelerador de soporte clave		x	x
Operación de ventanas		x	x
Salto de línea	CRLF	CR sólo	CR sólo (puede emular la versión 1.0)
Ampliar			x
Numeración de los párrafos			x

tabla simple			x
estilos normales y la partida			x
subrayan colorante			x
texto oculto			x
la fuente de unión			x

La tabla anterior no es en absoluto exhaustiva: sólo una lista de las características más importantes.

Crear el control RichEdit

Después de cargar el archivo DLL de RichEdit, puede llamar a **CreateWindowEx** para crear el control. Puede utilizar los estilos de edición de control y estilos comunes de la ventana en **CreateWindowEx** excepto **ES_LOWERCASE, ES_UPPERCASE y ES_OEMCONVERT.**

```
Const.
        RichEditID equ 300
. Datos
        RichEditDLL db "Riched20.dll", 0
RichEditClass db "RichEdit20A", 0
        ...
. Datos?
hRichEditDLL dd?
        hwndRichEdit dd?
. Código
        .....
        invocar LoadLibrary, addr RichEditDLL
        mov hRichEditDLL, eax
invocar CreateWindowEx, 0, addr RichEditClass, WS_VISIBLE o ES_MULTILINE o
WS_CHILD o WS_VSCROLL o WS_HSCROLL, \
                                                                CW_USE
        mov hwndRichEdit, eax
```

Configuración por defecto de texto y color de fondo

Usted puede tener el problema de la configuración de color de texto y el color de fondo del control de edición. Sin embargo, este problema ha sido recurso en el control RichEdit. Para establecer el color de fondo del control RichEdit, le enviaremos **EM_SETBKGNDCOLOR** al control RichEdit. Este mensaje tiene la siguiente sintaxis.

wParam == opción de color. El valor de 0 en este parámetro especifica que Windows utiliza el valor del color en **lParam** como el color de fondo. Si este valor es distinto de cero, Windows utiliza el sistema operativo Windows el color de fondo. Desde que enviamos este mensaje para cambiar el color de fondo, tenemos que pasar 0 en wParam.
lParam == especifica la estructura **COLORREF** del color que desee establecer si wParam es 0.

Por ejemplo, si desea establecer el color de fondo azul puro, que emitirá esta línea siguiente:

invoca SendMessage, hwndRichEdit, EM_SETBKGNDCOLOR, 0,0 FF0000h

Para establecer el color del texto, el control RichEdit proporciona otro nuevo mensaje, **EM_SETCHARFORMAT,** para el trabajo. Este mensaje controla el formato de texto de una gama de caracteres en la selección o todo el texto en el control. Este mensaje tiene la siguiente sintaxis:

wParam == opciones de formato:

SCF_ALL	La operación afecta a todo el texto en el control.
SCF_SELECTION	La operación afecta a sólo el texto en la selección
SCF_WORD o SCF_SELECTION	Afecta a la palabra en la selección. Si la selección es empy, es decir, sólo el símbolo de intercalación se encuentra en la palabra, la operación afecta a esa palabra. **SCF_WORD** bandera debe ser utilizado con **SCF_SELECTION.**

lParam == puntero a una estructura o **CHARFORMAT CHARFORMAT2** que especifica el formato de texto que debe aplicarse. **CHARFORMAT2** está disponible para richedit 2,0 y superiores. Esto no significa que usted debe utilizar **CHARFORMAT2** con RichEdit 2.0 o superior. Puede seguir utilizando **CHARFORMAT** si las características añadidas en **CHARFORMAT2** no son necesarios para su necesidad.

```
CHARFORMATA STRUCT
        cbSize DWORD?
        dwMask DWORD?
        dwEffects DWORD?
        yHeight DWORD?
        yOffset DWORD?
        crTextColor COLORREF?
        BYTE bCharSet?
        BYTE bPitchAndFamily?
        szFaceName BYTE LF_FACESIZE dup (?)
        _wPad2 PALABRA?
CHARFORMATA TERMINA
```

<table>
<tr><th>Nombre del campo</th><th>Descripción</th></tr>
<tr><td>cbSize</td><td>El tamaño de la estructura. El control RichEdit utiliza este campo para determinar la versión de la estructura si es CHARFORMAT o CHARFORMAT2</td></tr>
<tr><td>dwMask</td><td>Marcadores de bits que determinan cuál de los siguientes miembros son válidos.
<table>
<tr><td>CFM_BOLD</td><td>El valor CFE_BOLD del miembro dwEffects es válido</td></tr>
<tr><td>CFM_CHARSET</td><td>El miembro bCharSet es válida.</td></tr>
<tr><td>CFM_COLOR</td><td>El miembro crTextColor y el valor CFE_AUTOCOLOR del miembro dwEffects son</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>válidos</td></tr>
<tr><td>CFM_FACE</td><td>El miembro szFaceName es válida.</td></tr>
<tr><td>CFM_ITALIC</td><td>El valor CFE_ITALIC del miembro dwEffects es válido</td></tr>
<tr><td>CFM_OFFSET</td><td>El miembro yOffset es válido</td></tr>
<tr><td>CFM_PROTECTED</td><td>El valor CFE_PROTECTED del miembro dwEffects es válido</td></tr>
<tr><td>CFM_SIZE</td><td>El miembro yHeight es válido</td></tr>
<tr><td>CFM_STRIKEOUT</td><td>El valor CFE_STRIKEOUT del miembro dwEffects es válida.</td></tr>
<tr><td>CFM_UNDERLINE</td><td>El valor CFE_UNDERLINE del miembro dwEffects es válido</td></tr>
</table>
</td></tr>
<tr><td>dwEffects</td><td>Los efectos de los personajes. Puede ser la combinación de los siguientes valores
<table>
<tr><td>CFE_AUTOCOLOR</td><td>Utilice el color del texto del sistema</td></tr>
<tr><td>CFE_BOLD</td><td>Los personajes están en negrita</td></tr>
<tr><td>CFE_ITALIC</td><td>Los personajes son cursiva</td></tr>
<tr><td>CFE_STRIKEOUT</td><td>Los personajes son golpeados.</td></tr>
<tr><td>CFE_UNDERLINE</td><td>Los personajes están subrayados</td></tr>
<tr><td>CFE_PROTECTED</td><td>Los personajes están protegidos, un intento de modificar causará un mensaje de notificación EN_PROTECTED.</td></tr>
</table>
</td></tr>
<tr><td>yHeight</td><td>Altura de los caracteres, en twips (1/1440 de una pulgada o 1/20 de punto de una impresora).</td></tr>
<tr><td>yOffset</td><td>Desplazamiento de caracteres, en twips, desde la línea de base. Si el valor de este miembro es positivo, el personaje es un exponente, y si es negativa, el personaje es un subíndice.</td></tr>
</table>

crTextColor	Color del texto. Este miembro se ignora si el efecto de caracteres CFE_AUTOCOLOR se especifica.
bCharSet	Carácter valor ajustado
bPitchAndFamily	Fuentes de la familia y el terreno de juego.
szFaceName	Terminada en nulo con carácter de matriz que especifica el nombre de la fuente
_wPad2	Relleno

Del examen de la estructura, verás que podemos cambiar los efectos de texto (negrita, cursiva tachado, subrayado), color de texto **(crTextColor)** y tipo de fuente / tamaño / juego de caracteres. Una bandera notable es **CFE_RPOTECTED.** El texto con este indicador está marcado como protegido lo que significa que cuando el usuario intenta modificar, mensaje de notificación **EN_PROTECTED** se enviará a la ventana padre. Y usted puede permitir que el cambio ocurra o no.

CHARFORMAT2 añade más estilos de texto, como el peso de la fuente, el espaciado, el color de fondo del texto, espaciado, etc Si usted no necesita estas características adicionales, basta con utilizar **CHARFORMAT.**

Para establecer el formato de texto, tienes que pensar en el rango de texto que desea aplicar. El control RichEdit introduce el concepto de rango de caracteres de texto. El control RichEdit le da a cada personaje un número inicial de 0: el primero characterin el control tiene el Número de Identificación de 0, el segundo carácter 1 y así sucesivamente. Para especificar un intervalo de texto, debe dar el control RichEdit dos números: los identificadores de la primera y el último carácter de la gama. Para aplicar el formato de texto con **EM_SETCHARFORMAT,** tiene un máximo de tres opciones:

1. Aplicar a todo el texto en el control **(SCF_ALL)**
2. Aplicar al texto en la actualidad en la selección **(SCF_SELECTION)**
3. Aplicar a toda la palabra en la actualidad en la selección (o **SCF_WORD SCF_SELECTION)**

Las primeras opciones y la segunda son sencillas. La última opción requiere una pequeña explicación. Si la selección actual sólo cubre uno o más de los caracteres de la palabra pero no la palabra completa, especificando la bandera de **SCF_WORD SCF_SELECTION** + se aplica el formato de texto con la palabra entera. Incluso si no hay ninguna selección actual, es decir, sólo el cursor está posicionado en la palabra, la tercera opción también se aplica el formato de texto a la palabra completa.

Para utilizar **EM_SETCHARFORMAT,** es necesario rellenar varios miembros de **CHARFORMAT** (o **CHARFORMAT2)** de la estructura. Por ejemplo, si queremos establecer el color del texto, vamos a llenar la estructura **CHARFORMAT** la siguiente manera:

```
. Datos?
        cf CHARFORMAT <>
....
. Código
        mov cf.cbSize, sizeof cf
        mov cf.dwMask, CFM_COLOR
```

```
        mov cf.crTextColor, 0FF0000h
        invoca SendMessage, hwndRichEdit, EM_SETCHARFORMAT, SCF_ALL,
addr cf
```

El siguiente fragmento de código anterior establece el color de texto del control RichEdit a azul puro. Tenga en cuenta que si no hay un texto en el control RichEdit **EM_SETCHARFORMAT** cuando es emitido, el texto escrito en el control RichEdit siguiendo el mensaje se utiliza el formato de texto especificado por el mensaje **EM_SETCHARFORMAT.**

Ajuste del texto / guardar el texto

Para aquellos de ustedes que se utilizan para el control de edición, seguramente estará familiarizado con **WM_GETTEXT** / **WM_SETTEXT** como el medio para establecer el texto / obtener el texto desde / hacia el control. Este método todavía funciona con el control RichEdit, pero no puede ser eficiente si el archivo es grande. Control de edición limita el texto que se puede introducir en ella a 64 KB, pero el control RichEdit puede aceptar el texto mucho más grande que eso. Sería muy engorroso para asignar un bloque muy grande de memoria (como 10 MB o así) para recibir el texto de **WM_GETTEXT.** El control RichEdit ofrece un nuevo enfoque a este método, es decir. la transmisión de texto.

En pocas palabras, le proporcionará la dirección de una función de devolución de llamada para el control RichEdit. Y el control RichEdit llamará a la función callback, pasando la dirección de la memoria intermedia a la misma, cuando esté listo. La devolución de llamada se llene el búfer con los datos que quiere enviar al control o leer los datos de la memoria intermedia y luego espera para la siguiente llamada hasta que la operación ha terminado. Este paradigma se utiliza para la transmisión de (establecer el texto) y que salían (obtener el texto del control). Vas a ver que este método es más eficaz: el buffer es proporcionado por el control RichEdit mismo modo que los datos se dividen en trozos. Las operaciones involucran a dos mensajes: **EM_STREAMIN** y **EM_STREAMOUT**

Tanto **EM_STREAMIN** y **EM_STREAMOUT** utilizan la misma sintaxis:

wParam == opciones de formato.

SF_RTF	Los datos están en el formato de texto enriquecido (RTF)
SF_TEXT	Los datos están en el formato de texto plano
SFF_PLAINRTF	Sólo las palabras clave más comunes a todas las lenguas se transmiten in
SFF_SELECTION	Si se especifica, el objetivo de la operación es el texto actualmente en la selección. Si escuchar el texto en el texto sustituye a la selección actual. Si usted transmitir el texto a cabo, sólo el texto en la actualidad en la selección se transmite a cabo. Si esta bandera no se especifica, la operación afecta a todo el texto en el control.
SF_UNICODE	(Disponible en el RichEdit 2.0 o posterior) Especifique el texto Unicode.

lParam == apuntan a una estructura de **EDITSTREAM** que tiene la siguiente definición:

```
EDITSTREAM STRUCT
        dwCookie DWORD?
        dwError DWORD?
```

pfnCallback DWORD?
EDITSTREAM TERMINA

dwCookie	definido por la aplicación de valor que se pasa a la función de devolución de llamada en el miembro speficied **pfnCallback** a continuación. Normalmente pasar algún valor importante para la función de llamada como el identificador de archivo para utilizar en el procedimiento stream-in/out.
dwError	Indica los resultados de la corriente en (lectura) o la secuencia fuera-(escritura) operación. Un valor de cero indica que no hay error. Un valor distinto de cero puede ser el valor de retorno de la función EditStreamCallback o un código que indica que el control ha detectado un error.
pfnCallback	Puntero a una función EditStreamCallback, que es una función definida por la aplicación de que el control de llamadas para transferir datos. El control llama a la función de devolución de llamada repetidamente, transfiriendo una parte de los datos con cada llamada

La función de callback editstream tiene la siguiente definición:

EditStreamCallback dwCookie proto: DWORD,
pBuffer: DWORD,
NumBytes: DWORD,
pBytesTransferred: DWORD

Tienes que crear una función con el prototipo anterior en su programa. Y a continuación, pasar su dirección a **EM_STREAMIN** o **EM_STREAMOUT** a través de la estructura **EDITSTREAM.**

Para la corriente de funcionamiento (settting el texto en el control RichEdit):

dwCookie: el valor definido por la aplicación se pasa a través de la estructura **EM_STREAMIN EDITSTREAM.** Casi siempre pasar el identificador de archivo del archivo que desea establecer su contenido con el control aquí.

pBuffer: apunta a la protección contemplada por el control RichEdit que va a recibir el texto de su función de callback.

NumBytes: el número máximo de bytes que se puede escribir el buffer de la (pBuffer) en la presente convocatoria. Usted **debe** obedecer siempre a este límite, es decir, puede enviar menos datos que el valor en numBytes pero no debe enviar más datos de este valor. Usted puede pensar en este valor como el tamaño del buffer en pBuffer.

pBytesTransferred: apunta a un DWORD que debe establecer el valor que indica el número de bytes realmente transferidos a la memoria intermedia. Este valor suele ser idéntico al valor de **numBytes.** La excepción es cuando los datos son para enviar es menor que el tamaño del tampón proporcionado por ejemplo, cuando al final del archivo que se alcanza.

Para el funcionamiento corriente de salida (obtener el texto del control RichEdit):

dwCookie: Igual que la corriente de operación de protección. Por lo general, pasan el identificador de archivo que desea escribir los datos en este parámetro.

pBuffer: apunta a la memoria intermedia proporcionada por el control RichEdit que se llena con los datos del control RichEdit.

Para obtener su tamaño, debe examinar el valor de **numBytes.**

NumBytes: el tamaño de los datos en el buffer apuntado por pBuffer.

pBytesTransferred: apunta a un DWORD que debe establecer el valor que indica el número de bytes que realmente leer de la memoria intermedia.

La función de devolución de llamada devuelve 0 para indicar el éxito y el control RichEdit seguirá llamando a la función de devolución de llamada si todavía quedan datos de lectura / escritura. Si se produce algún error durante el proceso y desea detener la operación, devuelve un valor distinto de cero y el control RichEdit, se descartarán los datos apuntados por pBuffer. El valor de error / éxito se llenará en el campo de la **dwError EDITSTREAM** para que pueda examinar el estado de error / éxito de la operación después del retorno de flujo de **SendMessage.**

Ejemplo:

El ejemplo siguiente es un sencillo editor que se puede abrir un archivo de código fuente asm, editarlo y guardarlo. Se utiliza el control RichEdit versión 2.0 o superior.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ gdi32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ gdi32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

Const.
IDR_MAINMENU equ 101
IDM_OPEN equ 40001
IDM_SAVE equ 40002
IDM_CLOSE equ 40003
IDM_SAVEAS equ 40004
IDM_EXIT equ 40005
IDM_COPY equ 40006
IDM_CUT equ 40007
IDM_PASTE equ 40008
IDM_DELETE equ 40009
IDM_SELECTALL equ 40010
IDM_OPTION equ 40011
IDM_UNDO equ 40012
IDM_REDO equ 40013
IDD_OPTIONDLG equ 101
IDC_BACKCOLORBOX equ 1000
IDC_TEXTCOLORBOX equ 1001

RichEditID equ 300

. Datos
```

```
  ClassName db "IczEditClass", 0
  AppName db "IczEdit versión 1.0", 0
  RichEditDLL db "riched20.dll", 0
  RichEditClass db "RichEdit20A", 0
  NoRichEdit db "No se puede encontrar riched20.dll", 0
  ASMFilterString db "El código fuente ASM (*. Asm)", 0, "*. Asm", 0
                                        db "Todos los archivos (*. *)", 0, "*. *", 0,0
  OpenFileFail db "No se puede abrir el archivo", 0
  WannaSave db "Los datos en el control se ha modificado. ¿Quieres guardarlo?", 0
  FileOpened dd FALSO
  BackgroundColor 0FFFFFFh dd, por defecto en blanco
  TextColor dd 0; por defecto a negro

  . Datos?
  hInstance dd?
  hRichEdit dd?
  hwndRichEdit dd?
  NombreDeArchivo db 256 dup (?)
  AlternateFileName db 256 dup (?)
  CustomColors dd 16 dup (?)

  . Código
  empezar:
            invocar GetModuleHandle, NULL
            mov hInstance, eax
  invocar LoadLibrary, addr RichEditDLL
            . Si eax! = 0
                      mov hRichEdit, eax
                      WinMain invoca, hInstance, 0,0, SW_SHOWDEFAULT
                      invocar FreeLibrary, hRichEdit
            . Más
                      invocar el cuadro de mensajes, 0, NoRichEdit addr, addr
AppName, MB_OK o MB_ICONERROR
            . Endif
            invocar ExitProcess, eax

  WinMain proc hInst: DWORD, hPrevInst: DWORD, CmdLine: DWORD, CmdShow:
DWORD
            LOCAL wc: WNDCLASSEX
            Msg LOCAL: MSG
            LOCAL hwnd: DWORD
            mov wc.cbSize, sizeof WNDCLASSEX
            mov wc.style, CS_HREDRAW o CS_VREDRAW
            mov wc.lpfnWndProc, OFFSET WndProc
            mov wc.cbClsExtra, NULL
            mov wc.cbWndExtra, NULL
            impulsar hInst
            pop wc.hInstance
            mov wc.hbrBackground, COLOR_WINDOW un
            mov wc.lpszMenuName, IDR_MAINMENU
            wc.lpszClassName mov, ClassName OFFSET
            invocar LoadIcon, NULL, IDI_APPLICATION
            mov wc.hIcon, eax
            mov wc.hIconSm, eax
            invocar LoadCursor, NULL, IDC_ARROW
            wc.hCursor mov, eax
            invocar RegisterClassEx, addr wc
            INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
```

```
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
        hInst, NULL
            mov hwnd, eax
            invocar ShowWindow, hwnd, SW_SHOWNORMAL
            invocar UpdateWindow, hwnd
            . Mientras VERDADERO
                    invocar GetMessage, ADDR msg, 0,0,0
                    . Descanso. If (! Eax)
                    invocar TranslateMessage, ADDR msg
                    invocar DispatchMessage, ADDR msg
            . Endw
            mov eax, msg.wParam
            ret
  WinMain endp

  StreamInProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesRead: DWORD
            invocar ReadFile, hFile, pBuffer, numBytes, pBytesRead, 0
            xor eax, 1
            ret
  StreamInProc endp

  StreamOutProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesWritten: DWORD
            invocar WriteFile, hFile, pBuffer, numBytes, pBytesWritten, 0
            xor eax, 1
            ret
  StreamOutProc endp

  CheckModifyState proc hWnd: DWORD
            invoca SendMessage, hwndRichEdit, EM_GETMODIFY, 0,0
            . Si eax! = 0
                    invocar el cuadro de mensajes, hWnd, WannaSave addr, addr
AppName, MB_YESNOCANCEL
                    . Si eax == IDYES
                            invoca SendMessage, hWnd, WM_COMMAND,
IDM_SAVE, 0
                    . Elseif eax == IDCANCEL
                            mov eax, FALSO
                            ret
                    . Endif
            . Endif
            mov eax, TRUE
            ret
  CheckModifyState endp

  SetColor proc
            LOCAL pies cúbicos por minuto: CHARFORMAT
            invoca SendMessage, hwndRichEdit, EM_SETBKGNDCOLOR, 0,
BackgroundColor
            invocar RtlZeroMemory, addr pies cúbicos por minuto, sizeof pies
cúbicos por minuto
            mov cfm.cbSize, sizeof pies cúbicos por minuto
            mov cfm.dwMask, CFM_COLOR
            impulsar TextColor
            pop cfm.crTextColor
```

```
            invoca SendMessage, hwndRichEdit, EM_SETCHARFORMAT, SCF_ALL,
addr pies cúbicos por minuto
            ret
   SetColor endp

   OptionProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
            LOCAL clr: ChooseColor
            . Si uMsg == WM_INITDIALOG
            . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                    mov eax, wParam
                                    . Si ax == IDCANCEL
                                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                    . Elseif ax == IDC_BACKCOLORBOX
                                                invocar RtlZeroMemory, addr clr, sizeof clr
                                                mov clr.lStructSize, sizeof clr
                                                impulsar hWnd
                                                clr.hwndOwner pop
                                                impulsar hInstance
                                                pop clr.hInstance
                                                impulsar BackgroundColor
                                                pop clr.rgbResult
                                                mov clr.lpCustColors, CustomColors de
compensación
                                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                                invocar ChooseColor, addr clr
                                                . Si eax! = 0
                                                            impulsar clr.rgbResult
                                                            pop BackgroundColor
                                                            invocar GetDlgItem, hWnd,
IDC_BACKCOLORBOX
                                                            invocar InvalidateRect, eax, 0, TRUE
                                                . Endif
                                    . Elseif ax == IDC_TEXTCOLORBOX
                                                invocar RtlZeroMemory, addr clr, sizeof clr
                                                mov clr.lStructSize, sizeof clr
                                                impulsar hWnd
                                                clr.hwndOwner pop
                                                impulsar hInstance
                                                pop clr.hInstance
                                                impulsar TextColor
                                                pop clr.rgbResult
                                                mov clr.lpCustColors, CustomColors de
compensación
                                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                                invocar ChooseColor, addr clr
                                                . Si eax! = 0
                                                            impulsar clr.rgbResult
                                                            pop TextColor
                                                            invocar GetDlgItem, hWnd,
IDC_TEXTCOLORBOX
                                                            invocar InvalidateRect, eax, 0, TRUE
                                                . Endif
                                    . Elseif ax == IDOK
```

```
                                        ;
=============================================================
================================
                                        ; Guardar el estado de modificación del
control RichEdit porque al cambiar el color del texto cambia el
                                        , Modificar el estado del control RichEdit.
                                        ;
=============================================================
================================
                                        invoca SendMessage, hwndRichEdit,
EM_GETMODIFY, 0,0
                                        push eax
                                        invocar SetColor
                                        pop eax
                                        invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, eax, 0
                                        invocar EndDialog, hWnd, 0
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CTLCOLORSTATIC
                        invocar GetDlgItem, hWnd, IDC_BACKCOLORBOX
                        . Si eax == lParam
                                invocar CreateSolidBrush, BackgroundColor
                                ret
                        . Más
                                invocar GetDlgItem, hWnd, IDC_TEXTCOLORBOX
                                . Si eax == lParam
                                        invocar CreateSolidBrush, TextColor
                                        ret
                                . Endif
                        . Endif
                        mov eax, FALSO
                        ret
                . Elseif uMsg == WM_CLOSE
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
        OptionProc endp

        WndProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
                LOCAL chrg: CHARRANGE
                LOCAL de n: OPENFILENAME
                Búfer local [256]: BYTE
                LOCAL editstream: EDITSTREAM
                LOCAL hFile: DWORD
                . Si uMsg == WM_CREATE
                        invocar CreateWindowEx, WS_EX_CLIENTEDGE, addr
RichEditClass, 0, o WS_CHILD WS_VISIBLE o ES_MULTILINE o WS_VSCROLL o
WS_HSCROLL o estilo ES_NOHIDESEL, \
                                        CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, hWnd, RichEditID, hInstance, 0
                        mov hwndRichEdit, eax
```

```
                ;
================================================
============
                , Establecer el límite de texto.    El valor predeterminado es de
64 K
                ;
================================================
============
                invoca SendMessage, hwndRichEdit, EM_LIMITTEXT, -1,0
                ;
================================================
============
                ; Definir el valor predeterminado de texto / color de fondo
                ;
================================================
============
                invocar SetColor
                invoca SendMessage, hwndRichEdit, EM_SETMODIFY, FALSE, 0
                invoca SendMessage, hwndRichEdit, EM_EMPTYUNDOBUFFER,
0,0
        . UMsg elseif == WM_INITMENUPOPUP
                mov eax, lParam
                . Si ax == 0; menú de archivo
                        . Si FileOpened == TRUE, un archivo ya está abierto
                                invocar EnableMenuItem, wParam,
IDM_OPEN, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_SAVE, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_ENABLED
                        . Más
                                invocar EnableMenuItem, wParam,
IDM_OPEN, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_SAVE, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_GRAYED
                        . Endif
                . Elseif ax == 1; menú de edición
                        ;
================================================
============================
                        , Compruebe si hay algún texto en el portapapeles.
Si es así, permitir que la pasta menuitem
                        ;
================================================
============================
                        invoca SendMessage, hwndRichEdit, EM_CANPASTE,
CF_TEXT, 0
                        . Si eax == 0; no hay texto en el portapapeles
                                invocar EnableMenuItem, wParam,
IDM_PASTE, MF_GRAYED
                        . Más
```

```
                                    invocar EnableMenuItem, wParam,
IDM_PASTE, MF_ENABLED
                            . Endif
                            ;
================================================
=========
                            , Comprobar si la cola de deshacer está vacía
                            ;
================================================
=========
                            invoca SendMessage, hwndRichEdit, EM_CANUNDO,
0,0
                            . Si eax == 0
                                    invocar EnableMenuItem, wParam,
IDM_UNDO, MF_GRAYED
                            . Más
                                    invocar EnableMenuItem, wParam,
IDM_UNDO, MF_ENABLED
                            . Endif
                            ;
================================================
========
                            , Comprobar si la cola rehecha está vacía
                            ;
================================================
========
                            invoca SendMessage, hwndRichEdit, EM_CANREDO,
0,0
                            . Si eax == 0
                                    invocar EnableMenuItem, wParam,
IDM_REDO, MF_GRAYED
                            . Más
                                    invocar EnableMenuItem, wParam,
IDM_REDO, MF_ENABLED
                            . Endif
                            ;
================================================
========
                            , Comprobar si hay una selección actual en el control
RichEdit.
                            , Si es así, que permitir que el cortar / copiar / borrar
menuitem
                            ;
================================================
========
                            invoca SendMessage, hwndRichEdit, EM_EXGETSEL, 0,
addr chrg
                            mov eax, chrg.cpMin
                            . Si eax == chrg.cpMax; ninguna selección actual
                                    invocar EnableMenuItem, wParam,
IDM_COPY, MF_GRAYED
                                    invocar EnableMenuItem, wParam,
IDM_CUT, MF_GRAYED
                                    invocar EnableMenuItem, wParam,
IDM_DELETE, MF_GRAYED
                            . Más
                                    invocar EnableMenuItem, wParam,
IDM_COPY, MF_ENABLED
```

```
                                        invocar EnableMenuItem, wParam,
IDM_CUT, MF_ENABLED
                                        invocar EnableMenuItem, wParam,
IDM_DELETE, MF_ENABLED
                                . Endif
                        . Endif
                . Elseif uMsg == WM_COMMAND
                        . Si lParam == 0; comandos de menú
                                mov eax, wParam
                                . Si ax == IDM_OPEN
                                        invocar RtlZeroMemory, addr de n, sizeof
OFN
                                        mov ofn.lStructSize, sizeof OFN
                                        impulsar hWnd
                                        ofn.hwndOwner pop
                                        impulsar hInstance
                                        pop ofn.hInstance
                                        mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                        mov ofn.lpstrFile, FileName desplazamiento
                                        mov byte ptr [nombreDeArchivo], 0
                                        mov ofn.nMaxFile, sizeof nombreDeArchivo
                                        mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                        invocar GetOpenFileName, addr OFN
                                        . Si eax! = 0
                                                invocar CreateFile, addr FileName
GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0
                                                . Si eax! =
INVALID_HANDLE_VALUE
                                                        mov hFile, eax
                                                        ;
================================================
===============
                                                        ; Escuchar el texto en el
control RichEdit
                                                        ;
================================================
===============
                                                        mov editstream.dwCookie,
eax
                                                        mov
editstream.pfnCallback, desplazamiento StreamInProc
                                                        invoca SendMessage,
hwndRichEdit, EM_STREAMIN, SF_TEXT, addr editstream
                                                        ;
================================================
=========
                                                        ; Inicializa el estado de
modificación a la falsa
                                                        ;
================================================
=========
                                                        invoca SendMessage,
hwndRichEdit, EM_SETMODIFY, FALSE, 0
                                                        invocar CloseHandle, hFile
                                                        mov FileOpened, TRUE
```

```
                                        . Más
                                                invocar el cuadro de
mensajes, hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                        . Endif
                                . Endif
                        . Elseif ax == IDM_CLOSE
                                invocar CheckModifyState, hWnd
                                . Si eax == TRUE
                                        invocar SetWindowText,
hwndRichEdit, 0
                                        mov FileOpened, FALSO
                                . Endif
                        . Elseif ax == IDM_SAVE
                                invocar CreateFile, addr FileName
GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0
                                . Si eax! = INVALID_HANDLE_VALUE
  @ @:
                                        mov hFile, eax
                                        ;
================================================
===============
                                        ; Transmitir el texto en el archivo
                                        ;
================================================
===============
                                        mov editstream.dwCookie, eax
                                        mov editstream.pfnCallback,
desplazamiento StreamOutProc
                                        invoca SendMessage, hwndRichEdit,
EM_STREAMOUT, SF_TEXT, addr editstream
                                        ;
================================================
=========
                                        ; Inicializa el estado de modificación
a la falsa
                                        ;
================================================
=========
                                        invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, FALSE, 0
                                        invocar CloseHandle, hFile
                                . Más
                                        invocar el cuadro de mensajes,
hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                . Endif
                        . Elseif ax == IDM_COPY
                                invoca SendMessage, hwndRichEdit,
WM_COPY, 0,0
                        . Elseif ax == IDM_CUT
                                invoca SendMessage, hwndRichEdit,
WM_CUT, 0,0
                        . Elseif ax == IDM_PASTE
                                invoca SendMessage, hwndRichEdit,
WM_PASTE, 0,0
                        . Elseif ax == IDM_DELETE
                                invoca SendMessage, hwndRichEdit,
EM_REPLACESEL, TRUE, 0
```

```
                        . Elseif ax == IDM_SELECTALL
                                mov chrg.cpMin, 0
                                mov chrg.cpMax, -1
                                invoca SendMessage, hwndRichEdit,
EM_EXSETSEL, 0, addr chrg
                        . Elseif ax == IDM_UNDO
                                invoca SendMessage, hwndRichEdit,
EM_UNDO, 0,0
                        . Elseif ax == IDM_REDO
                                invoca SendMessage, hwndRichEdit,
EM_REDO, 0,0
                        . Elseif ax == IDM_OPTION
                                invocar DialogBoxParam, hInstance,
IDD_OPTIONDLG, hWnd, addr OptionProc, 0
                        . Elseif ax == IDM_SAVEAS
                                invocar RtlZeroMemory, addr de n, sizeof
OFN
                                mov ofn.lStructSize, sizeof OFN
                                impulsar hWnd
                                ofn.hwndOwner pop
                                impulsar hInstance
                                pop ofn.hInstance
                                mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                mov ofn.lpstrFile, AlternateFileName
desplazamiento
                                mov byte ptr [AlternateFileName], 0
                                mov ofn.nMaxFile, sizeof AlternateFileName
                                mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                invocar GetSaveFileName, addr OFN
                                . Si eax! = 0
                                        invocar CreateFile,
AlternateFileName addr, GENERIC_WRITE, FILE_SHARE_READ, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
                                        . Si eax! =
INVALID_HANDLE_VALUE
                                                jmp @ B
                                        . Endif
                                . Endif
                        . Elseif ax == IDM_EXIT
                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                        . Endif
                . Endif
        . Elseif uMsg == WM_CLOSE
                invocar CheckModifyState, hWnd
                . Si eax == TRUE
                        invocar DestroyWindow, hWnd
                . Endif
        . Elseif uMsg == WM_SIZE
                mov eax, lParam
                mov edx, eax
                y EAX, 0FFFFh
                shr edx, 16
                invocar MoveWindow, hwndRichEdit, 0,0, eax, edx, TRUE
        . Elseif uMsg == WM_DESTROY
                invocar PostQuitMessage, NULL
        . Más
```

```
                    invocar DefWindowProc, hWnd, uMsg, wParam, lParam
                    ret
            . Endif
            xor eax, eax
            ret
  WndProc ENDP
  poner fin a empezar a

  ; ================================================
==================
  ; El archivo de recursos
  ; ================================================
==================
  # Include "resource.h"
  # Define IDR_MAINMENU 101
  # Define IDD_OPTIONDLG 101
  # Define IDC_BACKCOLORBOX 1000
  # Define IDC_TEXTCOLORBOX 1001
  # Define IDM_OPEN 40001
  # Define IDM_SAVE 40002
  # Define IDM_CLOSE 40003
  # Define IDM_SAVEAS 40004
  # Define IDM_EXIT 40005
  # Define IDM_COPY 40006
  # Define IDM_CUT 40007
  # Define IDM_PASTE 40008
  # Define IDM_DELETE 40009
  # Define IDM_SELECTALL 40010
  # Define IDM_OPTION 40011
  # Define IDM_UNDO 40012
  # Define IDM_REDO 40013

  IDR_MAINMENU DISCARDABLE MENÚ
  COMENZAR
    POPUP "& Archivo"
    COMENZAR
      MENUITEM "& Abrir", IDM_OPEN
      MENUITEM "& Cerrar", IDM_CLOSE
      MENUITEM "& Guardar", IDM_SAVE
      MENUITEM "Save & As", IDM_SAVEAS
      MENUITEM SEPARATOR
      MENUITEM "& Salir", IDM_EXIT
    FIN
    POPUP "& Editar"
    COMENZAR
      MENUITEM "& Deshacer", IDM_UNDO
      MENUITEM "& Rehacer", IDM_REDO
      MENUITEM "& Copiar", IDM_COPY
      MENUITEM "C & ut", IDM_CUT
      MENUITEM "& Pegar", IDM_PASTE
      MENUITEM SEPARATOR
      MENUITEM "& Eliminar", IDM_DELETE
      MENUITEM SEPARATOR
      MENUITEM "& Seleccionar todo", IDM_SELECTALL
    FIN
    MENUITEM "Opciones", IDM_OPTION
  FIN
```

```
   DIÁLOGO IDD_OPTIONDLG DISCARDABLE 0, 0, 183, de 54 años
   ESTILO DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU | DS_CENTER
   TITULO "Opciones"
   FONT 8, "MS Sans Serif"
   COMENZAR
      DEFPUSHBUTTON en "Aceptar", IDOK, 137,7,39,14
      PULSADOR "Cancelar", IDCANCEL, 137,25,39,14
      GroupBox "", IDC_STATIC, 5,0,124,49
      LTEXT "Color de fondo:" IDC_STATIC, 20,14,60,8
      LTEXT "", IDC_BACKCOLORBOX, 85,11,28,14, SS_NOTIFY | WS_BORDER
      LTEXT "Color del texto:", IDC_STATIC, 20,33,35,8
      LTEXT "", IDC_TEXTCOLORBOX, 85,29,28,14, SS_NOTIFY | WS_BORDER
   FIN
```

Análisis:

El primer programa carga el archivo DLL de RichEdit, que en este caso es riched20.dll. Si el archivo DLL no se pueden cargar, sale a Windows.

```
   invocar LoadLibrary, addr RichEditDLL
   . Si eax! = 0
            mov hRichEdit, eax
            WinMain invoca, hInstance, 0,0, SW_SHOWDEFAULT
            invocar FreeLibrary, hRichEdit
   . Más
            invocar el cuadro de mensajes, 0, NoRichEdit addr, addr AppName,
MB_OK o MB_ICONERROR
   . Endif
   invocar ExitProcess, eax
```

Después de que el archivo DLL se carga correctamente, se procede a crear una ventana normal, que será el padre del control RichEdit. Dentro del controlador **WM_CREATE,** creamos el control RichEdit:

```
   invocar CreateWindowEx, WS_EX_CLIENTEDGE, addr RichEditClass, 0, o
WS_CHILD WS_VISIBLE o ES_MULTILINE o WS_VSCROLL o WS_HSCROLL o estilo
ES_NOHIDESEL, \
                                        CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, hWnd, RichEditID, hInstance, 0
                     mov hwndRichEdit, eax
```

Tenga en cuenta que especificar el estilo de **ES_MULTILINE** de lo contrario el control será una sola-alineado.

```
   invoca SendMessage, hwndRichEdit, EM_LIMITTEXT, -1,0
```

Después de que el control RichEdit se crea, hay que establecer el límite de un nuevo texto sobre el mismo. De forma predeterminada, el control RichEdit tiene 64K límite de texto, lo mismo que una simple multi-control de edición. Debemos extender este límite para permitir que opere con archivos más grandes. En la línea anterior, que especifica -1 lo que equivale a 0FFFFFFFFh, un valor muy grande.

```
         invocar SetColor
```

A continuación, establecer el color del texto / fondo. Puesto que esta operación se puede realizar en otra parte del programa, pongo el código en una función denominada SetColor.

```
SetColor proc
        LOCAL pies cúbicos por minuto: CHARFORMAT
        invoca SendMessage, hwndRichEdit, EM_SETBKGNDCOLOR, 0,
BackgroundColor
```

Ajuste del color de fondo del control RichEdit es una operación sencilla: basta con enviar un mensaje **EM_SETBKGNDCOLOR** al control RichEdit. (Si utiliza un control multi-línea de edición, lo que tiene que procesar **WM_CTLCOLOREDIT).** El color de fondo predeterminado es blanco.

```
invocar RtlZeroMemory, addr pies cúbicos por minuto, sizeof pies cúbicos por
minuto
        mov cfm.cbSize, sizeof pies cúbicos por minuto
        mov cfm.dwMask, CFM_COLOR
        impulsar TextColor
        pop cfm.crTextColor
```

Después de que el color de fondo se establece, llenamos los miembros de **CHARFORMAT** con el fin de establecer el color del texto. Tenga en cuenta que llenamos **cbSize** con el tamaño de la estructura para el control RichEdit sabe que está enviando **CHARFORMAT** no **CHARFORMAT2. DwMask** tiene una sola bandera, **CFM_COLOR,** ya que sólo desea establecer el color del texto y **crTextColor** se llena con el valor de el texto de color deseada.

```
invoke SendMessage,hwndRichEdit,EM_SETCHARFORMAT,SCF_ALL,addr cfm
        ret
SetColor endp
```

After settting the color, you have to empty undo buffer simply because the act of changing text/background color is undo-able. We send **EM_EMPTYUNDOBUFFER** message to achieve this.

```
invoke SendMessage,hwndRichEdit,EM_EMPTYUNDOBUFFER,0,0
```

After filling the **CHARFORMAT** structure, we send **EM_SETCHARFORMAT** to the richedit control, specifying **SCF_ALL** flag in **wParam** to indicate that we want the text formatting to be applied to all text in the control.

Note that when we first created the richedit control, we didn't specify its size/position at that time. That's because we want it to cover the whole client area of the parent window. We resize it whenever the size of the parent window changes.

```
.elseif uMsg==WM_SIZE
                mov eax,lParam
                mov edx,eax
                and eax,0FFFFh
                shr edx,16
                invoke MoveWindow,hwndRichEdit,0,0,eax,edx,TRUE
```

In the above code snippet, we use the new dimension of the client area passed in **lParam** to resize the richedit control with **MoveWindow** .

When the user clicks on the File/Edit menu bar, we process **WM_INITPOPUPMENU** so that we can prepare the states of the menuitems in the submenu before displaying it to the user. For example, if a file is already opened in the richedit control, we want to disable the open menuitem and enable all the remaining menuitems.

In the case of the File menu bar, we use the variable **FileOpened** as the flag to determine whether a file is already opened. If the value in this variable is TRUE, we know that a file is already opened.

```
  .elseif uMsg==WM_INITMENUPOPUP
                    mov eax,lParam
                    .if ax==0 ; file menu
                              .if FileOpened==TRUE ; a file is already opened
                                        invoke
EnableMenuItem,wParam,IDM_OPEN,MF_GRAYED
                                        invoke
EnableMenuItem,wParam,IDM_CLOSE,MF_ENABLED
                                        invoke
EnableMenuItem,wParam,IDM_SAVE,MF_ENABLED
                                        invoke
EnableMenuItem,wParam,IDM_SAVEAS,MF_ENABLED
                              .else
                                        invoke
EnableMenuItem,wParam,IDM_OPEN,MF_ENABLED
                                        invoke
EnableMenuItem,wParam,IDM_CLOSE,MF_GRAYED
                                        invoke
EnableMenuItem,wParam,IDM_SAVE,MF_GRAYED
                                        invoke
EnableMenuItem,wParam,IDM_SAVEAS,MF_GRAYED
                              .endif
```

As you can see, if a file is already opened, we gray out the open menuitem and enable the remaining menuitems. The reverse is true of **FileOpened** is false.

In the case of the edit menu bar, we need to check the state of the richedit control/clipboard first.

```
  invoke SendMessage,hwndRichEdit,EM_CANPASTE,CF_TEXT,0
                              .if eax==0 ; no text in the clipboard
                                        invoke
EnableMenuItem,wParam,IDM_PASTE,MF_GRAYED
                              .else
                                        invoke
EnableMenuItem,wParam,IDM_PASTE,MF_ENABLED
                              .endif
```

We first check whether some text is available in the clipboard by sending **EM_CANPASTE** message. If some text is available, **SendMessage** returns TRUE and we enable the paste menuitem. If not, we gray out the menuitem.

```
  invoke SendMessage,hwndRichEdit,EM_CANUNDO,0,0
                              .if eax==0
                                        invoke
EnableMenuItem,wParam,IDM_UNDO,MF_GRAYED
                              .else
```

```
                    invoke
EnableMenuItem,wParam,IDM_UNDO,MF_ENABLED
                .endif
```

Next, we check whether the undo buffer is empty by sending **EM_CANUNDO** message. If it's not empty, **SendMessage** returns TRUE and we enable the undo menuitem.

```
  invoke SendMessage,hwndRichEdit,EM_CANREDO,0,0
                .if eax==0
                    invoke
EnableMenuItem,wParam,IDM_REDO,MF_GRAYED
                .else
                    invoke
EnableMenuItem,wParam,IDM_REDO,MF_ENABLED
                .endif
```

We check the redo buffer by sending **EM_CANREDO** message to the richedit control. If it's not empty, **SendMessage** returns TRUE and we enable the redo menuitem.

```
  invoke SendMessage,hwndRichEdit,EM_EXGETSEL,0,addr chrg
                mov eax,chrg.cpMin
                .if eax==chrg.cpMax ; no current selection
                    invoke
EnableMenuItem,wParam,IDM_COPY,MF_GRAYED
                    invoke
EnableMenuItem,wParam,IDM_CUT,MF_GRAYED
                    invoke
EnableMenuItem,wParam,IDM_DELETE,MF_GRAYED
                .else
                    invoke
EnableMenuItem,wParam,IDM_COPY,MF_ENABLED
                    invoke
EnableMenuItem,wParam,IDM_CUT,MF_ENABLED
                    invoke
EnableMenuItem,wParam,IDM_DELETE,MF_ENABLED
                .endif
```

Lastly, we check whether a current selection exists by sending **EM_EXGETSEL** message. This message uses a **CHARRANGE** structure which is defined as follows:

```
  CHARRANGE STRUCT
  cpMin DWORD ?
  cpMax DWORD ?
CHARRANGE ENDS
```

cpMin contains the character position index immediately preceding the first character in the range.
cpMax contains the character position immediately following the last character in the range.

After **EM_EXGETSEL** returns, the **CHARRANGE** structure is filled with the starting-ending character position indices of the selection range. If there is no current selection, **cpMin** and **cpMax** are identical and we gray out the cut/copy/delete menuitems.

When the user clicks the Open menuitem, we display an open file dialog box and if the user selects a file, we open the file and stream its content to the richedit control.

```
  invoke CreateFile,addr
FileName,GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIB
UTE_NORMAL,0
                                        .if eax!=INVALID_HANDLE_VALUE
                                            mov hFile,eax

                                            mov
editstream.dwCookie,eax
                                            mov
editstream.pfnCallback,offset StreamInProc
                                            invoke
SendMessage,hwndRichEdit,EM_STREAMIN,SF_TEXT,addr editstream
```

After the file is successfully opened with **CreateFile** , we fill the **EDITSTREAM** structure in preparation for **EM_STREAMIN** message. We choose to send the handle to the opened file via **dwCookie** member and pass the address of the stream callback function in **pfnCallback** .

The stream callback procedure itself is the essence of simplicity.

```
 StreamInProc proc hFile:DWORD,pBuffer:DWORD, NumBytes:DWORD,
pBytesRead:DWORD
        invoke ReadFile,hFile,pBuffer,NumBytes,pBytesRead,0
        xor eax,1
           ret
StreamInProc endp
```

You can see that all parameters of the stream callback procedure fit perfectly with **ReadFile** . And the return value of **ReadFile** is xor-ed with 1 so that if it returns 1 (success), the actual value returned in eax is 0 and vice versa.

```
   invoke SendMessage,hwndRichEdit,EM_SETMODIFY,FALSE,0
        invoke CloseHandle,hFile
        mov FileOpened,TRUE
```

After **EM_STREAMIN** returns, it means the stream operation is completed. In reality, we must check the value of **dwError** member of the **EDITSTREAM** structure.

Richedit (and edit) control supports a flag to indicate whether its content is modified. We can obtain the value of this flag by sending **EM_GETMODIFY** message to the control. **SendMessage** returns TRUE if the content of the control was modified. Since we stream the text into the control, it's a kind of a modification. We must set the modify flag to FALSE by sending **EM_SETMODIFY** with wParam==FALSE to the control to start anew after the stream-in opertion is finished. We immediately close the file and set **FileOpened** to TRUE to indicate that a file was opened.

When the user clicks on save/saveas menuitem, we use **EM_STREAMOUT** message to output the content of the richedit control to a file. As with the streamin callback function, the stream-out callback function is simplicity in itself. It fits perfectly with **WriteFile** .

The text operations such as cut/copy/paste/redo/undo are easily implemented by sending single message to the richedit control, **WM_CUT** / **WM_COPY** / **WM_PASTE** / **WM_REDO** / **WM_UNDO** respectively.

The delete/select all operations are done as follows:

```
  .elseif ax==IDM_DELETE
```

```
                                        invoke
SendMessage,hwndRichEdit,EM_REPLACESEL,TRUE,0
                                .elseif ax==IDM_SELECTALL
                                        mov chrg.cpMin,0
                                        mov chrg.cpMax,-1
                                        invoke
SendMessage,hwndRichEdit,EM_EXSETSEL,0,addr chrg
```

The delete operation affects the currently selection. I send **EM_REPLACESEL** message with NULL string so the richedit control will replace the currently selected text with the null string.

The select-all operation is done by sending **EM_EXSETSEL** message, specifying **cpMin** ==0 and **cpMax** ==-1 which amounts to selecting all the text.

When the user selects Option menu bar, we display a dialog box presenting the current background/text colors.

When the user clicks on one of the color boxes, it displays the choose-color dialog box. The "color box" is in fact a static control with **SS_NOTIFY** and **WS_BORDER** flag. A static control with **SS_NOTIFY** flag will notify its parent window with mouse actions on it, such as **BN_CLICKED** (**STN_CLICKED**) . That's the trick.

```
  .elseif ax==IDC_BACKCOLORBOX
                                        invoke RtlZeroMemory,addr clr,sizeof clr
                                        mov clr.lStructSize,sizeof clr
                                        push hWnd
                                        pop clr.hwndOwner
                                        push hInstance
                                        pop clr.hInstance
                                        push BackgroundColor
                                        pop clr.rgbResult
                                        mov clr.lpCustColors,offset CustomColors
                                        mov clr.Flags,CC_ANYCOLOR or CC_RGBINIT
                                        invoke ChooseColor,addr clr
                                        .if eax!=0
                                                push clr.rgbResult
                                                pop BackgroundColor
                                                invoke
GetDlgItem,hWnd,IDC_BACKCOLORBOX
                                                invoke InvalidateRect,eax,0,TRUE
                                        .endif
```

When the user clicks on one of the color box, we fill the members of the **CHOOSECOLOR** structure and call **ChooseColor** to display the choose-color dialog box. If the user selects a color, the colorref value is returned in **rgbResult** member and we store that value in **BackgroundColor** variable. After that, we force a repaint on the color box by calling **InvalidateRect** on the handle to the color box. The color box sends **WM_CTLCOLORSTATIC** message to its parent window.

```
invoke GetDlgItem,hWnd,IDC_BACKCOLORBOX
                .if eax==lParam
                        invoke CreateSolidBrush,BackgroundColor
                          ret
```

Within the **WM_CTLCOLORSTATIC** handler, we compare the handle of the static control passed in **lParam** to that of both the color boxes. If the values match, we create a new brush using the color in the variable and immediately return. The static control will use the newly created brush to paint its background.

TUTORIAL 34: EL CONTROL RICHEDIT: MÁS OPERACIONES DE TEXTO

Usted aprenderá más acerca de las operaciones de texto bajo control RichEdit. En concreto, usted sabrá cómo buscar / reemplazar texto, saltar a número de línea específico.

TEORÍA

BÚSQUEDA DE TEXTO

Hay varias operaciones de texto bajo control RichEdit. Búsqueda de texto es una operación de este tipo. La búsqueda de texto se realiza mediante el envío de **EM_FINDTEXT** o **EM_FINDTEXTEX.** Estos mensajes tiene una pequeña diferencia.

EM_FINDTEXT
wParam == opciones de búsqueda. Puede ser cualquier combinación de los valores de las opciones de la tabla below.These son idénticos para ambos
EM_FINDTEXT y **EM_FINDTEXTEX**

FR_DOWN	Si se especifica este indicador, la búsqueda comienza desde el **final** de la selección actual hasta el **final** del texto en el control (hacia **abajo).** Esta bandera tiene efecto sólo para el RichEdit 2.0 o posterior: **este comportamiento es el predeterminado para RichEdit 1.0.** El comportamiento predeterminado de RichEdit 2.0 o posterior es la búsqueda de la final de la selección actual al principio del texto **(hacia arriba).** En resumen, si usted utiliza RichEdit 1.0, usted no puede hacer nada acerca de la dirección de la búsqueda: se busca siempre a la baja. Si utiliza RichEdit 2.0 y desea buscar a la baja, se **debe** especificar esta bandera de lo contrario la búsqueda sería hacia arriba.
FR_MATCHCASE	Si se especifica este indicador, la búsqueda distingue entre mayúsculas y minúsculas.
	Si se establece este indicador, la búsqueda encuentra toda la palabra que coincida con la cadena de búsqueda especificada.

FR_WHOLEWORD	

En realidad, hay algunas banderas más, pero que son relevantes para idiomas distintos del inglés.

lParam == Puntero a la estructura **FINDTEXT.**

FINDTEXT STRUCT
chrg CHARRANGE <>
lpstrText DWORD?
FINDTEXT TERMINA

chrg es una estructura **CHARRANGE** que se define como sigue:

CHARRANGE STRUCT
cpMin DWORD?
Cpmax DWORD?
CHARRANGE EXTREMOS

cpMin contiene el índice de carácter del primer carácter de la matriz de caracteres (rango).
Cpmax contiene el índice de carácter del personaje inmediatamente después del último carácter de la matriz de caracteres.

En esencia, para buscar una cadena de texto, tiene que especificar el intervalo de caracteres en los que buscar. **El significado de cpMin**
y Cpmax variar en función de si la búsqueda es hacia abajo o hacia arriba. Si la búsqueda es hacia abajo, **cpMin**
especifica el índice del carácter de partida para buscar y **Cpmax** el índice de carácter final. Si la búsqueda es hacia arriba, el
ocurre lo contrario, es decir. **cpMin** contiene el índice de carácter final, mientras que el índice de inicio **Cpmax** carácter.

lpstrText es el puntero a la cadena de texto a buscar.

EM_FINDTEXT devuelve el índice de carácter del primer carácter en la cadena de texto correspondiente en el control RichEdit. Devuelve -1 si
no hay coincidencias.

EM_FINDTEXTEX
wParam == las opciones de búsqueda. Igual que los de **EM_FINDTEXT.**
lParam == Puntero a la estructura **FINDTEXTEX.**

FINDTEXTEX STRUCT
chrg CHARRANGE <>
lpstrText DWORD?
chrgText CHARRANGE <>
FINDTEXTEX TERMINA

Los dos primeros miembros de **FINDTEXTEX** son idénticos a los de la estructura **FINDTEXT.** chrgText es una estructura que se **CHARRANGE**
se llenará con el inicio / fin characterindices si se encuentra una coincidencia.

El valor de retorno de **EM_FINDTEXTEX** es la misma que la de **EM_FINDTEXT.**

La diferencia entre **EM_FINDTEXT** y **EM_FINDTEXTEX** es que la estructura **FINDTEXTEX** tiene un miembro adicional,
chrgText, que se llena con el inicio / fin índices de caracteres si se encuentra una coincidencia. Esto es conveniente si queremos hacer

más operaciones de texto en la cadena.

Reemplazar / Insertar texto

El control RichEdit provee **EM_SETTEXTEX** para reemplazar / insertar texto. Este mensaje combina la funcionalidad de **WM_SETTEXT** y **EM_REPLACESEL.** Se tiene la siguiente sintaxis:

EM_SETTEXTEX
wParam == puntero a la estructura **SETTEXTEX.**

SETTEXTEX STRUCT
DWORD banderas?
página de códigos DWORD?
SETTEXTEX TERMINA

banderas puede ser la combinación de los siguientes valores:

ST_DEFAULT	Borra la pila de deshacer, los descartes formato de texto enriquecido, reemplaza todo el texto.
ST_KEEPUNDO	Mantiene la pila de deshacer
ST_SELECTION	Sustituye a la selección y mantiene formato de texto enriquecido

página de códigos es la constante que especifica la página de código que desea que el texto. Por lo general, basta con utilizar **CP_ACP.**

Texto de la selección

Podemos seleccionar el texto o la programación con **EM_SETSEL EM_EXSETSEL.** Cualquiera de los dos funciona bien. Que la elección de mensaje a utilizar depende del formato disponible de los índices de caracteres. Si ya están almacenados en una estructura **CHARRANGE,** es más fácil de usar **EM_EXSETSEL.**

EM_EXSETSEL
wParam == no se utilizan. Debe ser 0
lParam == puntero a una estructura **CHARRANGE** que contiene el rango de caracteres a seleccionar.

De notificación de eventos

En el caso de un control de edición multilínea, usted tiene que la subclase, a fin de obtener los mensajes de entrada, tales como los eventos de ratón / teclado. El control RichEdit proporciona un mejor esquema que se notificará a la ventana principal de estos eventos. Con el fin de registrar las notificaciones, la ventana principal envía el mensaje **EM_SETEVENTMASK** al control RichEdit, especificando que los eventos que está interesado pulg **EM_SETEVENTMASK** tiene la siguiente sintaxis:

EM_SETEVENTMASK
wParam == no se utilizan. Debe ser 0
lParam == valor del evento máscara. Puede ser la combinación de las banderas en la tabla siguiente.

ENM_CHANGE	Envía notificaciones **EN_CHANGE**
ENM_CORRECTTEXT	Envía notificaciones **EN_CORRECTTEXT**
ENM_DRAGDROPDONE	Envía notificaciones **EN_DRAGDROPDONE**
ENM_DROPFILES	Envía notificaciones **EN_DROPFILES.**
ENM_KEYEVENTS	Envía notificaciones **EN_MSGFILTER** para eventos de teclado
ENM_LINK	**Rich Edit 2.0 y versiones posteriores:** Envía notificaciones **EN_LINK** cuando el puntero del ratón está sobre el texto que tiene el **CFE_LINK** y una de las varias acciones del ratón se realiza.
ENM_MOUSEEVENTS	Envía notificaciones de eventos de ratón **EN_MSGFILTER**
ENM_OBJECTPOSITIONS	Envía notificaciones **EN_OBJECTPOSITIONS**
ENM_PROTECTED	Envía notificaciones **EN_PROTECTED**
ENM_REQUESTRESIZE	Envía notificaciones **EN_REQUESTRESIZE**
ENM_SCROLL	Envía notificaciones **EN_HSCROLL** y **EN_VSCROLL**
ENM_SCROLLEVENTS	Envía notificaciones **EN_MSGFILTER** para los eventos de la rueda del ratón
ENM_SELCHANGE	Envía notificaciones **EN_SELCHANGE**
ENM_UPDATE	Envía notificaciones **EN_UPDATE.** **Rich Edit 2.0 y versiones posteriores:** este indicador se tiene en cuenta y las notificaciones se envían **EN_UPDATE** siempre. Sin embargo, si Rich Edit 3.0 emula Rich Edit 1.0, debe utilizar esta opción para enviar las notificaciones **EN_UPDATE**

Todas las anteriores notificaciones se enviarán como mensaje **WM_NOTIFY:** usted tiene que comprobar el miembro de la estructura del código de **NMHDR** para el mensaje de notificación. Por ejemplo, si desea registrarse para los eventos del ratón (por ejemplo, se desea proporcionar un contexto emergente de menú sensible), debe hacer algo como esto:

```
invoca SendMessage, hwndRichEdit, EM_SETEVENTMASK, 0, ENM_MOUSEEVENTS
        .....
        .....
        WndProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD,
lParam: DWORD
        .....
        ....
                . UMsg elseif == WM_NOTIFY
```

```
impulsar esi
mov esi, lParam
asumir esi: ptr NMHDR
. Si [esi]. == Código EN_MSGFILTER
        ....
        [Hacer algo aquí]
        ....
. Endif
esi pop
```

Ejemplo:

El siguiente ejemplo es la actualización de IczEdit en ningún tutorial. 33. Añade búsqueda / reemplazo de la funcionalidad y las teclas de aceleración para el programa. También procesa los eventos de ratón y ofrece un menú emergente al hacer clic derecho del ratón.

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ gdi32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ gdi32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

Const.
IDR_MAINMENU equ 101
IDM_OPEN equ 40001
IDM_SAVE equ 40002
IDM_CLOSE equ 40003
IDM_SAVEAS equ 40004
IDM_EXIT equ 40005
IDM_COPY equ 40006
IDM_CUT equ 40007
IDM_PASTE equ 40008
IDM_DELETE equ 40009
IDM_SELECTALL equ 40010
IDM_OPTION equ 40011
IDM_UNDO equ 40012
IDM_REDO equ 40013
IDD_OPTIONDLG equ 101
IDC_BACKCOLORBOX equ 1000
IDC_TEXTCOLORBOX equ 1001
IDR_MAINACCEL equ 105
IDD_FINDDLG equ 102
IDD_GOTODLG equ 103
IDD_REPLACEDLG equ 104
IDC_FINDEDIT equ 1000
IDC_MATCHCASE equ 1001
IDC_REPLACEEDIT equ 1001
IDC_WHOLEWORD equ 1002
```

```
  IDC_DOWN equ 1003
  IDC_UP equ 1004
  IDC_LINENO equ 1005
  IDM_FIND equ 40014
  IDM_FINDNEXT equ 40015
  IDM_REPLACE equ 40016
  IDM_GOTOLINE equ 40017
  IDM_FINDPREV equ 40018
  RichEditID equ 300

  . Datos
  ClassName db "IczEditClass", 0
  AppName db "IczEdit versión 2.0", 0
  RichEditDLL db "riched20.dll", 0
  RichEditClass db "RichEdit20A", 0
  NoRichEdit db "No se puede encontrar riched20.dll", 0
  ASMFilterString db "El código fuente ASM (*. Asm)", 0, "*. Asm", 0
                                        db "Todos los archivos (*. *)", 0, "*. *", 0,0
  OpenFileFail db "No se puede abrir el archivo", 0
  WannaSave db "Los datos en el control se ha modificado. ¿Quieres guardarlo?", 0
  FileOpened dd FALSO
  BackgroundColor 0FFFFFFh dd, por defecto en blanco
  TextColor dd 0; por defecto a negro
  hsearch dd?    ; El mango a la búsqueda / reemplazo cuadro de diálogo
  hAccel dd?

  . Datos?
  hInstance dd?
  hRichEdit dd?
  hwndRichEdit dd?
  NombreDeArchivo db 256 dup (?)
  AlternateFileName db 256 dup (?)
  CustomColors dd 16 dup (?)
  FindBuffer db 256 dup (?)
  ReplaceBuffer db 256 dup (?)
  uFlags dd?
  FindText FINDTEXTEX <>

  . Código
  empezar:
            mov byte ptr [FindBuffer], 0
            mov byte ptr [ReplaceBuffer], 0
            invocar GetModuleHandle, NULL
            mov hInstance, eax
            invocar LoadLibrary, addr RichEditDLL
            . Si eax! = 0
                      mov hRichEdit, eax
                      WinMain invoca, hInstance, 0,0, SW_SHOWDEFAULT
                      invocar FreeLibrary, hRichEdit
            . Más
                      invocar el cuadro de mensajes, 0, NoRichEdit addr, addr
AppName, MB_OK o MB_ICONERROR
            . Endif
            invocar ExitProcess, eax

  WinMain proc hInst: DWORD, hPrevInst: DWORD, CmdLine: DWORD, CmdShow:
DWORD
            LOCAL wc: WNDCLASSEX
```

```
            Msg LOCAL: MSG
            LOCAL hwnd: DWORD
            mov wc.cbSize, sizeof WNDCLASSEX
            mov wc.style, CS_HREDRAW o CS_VREDRAW
            mov wc.lpfnWndProc, OFFSET WndProc
            mov wc.cbClsExtra, NULL
            mov wc.cbWndExtra, NULL
            impulsar hInst
            pop wc.hInstance
            mov wc.hbrBackground, COLOR_WINDOW un
            mov wc.lpszMenuName, IDR_MAINMENU
            wc.lpszClassName mov, ClassName OFFSET
            invocar LoadIcon, NULL, IDI_APPLICATION
            mov wc.hIcon, eax
            mov wc.hIconSm, eax
            invocar LoadCursor, NULL, IDC_ARROW
            wc.hCursor mov, eax
            invocar RegisterClassEx, addr wc
            INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
          WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
          CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
          hInst, NULL
            mov hwnd, eax
            invocar ShowWindow, hwnd, SW_SHOWNORMAL
            invocar UpdateWindow, hwnd
            invocar LoadAccelerators, hInstance, IDR_MAINACCEL
            mov hAccel, eax
            . Mientras VERDADERO
                        invocar GetMessage, ADDR msg, 0,0,0
                        . Descanso. If (! Eax)
                        invocar IsDialogMessage, hsearch, addr msg
                        . Si eax == FALSO
                                    invocar TranslateAccelerator, hwnd, hAccel, addr msg
                                    . Si eax == 0
                                                invocar TranslateMessage, ADDR msg
                                                invocar DispatchMessage, ADDR msg
                                    . Endif
                        . Endif
            . Endw
            mov eax, msg.wParam
            ret
   WinMain endp

   StreamInProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesRead: DWORD
            invocar ReadFile, hFile, pBuffer, numBytes, pBytesRead, 0
            xor eax, 1
            ret
   StreamInProc endp

   StreamOutProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesWritten: DWORD
            invocar WriteFile, hFile, pBuffer, numBytes, pBytesWritten, 0
            xor eax, 1
            ret
   StreamOutProc endp

   CheckModifyState proc hWnd: DWORD
```

```
            invoca SendMessage, hwndRichEdit, EM_GETMODIFY, 0,0
            . Si eax! = 0
                        invocar el cuadro de mensajes, hWnd, WannaSave addr, addr
AppName, MB_YESNOCANCEL
                        . Si eax == IDYES
                                    invoca SendMessage, hWnd, WM_COMMAND,
IDM_SAVE, 0
                        . Elseif eax == IDCANCEL
                                    mov eax, FALSO
                                    ret
                        . Endif
            . Endif
            mov eax, TRUE
            ret
   CheckModifyState endp

   SetColor proc
            LOCAL pies cúbicos por minuto: CHARFORMAT
            invoca SendMessage, hwndRichEdit, EM_SETBKGNDCOLOR, 0,
BackgroundColor
            invocar RtlZeroMemory, addr pies cúbicos por minuto, sizeof pies
cúbicos por minuto
            mov cfm.cbSize, sizeof pies cúbicos por minuto
            mov cfm.dwMask, CFM_COLOR
            impulsar TextColor
            pop cfm.crTextColor
            invoca SendMessage, hwndRichEdit, EM_SETCHARFORMAT, SCF_ALL,
addr pies cúbicos por minuto
            ret
   SetColor endp

   OptionProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
            LOCAL clr: ChooseColor
            . Si uMsg == WM_INITDIALOG
            . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                    mov eax, wParam
                                    . Si ax == IDCANCEL
                                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                    . Elseif ax == IDC_BACKCOLORBOX
                                                invocar RtlZeroMemory, addr clr, sizeof clr
                                                mov clr.lStructSize, sizeof clr
                                                impulsar hWnd
                                                clr.hwndOwner pop
                                                impulsar hInstance
                                                pop clr.hInstance
                                                impulsar BackgroundColor
                                                pop clr.rgbResult
                                                mov clr.lpCustColors, CustomColors de
compensación
                                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                                invocar ChooseColor, addr clr
                                                . Si eax! = 0
                                                            impulsar clr.rgbResult
                                                            pop BackgroundColor
```

```
                                        invocar GetDlgItem, hWnd,
IDC_BACKCOLORBOX
                                        invocar InvalidateRect, eax, 0, TRUE
                                . Endif
                        . Elseif ax == IDC_TEXTCOLORBOX
                                invocar RtlZeroMemory, addr clr, sizeof clr
                                mov clr.lStructSize, sizeof clr
                                impulsar hWnd
                                clr.hwndOwner pop
                                impulsar hInstance
                                pop clr.hInstance
                                impulsar TextColor
                                pop clr.rgbResult
                                mov clr.lpCustColors, CustomColors de
compensación
                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                invocar ChooseColor, addr clr
                                . Si eax! = 0
                                        impulsar clr.rgbResult
                                        pop TextColor
                                        invocar GetDlgItem, hWnd,
IDC_TEXTCOLORBOX
                                        invocar InvalidateRect, eax, 0, TRUE
                                . Endif
                        . Elseif ax == IDOK
                                invoca SendMessage, hwndRichEdit,
EM_GETMODIFY, 0,0
                                push eax
                                invocar SetColor
                                pop eax
                                invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, eax, 0
                                invocar EndDialog, hWnd, 0
                        . Endif
                . Endif
        . Elseif uMsg == WM_CTLCOLORSTATIC
                invocar GetDlgItem, hWnd, IDC_BACKCOLORBOX
                . Si eax == lParam
                        invocar CreateSolidBrush, BackgroundColor
                        ret
                . Más
                        invocar GetDlgItem, hWnd, IDC_TEXTCOLORBOX
                        . Si eax == lParam
                                invocar CreateSolidBrush, TextColor
                                ret
                        . Endif
                . Endif
                mov eax, FALSO
                ret
        . Elseif uMsg == WM_CLOSE
                invocar EndDialog, hWnd, 0
        . Más
                mov eax, FALSO
                ret
        . Endif
        mov eax, TRUE
        ret
  OptionProc endp
```

```
  SearchProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
        . Si uMsg == WM_INITDIALOG
                impulsar hWnd
                pop hsearch
                invocar CheckRadioButton, hWnd, IDC_DOWN, IDC_UP,
IDC_DOWN
                invocar SendDlgItemMessage, hWnd, IDC_FINDEDIT,
WM_SETTEXT, 0, addr FindBuffer
        . Elseif uMsg == WM_COMMAND
                mov eax, wParam
                SHR EAX, 16
                . Si ax == BN_CLICKED
                        mov eax, wParam
                        . Si ax == IDOK
                                mov uFlags, 0
                                invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                invocar GetDlgItemText, hWnd,
IDC_FINDEDIT, FindBuffer addr, sizeof FindBuffer
                                . Si eax! = 0
                                        invocar IsDlgButtonChecked, hWnd,
IDC_DOWN
                                        . Si eax == BST_CHECKED
                                                o uFlags, FR_DOWN
                                                mov eax,
findtext.chrg.cpMin
                                                . Si eax! =
Findtext.chrg.cpMax
                                                        impulsar
findtext.chrg.cpMax
                                                        pop
findtext.chrg.cpMin
                                                . Endif
                                                mov findtext.chrg.cpMax, -
1
                                        . Más
                                                mov findtext.chrg.cpMax, 0
                                        . Endif
                                        invocar IsDlgButtonChecked, hWnd,
IDC_MATCHCASE
                                        . Si eax == BST_CHECKED
                                                o uFlags, FR_MATCHCASE
                                        . Endif
                                        invocar IsDlgButtonChecked, hWnd,
IDC_WHOLEWORD
                                        . Si eax == BST_CHECKED
                                                o uFlags, FR_WHOLEWORD
                                        . Endif
                                        mov findtext.lpstrText,
desplazamiento FindBuffer
                                        invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, uFlags, addr FindText
                                        . Si eax! = -1
                                                invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                        . Endif
```

```
                                        . Endif
                                . Elseif ax == IDCANCEL
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Más
                                        mov eax, FALSO
                                        ret
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        mov hsearch, 0
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
  SearchProc endp

  ReplaceProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
                LOCAL setText: SETTEXTEX
                . Si uMsg == WM_INITDIALOG
                        impulsar hWnd
                        pop hsearch
                        invocar SetDlgItemText, hWnd, IDC_FINDEDIT, addr
FindBuffer
                        invocar SetDlgItemText, hWnd, IDC_REPLACEEDIT, addr
ReplaceBuffer
                . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                mov eax, wParam
                                . Si ax == IDCANCEL
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Elseif ax == IDOK
                                        invocar GetDlgItemText, hWnd,
IDC_FINDEDIT, FindBuffer addr, sizeof FindBuffer
                                        invocar GetDlgItemText, hWnd,
IDC_REPLACEEDIT, ReplaceBuffer addr, sizeof ReplaceBuffer
                                        mov findtext.chrg.cpMin, 0
                                        mov findtext.chrg.cpMax, -1
                                        mov findtext.lpstrText, desplazamiento
FindBuffer
                                        mov settext.flags, ST_SELECTION
                                        mov settext.codepage, CP_ACP
                                        . Mientras VERDADERO
                                                invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, FR_DOWN, addr FindText
                                                . Si eax == -1
                                                        . Descanso
                                                . Más
                                                        invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                        invoca SendMessage,
hwndRichEdit, EM_SETTEXTEX, setText addr, addr ReplaceBuffer
                                                . Endif
```

```
                                        . Endw
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        mov hsearch, 0
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
    ReplaceProc endp

    GoToProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
                LOCAL LineNo: DWORD
                LOCAL chrg: CHARRANGE
                . Si uMsg == WM_INITDIALOG
                        impulsar hWnd
                        pop hsearch
                . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                mov eax, wParam
                                . Si ax == IDCANCEL
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Elseif ax == IDOK
                                        invocar GetDlgItemInt, hWnd, IDC_LINENO,
NULL, FALSE
                                        mov LineNo, eax
                                        invoca SendMessage, hwndRichEdit,
EM_GETLINECOUNT, 0,0
                                        . Si eax> LineNo
                                                invoca SendMessage, hwndRichEdit,
EM_LINEINDEX, LineNo, 0
                                                mov chrg.cpMin, eax
                                                mov chrg.cpMax, eax
                                                invoca SendMessage, hwndRichEdit,
EM_EXSETSEL, 0, addr chrg
                                                invocar SetFocus, hwndRichEdit
                                        . Endif
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        mov hsearch, 0
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
    GoToProc endp

    PrepareEditMenu hSubMenu proc: DWORD
                LOCAL chrg: CHARRANGE
```

```
            invoca SendMessage, hwndRichEdit, EM_CANPASTE, CF_TEXT, 0
            . Si eax == 0; no hay texto en el portapapeles
                        invocar EnableMenuItem, hSubMenu, IDM_PASTE, MF_GRAYED
            . Más
                        invocar EnableMenuItem, hSubMenu, IDM_PASTE,
MF_ENABLED
            . Endif
            invoca SendMessage, hwndRichEdit, EM_CANUNDO, 0,0
            . Si eax == 0
                        invocar EnableMenuItem, hSubMenu, IDM_UNDO, MF_GRAYED
            . Más
                        invocar EnableMenuItem, hSubMenu, IDM_UNDO,
MF_ENABLED
            . Endif
            invoca SendMessage, hwndRichEdit, EM_CANREDO, 0,0
            . Si eax == 0
                        invocar EnableMenuItem, hSubMenu, IDM_REDO, MF_GRAYED
            . Más
                        invocar EnableMenuItem, hSubMenu, IDM_REDO,
MF_ENABLED
            . Endif
            invoca SendMessage, hwndRichEdit, EM_EXGETSEL, 0, addr chrg
            mov eax, chrg.cpMin
            . Si eax == chrg.cpMax; ninguna selección actual
                        invocar EnableMenuItem, hSubMenu, IDM_COPY, MF_GRAYED
                        invocar EnableMenuItem, hSubMenu, IDM_CUT, MF_GRAYED
                        invocar EnableMenuItem, hSubMenu, IDM_DELETE,
MF_GRAYED
            . Más
                        invocar EnableMenuItem, hSubMenu, IDM_COPY, MF_ENABLED
                        invocar EnableMenuItem, hSubMenu, IDM_CUT, MF_ENABLED
                        invocar EnableMenuItem, hSubMenu, IDM_DELETE,
MF_ENABLED
            . Endif
            ret
  PrepareEditMenu endp

  WndProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
            LOCAL de n: OPENFILENAME
            Búfer local [256]: BYTE
            LOCAL editstream: EDITSTREAM
            LOCAL hFile: DWORD
            LOCAL hPopup: DWORD
            Locales del PT: PUNTO
            LOCAL chrg: CHARRANGE
            . Si uMsg == WM_CREATE
                        invocar CreateWindowEx, WS_EX_CLIENTEDGE, addr
RichEditClass, 0, o WS_CHILD WS_VISIBLE o ES_MULTILINE o WS_VSCROLL o
WS_HSCROLL o estilo ES_NOHIDESEL, \
                                    CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, hWnd, RichEditID, hInstance, 0
                        mov hwndRichEdit, eax
                        invoca SendMessage, hwndRichEdit, EM_LIMITTEXT, -1,0
                        invocar SetColor
                        invoca SendMessage, hwndRichEdit, EM_SETMODIFY, FALSE, 0
                        invoca SendMessage, hwndRichEdit, EM_SETEVENTMASK, 0,
ENM_MOUSEEVENTS
```

```
                invoca SendMessage, hwndRichEdit, EM_EMPTYUNDOBUFFER,
0,0
        . UMsg elseif == WM_NOTIFY
                impulsar esi
                mov esi, lParam
                asumir esi: ptr NMHDR
                . Si [esi]. == Código EN_MSGFILTER
                        asumir esi: ptr msgfilter
                        . Si [esi]. Msg == WM_RBUTTONDOWN
                                invocar GetMenu, hWnd
                                invocar GetSubMenu, eax, 1
                                mov hPopup, eax
                                invocar PrepareEditMenu, hPopup
                                mov edx, [esi]. lParam
                                mov ecx, edx
                                y edx, 0FFFFh
                                shr ecx, 16
                                mov pt.x, edx
                                mov pt.y, ecx
                                invocar ClientToScreen, hWnd, addr pt
                                invocar TrackPopupMenu, hPopup,
TPM_LEFTALIGN o TPM_BOTTOMALIGN, pt.x, pt.y, NULL, hWnd, NULL
                        . Endif
                . Endif
                esi pop
        . UMsg elseif == WM_INITMENUPOPUP
                mov eax, lParam
                . Si ax == 0; menú de archivo
                        . Si FileOpened == TRUE, un archivo ya está abierto
                                invocar EnableMenuItem, wParam,
IDM_OPEN, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_SAVE, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_ENABLED
                        . Más
                                invocar EnableMenuItem, wParam,
IDM_OPEN, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_SAVE, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_GRAYED
                        . Endif
                . Elseif ax == 1; menú de edición
                        invocar PrepareEditMenu, wParam
                . Elseif ax == 2; barra de búsqueda del menú
                        . Si FileOpened == TRUE
                                invocar EnableMenuItem, wParam,
IDM_FIND, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_FINDNEXT, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_FINDPREV, MF_ENABLED
```

```
                                        invocar EnableMenuItem, wParam,
IDM_REPLACE, MF_ENABLED
                                        invocar EnableMenuItem, wParam,
IDM_GOTOLINE, MF_ENABLED
                                . Más
                                        invocar EnableMenuItem, wParam,
IDM_FIND, MF_GRAYED
                                        invocar EnableMenuItem, wParam,
IDM_FINDNEXT, MF_GRAYED
                                        invocar EnableMenuItem, wParam,
IDM_FINDPREV, MF_GRAYED
                                        invocar EnableMenuItem, wParam,
IDM_REPLACE, MF_GRAYED
                                        invocar EnableMenuItem, wParam,
IDM_GOTOLINE, MF_GRAYED
                                . Endif
                        . Endif
                . Elseif uMsg == WM_COMMAND
                        . Si lParam == 0; comandos de menú
                                mov eax, wParam
                                . Si ax == IDM_OPEN
                                        invocar RtlZeroMemory, addr de n, sizeof
OFN
                                        mov ofn.lStructSize, sizeof OFN
                                        impulsar hWnd
                                        ofn.hwndOwner pop
                                        impulsar hInstance
                                        pop ofn.hInstance
                                        mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                        mov ofn.lpstrFile, FileName desplazamiento
                                        mov byte ptr [nombreDeArchivo], 0
                                        mov ofn.nMaxFile, sizeof nombreDeArchivo
                                        mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                        invocar GetOpenFileName, addr OFN
                                        . Si eax! = 0
                                                invocar CreateFile, addr FileName
GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0
                                                . Si eax! =
INVALID_HANDLE_VALUE
                                                        mov hFile, eax
                                                        ;
================================================
===============
                                                        ; Escuchar el texto en el
control RichEdit
                                                        ;
================================================
===============
                                                        mov editstream.dwCookie,
eax
                                                        mov
editstream.pfnCallback, desplazamiento StreamInProc
                                                        invoca SendMessage,
hwndRichEdit, EM_STREAMIN, SF_TEXT, addr editstream
```

```
                                                ;
================================================
=========
                                                ; Inicializa el estado de
modificación a la falsa
                                                ;
================================================
=========
                                                invoca SendMessage,
hwndRichEdit, EM_SETMODIFY, FALSE, 0
                                                invocar CloseHandle, hFile
                                                mov FileOpened, TRUE
                                        . Más
                                                invocar el cuadro de
mensajes, hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                        . Endif
                                . Endif
                        . Elseif ax == IDM_CLOSE
                                invocar CheckModifyState, hWnd
                                . Si eax == TRUE
                                        invocar SetWindowText,
hwndRichEdit, 0
                                        mov FileOpened, FALSO
                                . Endif
                        . Elseif ax == IDM_SAVE
                                invocar CreateFile, addr FileName
GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0
                                . Si eax! = INVALID_HANDLE_VALUE
  @ @:
                                        mov hFile, eax
                                        ;
================================================
===============
                                        ; Transmitir el texto en el archivo
                                        ;
================================================
===============
                                        mov editstream.dwCookie, eax
                                        mov editstream.pfnCallback,
desplazamiento StreamOutProc
                                        invoca SendMessage, hwndRichEdit,
EM_STREAMOUT, SF_TEXT, addr editstream
                                        ;
================================================
=========
                                        ; Inicializa el estado de modificación
a la falsa
                                        ;
================================================
=========
                                        invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, FALSE, 0
                                        invocar CloseHandle, hFile
                                . Más
                                        invocar el cuadro de mensajes,
hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                . Endif
```

```
                        . Elseif ax == IDM_COPY
                                invoca SendMessage, hwndRichEdit,
WM_COPY, 0,0
                        . Elseif ax == IDM_CUT
                                invoca SendMessage, hwndRichEdit,
WM_CUT, 0,0
                        . Elseif ax == IDM_PASTE
                                invoca SendMessage, hwndRichEdit,
WM_PASTE, 0,0
                        . Elseif ax == IDM_DELETE
                                invoca SendMessage, hwndRichEdit,
EM_REPLACESEL, TRUE, 0
                        . Elseif ax == IDM_SELECTALL
                                mov chrg.cpMin, 0
                                mov chrg.cpMax, -1
                                invoca SendMessage, hwndRichEdit,
EM_EXSETSEL, 0, addr chrg
                        . Elseif ax == IDM_UNDO
                                invoca SendMessage, hwndRichEdit,
EM_UNDO, 0,0
                        . Elseif ax == IDM_REDO
                                invoca SendMessage, hwndRichEdit,
EM_REDO, 0,0
                        . Elseif ax == IDM_OPTION
                                invocar DialogBoxParam, hInstance,
IDD_OPTIONDLG, hWnd, addr OptionProc, 0
                        . Elseif ax == IDM_SAVEAS
                                invocar RtlZeroMemory, addr de n, sizeof
OFN
                                mov ofn.lStructSize, sizeof OFN
                                impulsar hWnd
                                ofn.hwndOwner pop
                                impulsar hInstance
                                pop ofn.hInstance
                                mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                mov ofn.lpstrFile, AlternateFileName
desplazamiento
                                mov byte ptr [AlternateFileName], 0
                                mov ofn.nMaxFile, sizeof AlternateFileName
                                mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                invocar GetSaveFileName, addr OFN
                                . Si eax! = 0
                                        invocar CreateFile,
AlternateFileName addr, GENERIC_WRITE, FILE_SHARE_READ, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
                                        . Si eax! =
INVALID_HANDLE_VALUE
                                                jmp @ B
                                        . Endif
                                . Endif
                        . Elseif ax == IDM_FIND
                                . Si hsearch == 0
                                        invocar CreateDialogParam,
hInstance, IDD_FINDDLG, hWnd, addr SearchProc, 0
                                . Endif
                        . Elseif ax == IDM_REPLACE
```

```
                                        . Si hsearch == 0
                                                invocar CreateDialogParam,
hInstance, IDD_REPLACEDLG, hWnd, addr ReplaceProc, 0
                                        . Endif
                                . Elseif ax == IDM_GOTOLINE
                                        . Si hsearch == 0
                                                invocar CreateDialogParam,
hInstance, IDD_GOTODLG, hWnd, addr GoToProc, 0
                                        . Endif
                                . Elseif ax == IDM_FINDNEXT
                                        invocar lstrlen, addr FindBuffer
                                        . Si eax! = 0
                                                invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                                mov eax, findtext.chrg.cpMin
                                                . Si eax! = Findtext.chrg.cpMax
                                                        impulsar
findtext.chrg.cpMax
                                                        pop findtext.chrg.cpMin
                                                . Endif
                                                mov findtext.chrg.cpMax, -1
                                                mov findtext.lpstrText,
desplazamiento FindBuffer
                                                invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, FR_DOWN, addr FindText
                                                . Si eax! = -1
                                                        invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                . Endif
                                        . Endif
                                . Elseif ax == IDM_FINDPREV
                                        invocar lstrlen, addr FindBuffer
                                        . Si eax! = 0
                                                invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                                mov findtext.chrg.cpMax, 0
                                                mov findtext.lpstrText,
desplazamiento FindBuffer
                                                invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, 0, addr FindText
                                                . Si eax! = -1
                                                        invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                . Endif
                                        . Endif
                                . Elseif ax == IDM_EXIT
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        invocar CheckModifyState, hWnd
                        . Si eax == TRUE
                                invocar DestroyWindow, hWnd
                        . Endif
                . Elseif uMsg == WM_SIZE
                        mov eax, lParam
                        mov edx, eax
                        y EAX, 0FFFFh
```

```
                shr edx, 16
                invocar MoveWindow, hwndRichEdit, 0,0, eax, edx, TRUE
        . Elseif uMsg == WM_DESTROY
                invocar PostQuitMessage, NULL
        . Más
                invocar DefWindowProc, hWnd, uMsg, wParam, lParam
                ret
        . Endif
        xor eax, eax
        ret
WndProc ENDP
poner fin a empezar a
```

Análisis

La capacidad de búsqueda de texto se realiza con **EM_FINDTEXTEX.** Cuando el usuario hace clic en Buscar menuitem, el mensaje se envía **IDM_FIND** y el cuadro de diálogo Buscar aparece.


```
invocar GetDlgItemText, hWnd, IDC_FINDEDIT, FindBuffer addr, sizeof
FindBuffer
        . Si eax! = 0
```

Cuando el usuario escribe el texto que desea buscar y luego presione el botón OK, tenemos el texto que se debe buscar en FindBuffer.

```
mov uFlags, 0
                invoca SendMessage, hwndRichEdit, EM_EXGETSEL, 0, addr
findtext.chrg
```

Si la cadena de texto no es nulo, seguimos para inicializar **uFlags** variables a 0. Esta variable se utiliza para almacenar las banderas de búsqueda utilizados con **EM_FINDTEXTEX.** Después de eso, se obtiene la selección actual con **EM_EXGETSEL** porque tenemos que conocer el punto de partida de la operación de búsqueda.

```
invocar IsDlgButtonChecked, hWnd, IDC_DOWN
                . Si eax == BST_CHECKED
                        o uFlags, FR_DOWN
                        mov eax, findtext.chrg.cpMin
                        . Si eax! = Findtext.chrg.cpMax
                                impulsar findtext.chrg.cpMax
                                pop findtext.chrg.cpMin
                        . Endif
                        mov findtext.chrg.cpMax, -1
                . Más
                        mov findtext.chrg.cpMax, 0
```

```
                . Endif
```

La siguiente parte es un poco difícil. Comprobamos el botón de radio para determinar la dirección en qué dirección debe ir la búsqueda. Si la búsqueda hacia abajo se indica, nos pusimos la bandera de **FR_DOWN uFlags.** Después de eso, comprobar si la selección está actualmente en vigor mediante la comparación de los valores de **cpMin** y **Cpmax.** Si ambos valores no son iguales, significa que hay una selección actual y que debe continuar la búsqueda a partir del final de la selección al final del texto en el control. Por lo tanto tenemos que sustituir el valor de **Cpmax** con la de **cpMin** y cambiar el valor de **Cpmax** a -1 (0FFFFFFFFh). Si no hay ninguna selección actual, el intervalo para realizar la búsqueda es desde la posición de cursor actual hasta el final del texto.

Si el usuario elige para realizar la búsqueda hacia arriba, se utiliza el intervalo desde el inicio de la selección al comienzo del texto en el control. Es por eso que sólo se modifica el valor de **Cpmax** a 0. En el caso de búsqueda hacia arriba, **cpMin** contiene el índice de carácter del último carácter en el rango de búsqueda y **Cpmax** el índice de carácter del primer carácter en el rango de búsqueda. Es la inversa de la búsqueda hacia abajo.

```
   invocar IsDlgButtonChecked, hWnd, IDC_MATCHCASE
                . Si eax == BST_CHECKED
                        o uFlags, FR_MATCHCASE
                . Endif
                invocar IsDlgButtonChecked, hWnd, IDC_WHOLEWORD
                . Si eax == BST_CHECKED
                        o uFlags, FR_WHOLEWORD
                . Endif
                mov findtext.lpstrText, desplazamiento FindBuffer
```

Seguimos seleccione las casillas de las banderas de la búsqueda, es decir, **FR_MATCHCASE** y **FR_WHOLEWORD.** Por último, ponemos el desplazamiento del texto que desea buscar en el miembro **lpstrText.**

```
   invoca SendMessage, hwndRichEdit, EM_FINDTEXTEX, uFlags, addr FindText
                . Si eax! = -1
                        invoca SendMessage, hwndRichEdit, EM_EXSETSEL, 0,
addr findtext.chrgText
                . Endif
        . Endif
```

Ahora estamos listos para emitir **EM_FINDTEXTEX.** Después de eso, se examina el resultado de la búsqueda que devuelve **SendMessage.** Si el valor de retorno es -1, no se encuentran coincidencias en el rango de búsqueda. De lo contrario, el miembro de estructura **chrgText FINDTEXTEX** se llena con los índices de caracteres del texto coincidente. Por lo tanto proceder a seleccionar con **EM_EXSETSEL.**

La operación de sustitución se hace en gran parte de la misma manera.

```
   invocar GetDlgItemText, hWnd, IDC_FINDEDIT, FindBuffer addr, sizeof
FindBuffer
        invocar GetDlgItemText, hWnd, IDC_REPLACEEDIT, ReplaceBuffer addr,
sizeof ReplaceBuffer
```

Nos recuperar el texto que desea buscar y el texto que se utiliza para reemplazar.

```
   mov findtext.chrg.cpMin, 0
        mov findtext.chrg.cpMax, -1
```

```
            mov findtext.lpstrText, desplazamiento FindBuffer
```

Para facilitar la operación de reemplazo afecta a todo el texto en el control. Así, el índice de partida es 0 y el índice final es -1.

```
   mov settext.flags, ST_SELECTION
            mov settext.codepage, CP_ACP
```

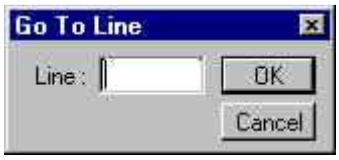
Inicializamos la estructura **SETTEXTEX** para indicar que queremos reemplazar la selección actual y utilizar el sistema por defecto la página de códigos.

```
   . Mientras VERDADERO
                        invoca SendMessage, hwndRichEdit, EM_FINDTEXTEX,
FR_DOWN, addr FindText
                        . Si eax == -1
                                    . Descanso
                        . Más
                                    invoca SendMessage, hwndRichEdit, EM_EXSETSEL, 0,
addr findtext.chrgText
                                    invoca SendMessage, hwndRichEdit, EM_SETTEXTEX,
setText addr, addr ReplaceBuffer
                        . Endif
            . Endw
```

Entramos en un bucle infinito, buscando el texto coincidente. Si uno se encuentra, lo seleccionamos con **EM_EXSETSEL** y reemplazarlo con **EM_SETTEXTEX.** Cuando no hay más coincidencia, salimos del bucle.

Buscar siguiente y **Buscar anterior.** Características de uso de mensaje de **EM_FINDTEXTEX** de manera similar a la operación de búsqueda.

Vamos a examinar la función Ir a la siguiente línea. Cuando el usuario hace clic en Ir a menuitem línea, se muestra un cuadro de diálogo a continuación:



Cuando el usuario escribe un número de línea y presiona el botón Ok, comenzamos la operación.

```
   invocar GetDlgItemInt, hWnd, IDC_LINENO, NULL, FALSE
            mov LineNo, eax
```

Obtenga el número de línea desde el control de edición

```
   invoca SendMessage, hwndRichEdit, EM_GETLINECOUNT, 0,0
            . Si eax> LineNo
```

Obtener el número de líneas en el control. Compruebe si el usuario especifica el número de línea que está fuera de la gama.

```
   invoca SendMessage, hwndRichEdit, EM_LINEINDEX, LineNo, 0
```

Si el número de línea es válida, queremos mover el cursor al primer carácter de esa línea. Así que enviamos **EM_LINEINDEX** mensaje al control RichEdit. Este mensaje devuelve el índice de carácter del primer carácter en la línea indicada. Enviamos el número de línea en wParam y, en cambio, que tiene el índice de carácter.

```
invoca SendMessage, hwndRichEdit, EM_SETSEL, eax, eax
```

Para establecer la selección actual, en este caso usamos **EM_SETSEL** porque los índices de caracteres no se encuentran ya en una estructura **CHARRANGE** por lo que nos salva dos instrucciones (para poner los índices en una estructura **CHARRANGE).**

```
invocar SetFocus, hwndRichEdit
        . Endif
```

El cursor no se mostrará a menos que el control RichEdit tiene el foco. Así que llamamos a **SetFocus** en él.

---

http://www.youtube.com/watch?v=78okbl9-UZQ de marco.

Tutorial 35: Control RichEdit: Sintaxis resaltado de

Antes de leer este tutorial, permítame que le advierta que es un tema complicado: no es adecuado para un principiante. Este es el último de los tutoriales de control RichEdit.

TEORÍA

Resaltado de sintaxis es un tema de debate caliente para los editores de la escritura de texto. El mejor método (en mi opinión) es el código de un control personalizado de edición y este es el enfoque adoptado por gran cantidad de software comercial. Sin embargo, para aquellos de nosotros que no tenemos tiempo para la codificación de dicho control, la mejor cosa siguiente es adaptar el control existente para que se adapte a nuestras necesidades.

Echemos un vistazo a lo que ofrece el control RichEdit para que nos ayuden en la implementación de resaltado de sintaxis. Debo decir en este momento en que el método que sigue no es la "correcta" ruta de acceso: sólo quiero mostrar la trampa que caen muchos de ellos por. El control RichEdit provee **EM_SETCHARFORMAT** mensaje que se puede utilizar para cambiar el color del texto. A primera vista, este mensaje parece ser la solución perfecta (lo sé porque yo era uno de la víctima). Sin embargo, un examen más detallado le mostrará varias cosas que son indeseables:

- **EM_SETCHARFORMAT** sólo funciona para un texto en la actualidad en la selección o todo el texto en el control. Si desea cambiar el color del texto (seleccionalo) una determinada palabra, primero debe seleccionarlo.
- **EM_SETCHARFORMAT** es muy lenta
- Tiene un problema con la posición del cursor en el control RichEdit

Con la discusión anterior, se puede ver que el uso de **EM_SETCHARFORMAT** es una mala elección. Te voy a mostrar la "relativamente correcta" la elección.

El método que utilizo es "resaltado de sintaxis justo a tiempo". Me busca objetos sólo la parte visible de texto. Así, la velocidad del resaltado de no estar relacionado con el tamaño del archivo en absoluto. No importa qué tan grande el archivo, sólo una pequeña parte de él es visible en un tiempo.

¿Cómo hacer eso? La respuesta es simple:

1. subclase del control RichEdit y el mensaje **WM_PAINT** mango dentro de su propio procedimiento de ventana
2. Cuando se recibe un mensaje **WM_PAINT,** llama al procedimiento de ventana original del control RichEdit dejar que se actualice la pantalla, como de costumbre.
3. Después de eso, sobrescribir las palabras que se hilighted con un color diferente

Por supuesto, el camino no es tan fácil: todavía hay un par de cosas de menor importancia para fijar el método anterior, pero funciona bastante bien. La velocidad de la pantalla es muy satisfactorio.

Ahora vamos a concentrarnos en los detalles. El proceso de creación de subclases es simple y no requiere mucha atención. La parte más complicada es cuando tenemos que encontrar una manera rápida de buscar las palabras que se hilighted. Esto se complica aún más por la necesidad de **no** busca objetos de cualquier palabra dentro de un bloque de comentario.

El método que yo uso puede no ser la mejor, pero funciona bien. Estoy seguro de que usted puede encontrar una manera más rápida. De todos modos, aquí está:

- Puedo crear una matriz de 256 dword, inicializado a 0. Cada dword corresponde a un posible carácter ASCII, llamado **ASMSyntaxArray.** Por ejemplo, el valor DWORD 21 representa el ascii 20 horas (el espacio). Yo las utilizo como una tabla de búsqueda rápida: Por ejemplo, si tengo la palabra "incluir", voy a extraer el primer carácter (i) de la palabra y buscar el valor DWORD en el índice correspondiente. Si eso es DWORD 0, sé inmediatamente que no hay palabras que se hilighted comenzando con "i". Si el valor DWORD es distinto de cero, que contiene el puntero a la lista enlazada de la estructura **WORDINFO** que contiene la información acerca de la palabra que se hilighted.
- He leído las palabras que se hilighted y crear una estructura **WORDINFO** para cada uno de ellos.

WORDINFO estructura
WordLen dd? , La longitud de la palabra: se utiliza como una comparación rápida
pszWord dd? ; Puntero a la palabra
pcolor dd? , Apuntan a la DWORD que contiene el color utilizado para hilite la palabra
Nextlink dd? , Apuntan a la estructura de la próxima WORDINFO
WORDINFO termina

Como puedes ver, yo uso la longitud de la palabra como la comparación rápida segundos. Si el primer carácter de las coincidencias de palabras, el próximo comparar su longitud a las palabras disponibles. Cada DWORD en **ASMSyntaxArray** contiene un puntero a la cabeza de la matriz **WORDINFO** asociado. Por ejemplo, el valor DWORD que representa el carácter "i" contiene el puntero a la lista enlazada de las palabras que comienzan con "i". Puntos miembros **pcolor** en

el valor DWORD que contiene el valor del color utilizado para la palabra. Selecciona los puntos a **pszWord** la palabra que se hilighted, en minúsculas.

- La memoria para la lista enlazada se asigna desde el montón predeterminado, así que es rápido y fácil de limpiar, es decir, no hay limpieza necesaria en absoluto.

La lista de palabras se almacena en un archivo llamado "wordfile.txt" y acceder a ella con **GetPrivateProfileString** API. Me ofrecen hasta un 10 por coloreado de sintaxis diferente, a partir de C1 a C10. La matriz de color se llama **ASMColorArray.** pcolor miembro de cada estructura de puntos **WORDINFO** a uno de los dwords en **ASMColorArray.** Por lo tanto, es fácil cambiar el color de sintaxis sobre la marcha: sólo tiene que cambiar el valor DWORD de **ASMColorArray** y todas las palabras que utilizan ese color a utilizar el nuevo color de inmediato.

Ejemplo

```
0.386
. Modelo plano, stdcall
casemap opción: ninguno
include \ masm32 \ include \ windows.inc
include \ masm32 \ include \ user32.inc
include \ masm32 \ include \ comdlg32.inc
include \ masm32 \ include \ gdi32.inc
include \ masm32 \ include \ kernel32.inc
includelib \ masm32 \ lib \ gdi32.lib
includelib \ masm32 \ lib \ comdlg32.lib
includelib \ masm32 \ lib \ user32.lib
includelib \ masm32 \ lib \ kernel32.lib

WinMain proto: DWORD,: DWORD,: DWORD,: DWORD

WORDINFO estructura
        WordLen dd?    , La longitud de la palabra: se utiliza como una
comparación rápida
        pszWord dd?    ; Puntero a la palabra
        pcolor dd?     , Apuntan a la DWORD que contiene el color utilizado para
hilite la palabra
        Nextlink dd?   , Apuntan a la estructura de la próxima WORDINFO
WORDINFO termina

Const.
IDR_MAINMENU equ 101
IDM_OPEN equ 40001
IDM_SAVE equ 40002
IDM_CLOSE equ 40003
IDM_SAVEAS equ 40004
IDM_EXIT equ 40005
IDM_COPY equ 40006
IDM_CUT equ 40007
IDM_PASTE equ 40008
IDM_DELETE equ 40009
IDM_SELECTALL equ 40010
IDM_OPTION equ 40011
IDM_UNDO equ 40012
IDM_REDO equ 40013
IDD_OPTIONDLG equ 101
IDC_BACKCOLORBOX equ 1000
```

```
IDC_TEXTCOLORBOX equ 1001
IDR_MAINACCEL equ 105
IDD_FINDDLG equ 102
IDD_GOTODLG equ 103
IDD_REPLACEDLG equ 104
IDC_FINDEDIT equ 1000
IDC_MATCHCASE equ 1001
IDC_REPLACEEDIT equ 1001
IDC_WHOLEWORD equ 1002
IDC_DOWN equ 1003
IDC_UP equ 1004
IDC_LINENO equ 1005
IDM_FIND equ 40014
IDM_FINDNEXT equ 40015
IDM_REPLACE equ 40016
IDM_GOTOLINE equ 40017
IDM_FINDPREV equ 40018
RichEditID equ 300

. Datos
ClassName db "IczEditClass", 0
AppName db "IczEdit versión 3.0", 0
RichEditDLL db "riched20.dll", 0
RichEditClass db "RichEdit20A", 0
NoRichEdit db "No se puede encontrar riched20.dll", 0
ASMFilterString db "El código fuente ASM (*. Asm)", 0, "*. Asm", 0
                                        db "Todos los archivos (*. *)", 0, "*. *", 0,0
OpenFileFail db "No se puede abrir el archivo", 0
WannaSave db "Los datos en el control se ha modificado. ¿Quieres guardarlo?", 0
FileOpened dd FALSO
BackgroundColor 0FFFFFFh dd, por defecto en blanco
TextColor dd 0; por defecto a negro
WordFileName db "\ wordfile.txt", 0
ASMSection db "asamblea", 0
C1Key db "C1", 0
C2Key db "C2", 0
C3Key db "C3", 0
C4Key db "C4", 0
C5Key db "C5", 0
C6Key db "C6", 0
C7Key db "C7", 0
C8Key db "C8", 0
C9Key db "C9", 0
C10Key db "C10", 0
ZeroString db 0
ASMColorArray dd 0FF0000h, 0805F50h, 0FFh, 666F00h, 44F0h, 5F8754h, 4 dup
(0FF0000h)
CommentColor dd 808000h

. Datos?
hInstance dd?
hRichEdit dd?
hwndRichEdit dd?
NombreDeArchivo db 256 dup (?)
AlternateFileName db 256 dup (?)
CustomColors dd 16 dup (?)
FindBuffer db 256 dup (?)
ReplaceBuffer db 256 dup (?)
```

```
  uFlags dd?
  FindText FINDTEXTEX <>
  ASMSyntaxArray dd 256 dup (?)
  hsearch dd?    ; El mango a la búsqueda / reemplazo cuadro de diálogo
  hAccel dd?
  hMainHeap dd?    ; Montón de mango
  OldWndProc dd?
  RichEditVersion dd?

  . Código
  empezar:
            mov byte ptr [FindBuffer], 0
            mov byte ptr [ReplaceBuffer], 0
            invocar GetModuleHandle, NULL
            mov hInstance, eax
            invocar LoadLibrary, addr RichEditDLL
            . Si eax! = 0
                      mov hRichEdit, eax
                      invocar GetProcessHeap
                      mov hMainHeap, eax
                      llame FillHiliteInfo
                      WinMain invoca, hInstance, 0,0, SW_SHOWDEFAULT
                      invocar FreeLibrary, hRichEdit
            . Más
                      invocar el cuadro de mensajes, 0, NoRichEdit addr, addr
AppName, MB_OK o MB_ICONERROR
            . Endif
            invocar ExitProcess, eax

  WinMain proc hInst: DWORD, hPrevInst: DWORD, CmdLine: DWORD, CmdShow:
DWORD
            LOCAL wc: WNDCLASSEX
            Msg LOCAL: MSG
            LOCAL hwnd: DWORD
            mov wc.cbSize, sizeof WNDCLASSEX
            mov wc.style, CS_HREDRAW o CS_VREDRAW
            mov wc.lpfnWndProc, OFFSET WndProc
            mov wc.cbClsExtra, NULL
            mov wc.cbWndExtra, NULL
            impulsar hInst
            pop wc.hInstance
            mov wc.hbrBackground, COLOR_WINDOW un
            mov wc.lpszMenuName, IDR_MAINMENU
            wc.lpszClassName mov, ClassName OFFSET
            invocar LoadIcon, NULL, IDI_APPLICATION
            mov wc.hIcon, eax
            mov wc.hIconSm, eax
            invocar LoadCursor, NULL, IDC_ARROW
            wc.hCursor mov, eax
            invocar RegisterClassEx, addr wc
            INVOKE CreateWindowEx, NULL, ADDR ClassName, ADDR AppName, \
          WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
          CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, \
          hInst, NULL
            mov hwnd, eax
            invocar ShowWindow, hwnd, SW_SHOWNORMAL
            invocar UpdateWindow, hwnd
            invocar LoadAccelerators, hInstance, IDR_MAINACCEL
```

```
            mov hAccel, eax
            . Mientras VERDADERO
                        invocar GetMessage, ADDR msg, 0,0,0
                        . Descanso. If (! Eax)
                        invocar IsDialogMessage, hsearch, addr msg
                        . Si eax == FALSO
                                    invocar TranslateAccelerator, hwnd, hAccel, addr msg
                                    . Si eax == 0
                                                invocar TranslateMessage, ADDR msg
                                                invocar DispatchMessage, ADDR msg
                                    . Endif
                        . Endif
            . Endw
            mov eax, msg.wParam
            ret
  WinMain endp

  StreamInProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesRead: DWORD
            invocar ReadFile, hFile, pBuffer, numBytes, pBytesRead, 0
            xor eax, 1
            ret
  StreamInProc endp

  StreamOutProc proc hFile: DWORD, pBuffer: DWORD, numBytes: DWORD,
pBytesWritten: DWORD
            invocar WriteFile, hFile, pBuffer, numBytes, pBytesWritten, 0
            xor eax, 1
            ret
  StreamOutProc endp

  CheckModifyState proc hWnd: DWORD
            invoca SendMessage, hwndRichEdit, EM_GETMODIFY, 0,0
            . Si eax! = 0
                        invocar el cuadro de mensajes, hWnd, WannaSave addr, addr
AppName, MB_YESNOCANCEL
                        . Si eax == IDYES
                                    invoca SendMessage, hWnd, WM_COMMAND,
IDM_SAVE, 0
                        . Elseif eax == IDCANCEL
                                    mov eax, FALSO
                                    ret
                        . Endif
            . Endif
            mov eax, TRUE
            ret
  CheckModifyState endp

  SetColor proc
            LOCAL pies cúbicos por minuto: CHARFORMAT
            invoca SendMessage, hwndRichEdit, EM_SETBKGNDCOLOR, 0,
BackgroundColor
            invocar RtlZeroMemory, addr pies cúbicos por minuto, sizeof pies
cúbicos por minuto
            mov cfm.cbSize, sizeof pies cúbicos por minuto
            mov cfm.dwMask, CFM_COLOR
            impulsar TextColor
            pop cfm.crTextColor
```

```
            invoca SendMessage, hwndRichEdit, EM_SETCHARFORMAT, SCF_ALL,
addr pies cúbicos por minuto
            ret
   SetColor endp

   OptionProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
            LOCAL clr: ChooseColor
            . Si uMsg == WM_INITDIALOG
            . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                    mov eax, wParam
                                    . Si ax == IDCANCEL
                                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                    . Elseif ax == IDC_BACKCOLORBOX
                                                invocar RtlZeroMemory, addr clr, sizeof clr
                                                mov clr.lStructSize, sizeof clr
                                                impulsar hWnd
                                                clr.hwndOwner pop
                                                impulsar hInstance
                                                pop clr.hInstance
                                                impulsar BackgroundColor
                                                pop clr.rgbResult
                                                mov clr.lpCustColors, CustomColors de
compensación
                                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                                invocar ChooseColor, addr clr
                                                . Si eax! = 0
                                                            impulsar clr.rgbResult
                                                            pop BackgroundColor
                                                            invocar GetDlgItem, hWnd,
IDC_BACKCOLORBOX
                                                            invocar InvalidateRect, eax, 0, TRUE
                                                . Endif
                                    . Elseif ax == IDC_TEXTCOLORBOX
                                                invocar RtlZeroMemory, addr clr, sizeof clr
                                                mov clr.lStructSize, sizeof clr
                                                impulsar hWnd
                                                clr.hwndOwner pop
                                                impulsar hInstance
                                                pop clr.hInstance
                                                impulsar TextColor
                                                pop clr.rgbResult
                                                mov clr.lpCustColors, CustomColors de
compensación
                                                mov clr.Flags, CC_ANYCOLOR o CC_RGBINIT
                                                invocar ChooseColor, addr clr
                                                . Si eax! = 0
                                                            impulsar clr.rgbResult
                                                            pop TextColor
                                                            invocar GetDlgItem, hWnd,
IDC_TEXTCOLORBOX
                                                            invocar InvalidateRect, eax, 0, TRUE
                                                . Endif
                                    . Elseif ax == IDOK
```

```
                                        invoca SendMessage, hwndRichEdit,
EM_GETMODIFY, 0,0
                                        push eax
                                        invocar SetColor
                                        pop eax
                                        invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, eax, 0
                                        invocar EndDialog, hWnd, 0
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CTLCOLORSTATIC
                        invocar GetDlgItem, hWnd, IDC_BACKCOLORBOX
                        . Si eax == lParam
                                invocar CreateSolidBrush, BackgroundColor
                                ret
                        . Más
                                invocar GetDlgItem, hWnd, IDC_TEXTCOLORBOX
                                . Si eax == lParam
                                        invocar CreateSolidBrush, TextColor
                                        ret
                                . Endif
                        . Endif
                        mov eax, FALSO
                        ret
                . Elseif uMsg == WM_CLOSE
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
        OptionProc endp

        SearchProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
                . Si uMsg == WM_INITDIALOG
                        impulsar hWnd
                        pop hsearch
                        invocar CheckRadioButton, hWnd, IDC_DOWN, IDC_UP,
IDC_DOWN
                        invocar SendDlgItemMessage, hWnd, IDC_FINDEDIT,
WM_SETTEXT, 0, addr FindBuffer
                . Elseif uMsg == WM_COMMAND
                        mov eax, wParam
                        SHR EAX, 16
                        . Si ax == BN_CLICKED
                                mov eax, wParam
                                . Si ax == IDOK
                                        mov uFlags, 0
                                        invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                        invocar GetDlgItemText, hWnd,
IDC_FINDEDIT, FindBuffer addr, sizeof FindBuffer
                                        . Si eax! = 0
                                                invocar IsDlgButtonChecked, hWnd,
IDC_DOWN
                                                . Si eax == BST_CHECKED
```

```
                                                o uFlags, FR_DOWN
                                                mov eax,
findtext.chrg.cpMin
                                                . Si eax! =
Findtext.chrg.cpMax
                                                        impulsar
findtext.chrg.cpMax
                                                        pop
findtext.chrg.cpMin
                                                . Endif
                                                mov findtext.chrg.cpMax, -
1
                                        . Más
                                                mov findtext.chrg.cpMax, 0
                                        . Endif
                                        invocar IsDlgButtonChecked, hWnd,
IDC_MATCHCASE
                                        . Si eax == BST_CHECKED
                                                o uFlags, FR_MATCHCASE
                                        . Endif
                                        invocar IsDlgButtonChecked, hWnd,
IDC_WHOLEWORD
                                        . Si eax == BST_CHECKED
                                                o uFlags, FR_WHOLEWORD
                                        . Endif
                                        mov findtext.lpstrText,
desplazamiento FindBuffer
                                        invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, uFlags, addr FindText
                                        . Si eax! = -1
                                                invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                        . Endif
                                . Endif
                        . Elseif ax == IDCANCEL
                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                        . Más
                                mov eax, FALSO
                                ret
                        . Endif
                . Endif
        . Elseif uMsg == WM_CLOSE
                mov hsearch, 0
                invocar EndDialog, hWnd, 0
        . Más
                mov eax, FALSO
                ret
        . Endif
        mov eax, TRUE
        ret
  SearchProc endp

  ReplaceProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
        LOCAL setText: SETTEXTEX
        . Si uMsg == WM_INITDIALOG
                impulsar hWnd
                pop hsearch
```

```
                invocar SetDlgItemText, hWnd, IDC_FINDEDIT, addr
FindBuffer
                invocar SetDlgItemText, hWnd, IDC_REPLACEEDIT, addr
ReplaceBuffer
        . Elseif uMsg == WM_COMMAND
                mov eax, wParam
                SHR EAX, 16
                . Si ax == BN_CLICKED
                        mov eax, wParam
                        . Si ax == IDCANCEL
                                invoca SendMessage, hWnd, WM_CLOSE, 0,0
                        . Elseif ax == IDOK
                                invocar GetDlgItemText, hWnd,
IDC_FINDEDIT, FindBuffer addr, sizeof FindBuffer
                                invocar GetDlgItemText, hWnd,
IDC_REPLACEEDIT, ReplaceBuffer addr, sizeof ReplaceBuffer
                                mov findtext.chrg.cpMin, 0
                                mov findtext.chrg.cpMax, -1
                                mov findtext.lpstrText, desplazamiento
FindBuffer
                                mov settext.flags, ST_SELECTION
                                mov settext.codepage, CP_ACP
                                . Mientras VERDADERO
                                        invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, FR_DOWN, addr FindText
                                        . Si eax == -1
                                                . Descanso
                                        . Más
                                                invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                invoca SendMessage,
hwndRichEdit, EM_SETTEXTEX, setText addr, addr ReplaceBuffer
                                        . Endif
                                . Endw
                        . Endif
                . Endif
        . Elseif uMsg == WM_CLOSE
                mov hsearch, 0
                invocar EndDialog, hWnd, 0
        . Más
                mov eax, FALSO
                ret
        . Endif
        mov eax, TRUE
        ret
  ReplaceProc endp

  GoToProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
        LOCAL LineNo: DWORD
        LOCAL chrg: CHARRANGE
        . Si uMsg == WM_INITDIALOG
                impulsar hWnd
                pop hsearch
        . Elseif uMsg == WM_COMMAND
                mov eax, wParam
                SHR EAX, 16
                . Si ax == BN_CLICKED
```

```
                                mov eax, wParam
                                . Si ax == IDCANCEL
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Elseif ax == IDOK
                                        invocar GetDlgItemInt, hWnd, IDC_LINENO,
NULL, FALSE
                                        mov LineNo, eax
                                        invoca SendMessage, hwndRichEdit,
EM_GETLINECOUNT, 0,0
                                        . Si eax> LineNo
                                                invoca SendMessage, hwndRichEdit,
EM_LINEINDEX, LineNo, 0
                                                invoca SendMessage, hwndRichEdit,
EM_SETSEL, eax, eax
                                                invocar SetFocus, hwndRichEdit
                                        . Endif
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        mov hsearch, 0
                        invocar EndDialog, hWnd, 0
                . Más
                        mov eax, FALSO
                        ret
                . Endif
                mov eax, TRUE
                ret
        GoToProc endp

        PrepareEditMenu hSubMenu proc: DWORD
                LOCAL chrg: CHARRANGE
                invoca SendMessage, hwndRichEdit, EM_CANPASTE, CF_TEXT, 0
                . Si eax == 0; no hay texto en el portapapeles
                        invocar EnableMenuItem, hSubMenu, IDM_PASTE, MF_GRAYED
                . Más
                        invocar EnableMenuItem, hSubMenu, IDM_PASTE,
MF_ENABLED
                . Endif
                invoca SendMessage, hwndRichEdit, EM_CANUNDO, 0,0
                . Si eax == 0
                        invocar EnableMenuItem, hSubMenu, IDM_UNDO, MF_GRAYED
                . Más
                        invocar EnableMenuItem, hSubMenu, IDM_UNDO,
MF_ENABLED
                . Endif
                invoca SendMessage, hwndRichEdit, EM_CANREDO, 0,0
                . Si eax == 0
                        invocar EnableMenuItem, hSubMenu, IDM_REDO, MF_GRAYED
                . Más
                        invocar EnableMenuItem, hSubMenu, IDM_REDO,
MF_ENABLED
                . Endif
                invoca SendMessage, hwndRichEdit, EM_EXGETSEL, 0, addr chrg
                mov eax, chrg.cpMin
                . Si eax == chrg.cpMax; ninguna selección actual
                        invocar EnableMenuItem, hSubMenu, IDM_COPY, MF_GRAYED
                        invocar EnableMenuItem, hSubMenu, IDM_CUT, MF_GRAYED
```

```
                    invocar EnableMenuItem, hSubMenu, IDM_DELETE,
MF_GRAYED
            . Más
                    invocar EnableMenuItem, hSubMenu, IDM_COPY, MF_ENABLED
                    invocar EnableMenuItem, hSubMenu, IDM_CUT, MF_ENABLED
                    invocar EnableMenuItem, hSubMenu, IDM_DELETE,
MF_ENABLED
            . Endif
            ret
  PrepareEditMenu endp

  Proc ParseBuffer utiliza edi esi hHeap: DWORD, pBuffer: DWORD, nSize: DWORD,
ArrayOffset: DWORD, pArray: DWORD
            Búfer local [128]: BYTE
            LOCAL InProgress: DWORD
            mov InProgress, FALSO
            esi lea, tampón
            MOV EDI, pBuffer
            invocar CharLower, edición
            mov ecx, nSize
  SearchLoop:
            o ecx, ecx
            jz final
            cmp byte ptr [edi], ""
            je EndOfWord
            cmp byte ptr [edi], 9; ficha
            je EndOfWord
            mov InProgress, TRUE
            mov al, byte ptr [edi]
            mov byte ptr [esi], al
            inc esi
  SkipIt:
            inc edi
            diciembre ecx
            jmp SearchLoop
  EndOfWord:
            cmp InProgress, TRUE
            je WordFound
            jmp SkipIt
  WordFound:
            mov byte ptr [esi], 0
            empujar ecx
            invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, sizeof WORDINFO
            impulsar esi
            mov esi, eax
            asumir esi: ptr WORDINFO
            invocar lstrlen, addr tampón
            mov [esi]. WordLen, eax
            impulsar ArrayOffset
            pop [esi]. pcolor
            inc eax
            invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, eax
            mov [esi]. pszWord, eax
            mov edx, eax
            invocar lstrcpy, edx, addr tampón
            mov eax, pArray
            movzx edx, byte ptr [buffer]
            SHL edx, 2, multiplique por 4
```

```
            add eax, edx
            . Si dword ptr [eax] == 0
                        mov dword ptr [eax], esi
            . Más
                        push dword ptr [eax]
                        pop [esi]. Nextlink
                        mov dword ptr [eax], esi
            . Endif
            esi pop
            pop ecx
            esi lea, tampón
            mov InProgress, FALSO
            jmp SkipIt
  Final:
            . Si InProgress == TRUE
                        invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, sizeof
WORDINFO
                        impulsar esi
                        mov esi, eax
                        asumir esi: ptr WORDINFO
                        invocar lstrlen, addr tampón
                        mov [esi]. WordLen, eax
                        impulsar ArrayOffset
                        pop [esi]. pcolor
                        inc eax
                        invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, eax
                        mov [esi]. pszWord, eax
                        mov edx, eax
                        invocar lstrcpy, edx, addr tampón
                        mov eax, pArray
                        movzx edx, byte ptr [buffer]
                        SHL edx, 2, multiplique por 4
                        add eax, edx
                        . Si dword ptr [eax] == 0
                                    mov dword ptr [eax], esi
                        . Más
                                    push dword ptr [eax]
                                    pop [esi]. Nextlink
                                    mov dword ptr [eax], esi
                        . Endif
                        esi pop
            . Endif
            ret
  ParseBuffer endp

  FillHiliteInfo proc utiliza EDI
            Búfer local [1024]: BYTE
            LOCAL pTemp: DWORD
            LOCAL BlockSize: DWORD
            invocar RtlZeroMemory, addr ASMSyntaxArray, sizeof ASMSyntaxArray
            invocar GetModuleFileName, hInstance, addr buffer, sizeof tampón
            invocar lstrlen, addr tampón
            mov ecx, eax
            diciembre ecx
            Lea edición, tampón
            añadir EDI, ECX
            enfermedades de transmisión sexual
            mov al, "\"
```

```
            REPNE SCASB
            EPC
            inc edi
            mov byte ptr [edi], 0
            invocar lstrcat, addr buffer, addr WordFileName
            invocar GetFileAttributes, addr tampón
            . Si eax! = -1
                        mov BlockSize, 1024 * 10
                        invocar HeapAlloc, hMainHeap, 0, BlockSize
                        mov pTemp, eax
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C1Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                    inc eax
                                    . Si eax == BlockSize, el búfer es demasiado pequeño
                                                añadir BlockSize, 1024 * 10
                                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                                mov pTemp, eax
                                                jmp @ B
                                    . Endif
                                    mov edx, offset ASMColorArray
                                    invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C2Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                    inc eax
                                    . Si eax == BlockSize, el búfer es demasiado pequeño
                                                añadir BlockSize, 1024 * 10
                                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                                mov pTemp, eax
                                                jmp @ B
                                    . Endif
                                    mov edx, offset ASMColorArray
                                    añadir edx, 4
                                    invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C3Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                    inc eax
                                    . Si eax == BlockSize, el búfer es demasiado pequeño
                                                añadir BlockSize, 1024 * 10
                                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                                mov pTemp, eax
                                                jmp @ B
                                    . Endif
                                    mov edx, offset ASMColorArray
                                    añadir EDX, 8
```

```
                        invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                . Endif
  @ @:
                invocar GetPrivateProfileString, ASMSection addr, C4Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                . Si eax! = 0
                        inc eax
                        . Si eax == BlockSize, el búfer es demasiado pequeño
                                añadir BlockSize, 1024 * 10
                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                mov pTemp, eax
                                jmp @ B
                        . Endif
                        mov edx, offset ASMColorArray
                        añadir edx, 12
                        invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                . Endif
  @ @:
                invocar GetPrivateProfileString, ASMSection addr, C5Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                . Si eax! = 0
                        inc eax
                        . Si eax == BlockSize, el búfer es demasiado pequeño
                                añadir BlockSize, 1024 * 10
                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                mov pTemp, eax
                                jmp @ B
                        . Endif
                        mov edx, offset ASMColorArray
                        añadir edx, 16
                        invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                . Endif
  @ @:
                invocar GetPrivateProfileString, ASMSection addr, C6Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                . Si eax! = 0
                        inc eax
                        . Si eax == BlockSize, el búfer es demasiado pequeño
                                añadir BlockSize, 1024 * 10
                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                mov pTemp, eax
                                jmp @ B
                        . Endif
                        mov edx, offset ASMColorArray
                        añadir EDX, 20
                        invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                . Endif
  @ @:
                invocar GetPrivateProfileString, ASMSection addr, C7Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                . Si eax! = 0
```

```
                                inc eax
                                . Si eax == BlockSize, el búfer es demasiado pequeño
                                        añadir BlockSize, 1024 * 10
                                        invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                        mov pTemp, eax
                                        jmp @ B
                                . Endif
                                mov edx, offset ASMColorArray
                                añadir edx, 24
                                invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C8Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                inc eax
                                . Si eax == BlockSize, el búfer es demasiado pequeño
                                        añadir BlockSize, 1024 * 10
                                        invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                        mov pTemp, eax
                                        jmp @ B
                                . Endif
                                mov edx, offset ASMColorArray
                                añadir edx, 28
                                invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C9Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                inc eax
                                . Si eax == BlockSize, el búfer es demasiado pequeño
                                        añadir BlockSize, 1024 * 10
                                        invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                        mov pTemp, eax
                                        jmp @ B
                                . Endif
                                mov edx, offset ASMColorArray
                                añadir edx, 32
                                invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
  @ @:
                        invocar GetPrivateProfileString, ASMSection addr, C10Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                        . Si eax! = 0
                                inc eax
                                . Si eax == BlockSize, el búfer es demasiado pequeño
                                        añadir BlockSize, 1024 * 10
                                        invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                        mov pTemp, eax
                                        jmp @ B
```

```
                                . Endif
                                mov edx, offset ASMColorArray
                                añadir edx, 36
                                invocar ParseBuffer, hMainHeap, pTemp, eax, edx,
addr ASMSyntaxArray
                        . Endif
                        invocar HeapFree, hMainHeap, 0, pTemp
                . Endif
                ret
    FillHiliteInfo endp

    NewRichEditProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
                LOCAL hdc: DWORD
                LOCAL hOldFont: DWORD
                LOCAL aPartirDe: DWORD
                LOCAL rect: RECT
                LOCAL txtrange: TextRange
                Búfer local [1024 * 10]: BYTE
                LOCAL hrgn: DWORD
                LOCAL hOldRgn: DWORD
                LOCAL RealRect: RECT
                LOCAL PString: DWORD
                LOCAL BufferSize: DWORD
                Locales del PT: PUNTO
                . Si uMsg == WM_PAINT
                        impulsar edición
                        impulsar esi
                        invocar HideCaret, hWnd
                        invocar CallWindowProc, OldWndProc, hWnd, uMsg, wParam,
lParam
                        push eax
                        MOV EDI, ASMSyntaxArray desplazamiento
                        invocar GetDC, hWnd
                        mov hdc, eax
                        invocar SetBkMode, hdc, TRANSPARENTE
                        invoca SendMessage, hWnd, EM_GETRECT, 0, addr rect
                        invoca SendMessage, hWnd, EM_CHARFROMPOS, 0, addr rect
                        invoca SendMessage, hWnd, EM_LINEFROMCHAR, eax, 0
                        invoca SendMessage, hWnd, EM_LINEINDEX, eax, 0
                        mov txtrange.chrg.cpMin, eax
                        mov aPartirDe, eax
                        invoca SendMessage, hWnd, EM_CHARFROMPOS, 0, addr
rect.right
                        mov txtrange.chrg.cpMax, eax
                        impulsar rect.left
                        pop RealRect.left
                        impulsar rect.top
                        pop RealRect.top
                        impulsar rect.right
                        pop RealRect.right
                        impulsar rect.bottom
                        pop RealRect.bottom
                        invocar CreateRectRgn, RealRect.left, RealRect.top,
RealRect.right, RealRect.bottom
                        mov hrgn, eax
                        invocar SelectObject, hdc, hrgn
                        mov hOldRgn, eax
```

```
                    invocar SetTextColor, hdc, CommentColor
                    lea eax, tampón
                    mov txtrange.lpstrText, eax
                    invoca SendMessage, hWnd, EM_GETTEXTRANGE, 0, addr
txtrange
                    . Si eax> 0
                              mov esi, eax; esi == tamaño del texto
                              mov BufferSize, eax
                              impulsar edición
                              PUSH EBX
                              Lea edición, tampón
                              mov edx, edición, que se utiliza como punto de
referencia
                              mov ecx, esi
                              mov al, ";"
  ScanMore:
                              REPNE SCASB
                              je NextSkip
                              jmp NoMoreHit
  NextSkip:
                              diciembre edición
                              inc ecx
                              mov PString, edi
                              mov ebx, edi
                              sub ebx, edx
                              añadir ebx, aPartirDe
                              mov txtrange.chrg.cpMin, ebx
                              push eax
                              mov al, 0Dh
                              REPNE SCASB
                              pop eax
  HiliteTheComment:
                              . Si ecx> 0
                                        mov byte ptr [edi-1], 0
                              . Endif
                              mov ebx, edi
                              sub ebx, edx
                              añadir ebx, aPartirDe
                              mov txtrange.chrg.cpMax, ebx
                              pushad
                              MOV EDI, PString
                              mov esi, txtrange.chrg.cpMax
                              esi sub, txtrange.chrg.cpMin; esi contiene la longitud
del buffer
                              mov eax, esi
                              impulsar edición
                              . Mientras eax> 0
                                        . Si byte ptr [edi] == 9
                                                  mov byte ptr [edi], 0
                                        . Endif
                                        inc edi
                                        diciembre eax
                              . Endw
                              edición del pop
                              . Mientras esi> 0
                                        . Si byte ptr [edi]! = 0
                                                  invocar lstrlen, edición
                                                  push eax
```

```
                                                mov ecx, edi
                                                lea edx, tampón
                                                sub ecx, edx
                                                añadir ecx, aPartirDe
                                                . Si RichEditVersion == 3
                                                        invoca SendMessage,
hWnd, EM_POSFROMCHAR, addr rect, ecx
                                                . Más
                                                        invoca SendMessage,
hWnd, EM_POSFROMCHAR, ecx, 0
                                                        mov ecx, eax
                                                        y ECX, 0FFFFh
                                                        mov rect.left, ecx
                                                        SHR EAX, 16
                                                        mov rect.top, eax
                                                . Endif
                                                invocar DrawText, hdc, edición, -1,
rect addr, 0
                                                pop eax
                                                añadir edi, eax
                                                esi sub eax
                                        . Más
                                                inc edi
                                                diciembre esi
                                        . Endif
                                . Endw
                                mov ecx, txtrange.chrg.cpMax
                                sub ecx, txtrange.chrg.cpMin
                                invocar RtlZeroMemory, PString, ecx
                                POPAD
                                . Si ecx> 0
                                        jmp ScanMore
                                . Endif
  NoMoreHit:
                                pop ebx
                                edición del pop
                                mov ecx, BufferSize
                                esi lea, tampón
                                . Mientras ecx> 0
                                        mov al, byte ptr [esi]
                                        . Si al == "" | | == al 0Dh | | Al == "/" | |
Al == "" | | Al == "|" | | == al "+" | | Al == "-" | | Al == "*" | | Al == "y" | | Al
== "<" | | Al == ">" | | == al "=" | | Al == "( "| | Al ==") "| | Al ==" {"| | ==
al"} "| | Al ==" ["| | Al =="] "| | Al ==" ^ "| | Al == "" | | == al 9
                                                mov byte ptr [esi], 0
                                        . Endif
                                        diciembre ecx
                                        inc esi
                                . Endw
                                esi lea, tampón
                                mov ecx, BufferSize
                                . Mientras ecx> 0
                                        mov al, byte ptr [esi]
                                        . Si al! = 0
                                                empujar ecx
                                                invocar lstrlen, esi
                                                push eax
```

```
                                                mov edx, eax; edx contiene la
longitud de la cadena

                                                movzx eax, byte ptr [esi]
                                                . Si al> = "A" al && <= "Z"
                                                        sub al, "A"
                                                        añadir otros, "a"
                                                . Endif
                                                SHL eax, 2
                                                add eax, edición, edición contiene el
puntero a la matriz de punteros WORDINFO

                                                . Si dword ptr [eax]! = 0
                                                        mov eax, dword ptr [eax]
                                                        asumir eax: ptr
WORDINFO

                                                        . Mientras eax! = 0
                                                                . Si edx == [eax].
WordLen
                                                                        pushad
                                                                        invocar
lstrcmpi, [eax]. pszWord, esi
                                                                        . Si eax
== 0

POPAD

mov ecx, esi

lea edx, tampón

sub ecx, edx

añadir ecx, aPartirDe

pushad

. Si RichEditVersion == 3
                                                                                invoca S

. Más

                                                                                invoca S
                                                                                mov ecx
                                                                                y ECX, 0
                                                                                mov rec
                                                                                SHR EAX
                                                                                mov rec

. Endif

POPAD

mov edx, [eax]. pcolor

invocar SetTextColor, hdc, dword ptr [edx]

invocar DrawText, hdc, esi, -1, rect addr, 0

. Descanso
                                                                        . Endif
```

```
                                                                    POPAD
                                                                . Endif
                                                                pulse el botón
[eax]. Nextlink
                                                                pop eax
                                                            . Endw
                                                        . Endif
                                                        pop eax
                                                        pop ecx
                                                        añadir esi, eax
                                                        sub ecx, eax
                                                    . Más
                                                        inc esi
                                                        diciembre ecx
                                                    . Endif
                                                . Endw
                                            . Endif
                                            invocar SelectObject, hdc, hOldRgn
                                            invocar DeleteObject, hrgn
                                            invocar SelectObject, hdc, hOldFont
                                            invocar ReleaseDC, hWnd, hdc
                                            invocar ShowCaret, hWnd
                                            pop eax
                                            esi pop
                                            edición del pop
                                            ret
                                    . Elseif uMsg == WM_CLOSE
                                            invocar SetWindowLong, hWnd, GWL_WNDPROC, OldWndProc
                                    . Más
                                            invocar CallWindowProc, OldWndProc, hWnd, uMsg, wParam,
lParam
                                            ret
                                    . Endif
    NewRichEditProc endp

    WndProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam: DWORD
                LOCAL de n: OPENFILENAME
                Búfer local [256]: BYTE
                LOCAL editstream: EDITSTREAM
                LOCAL hFile: DWORD
                LOCAL hPopup: DWORD
                Locales del PT: PUNTO
                LOCAL chrg: CHARRANGE
                . Si uMsg == WM_CREATE
                            invocar CreateWindowEx, WS_EX_CLIENTEDGE, addr
RichEditClass, 0, o WS_CHILD WS_VISIBLE o ES_MULTILINE o WS_VSCROLL o
WS_HSCROLL o estilo ES_NOHIDESEL, \
                                                CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, hWnd, RichEditID, hInstance, 0
                            mov hwndRichEdit, eax
                            invoca SendMessage, hwndRichEdit,
EM_SETTYPOGRAPHYOPTIONS, TO_SIMPLELINEBREAK, TO_SIMPLELINEBREAK
                            invoca SendMessage, hwndRichEdit,
EM_GETTYPOGRAPHYOPTIONS, 1,1
                            . Si eax == 0; significa este mensaje no se procesa
                                        mov RichEditVersion, 2
                            . Más
                                        mov RichEditVersion, 3
```

```
                        invoca SendMessage, hwndRichEdit,
EM_SETEDITSTYLE, SES_EMULATESYSEDIT, SES_EMULATESYSEDIT
                    . Endif
                    invocar SetWindowLong, hwndRichEdit, GWL_WNDPROC,
NewRichEditProc addr
                    mov OldWndProc, eax
                    invoca SendMessage, hwndRichEdit, EM_LIMITTEXT, -1,0
                    invocar SetColor
                    invoca SendMessage, hwndRichEdit, EM_SETMODIFY, FALSE, 0
                    invoca SendMessage, hwndRichEdit, EM_SETEVENTMASK, 0,
ENM_MOUSEEVENTS
                    invoca SendMessage, hwndRichEdit, EM_EMPTYUNDOBUFFER,
0,0
            . UMsg elseif == WM_NOTIFY
                    impulsar esi
                    mov esi, lParam
                    asumir esi: ptr NMHDR
                    . Si [esi]. == Código EN_MSGFILTER
                            asumir esi: ptr msgfilter
                            . Si [esi]. Msg == WM_RBUTTONDOWN
                                    invocar GetMenu, hWnd
                                    invocar GetSubMenu, eax, 1
                                    mov hPopup, eax
                                    invocar PrepareEditMenu, hPopup
                                    mov edx, [esi]. lParam
                                    mov ecx, edx
                                    y edx, 0FFFFh
                                    shr ecx, 16
                                    mov pt.x, edx
                                    mov pt.y, ecx
                                    invocar ClientToScreen, hWnd, addr pt
                                    invocar TrackPopupMenu, hPopup,
TPM_LEFTALIGN o TPM_BOTTOMALIGN, pt.x, pt.y, NULL, hWnd, NULL
                            . Endif
                    . Endif
                    esi pop
            . UMsg elseif == WM_INITMENUPOPUP
                    mov eax, lParam
                    . Si ax == 0; menú de archivo
                            . Si FileOpened == TRUE, un archivo ya está abierto
                                    invocar EnableMenuItem, wParam,
IDM_OPEN, MF_GRAYED
                                    invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_ENABLED
                                    invocar EnableMenuItem, wParam,
IDM_SAVE, MF_ENABLED
                                    invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_ENABLED
                            . Más
                                    invocar EnableMenuItem, wParam,
IDM_OPEN, MF_ENABLED
                                    invocar EnableMenuItem, wParam,
IDM_CLOSE, MF_GRAYED
                                    invocar EnableMenuItem, wParam,
IDM_SAVE, MF_GRAYED
                                    invocar EnableMenuItem, wParam,
IDM_SAVEAS, MF_GRAYED
                            . Endif
```

```
                . Elseif ax == 1; menú de edición
                        invocar PrepareEditMenu, wParam
                . Elseif ax == 2; barra de búsqueda del menú
                        . Si FileOpened == TRUE
                                invocar EnableMenuItem, wParam,
IDM_FIND, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_FINDNEXT, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_FINDPREV, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_REPLACE, MF_ENABLED
                                invocar EnableMenuItem, wParam,
IDM_GOTOLINE, MF_ENABLED
                        . Más
                                invocar EnableMenuItem, wParam,
IDM_FIND, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_FINDNEXT, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_FINDPREV, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_REPLACE, MF_GRAYED
                                invocar EnableMenuItem, wParam,
IDM_GOTOLINE, MF_GRAYED
                        . Endif
                . Endif
        . Elseif uMsg == WM_COMMAND
                . Si lParam == 0; comandos de menú
                        mov eax, wParam
                        . Si ax == IDM_OPEN
                                invocar RtlZeroMemory, addr de n, sizeof
OFN
                                mov ofn.lStructSize, sizeof OFN
                                impulsar hWnd
                                ofn.hwndOwner pop
                                impulsar hInstance
                                pop ofn.hInstance
                                mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                mov ofn.lpstrFile, FileName desplazamiento
                                mov byte ptr [nombreDeArchivo], 0
                                mov ofn.nMaxFile, sizeof nombreDeArchivo
                                mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                invocar GetOpenFileName, addr OFN
                                . Si eax! = 0
                                        invocar CreateFile, addr FileName
GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0
                                        . Si eax! =
INVALID_HANDLE_VALUE
                                                mov hFile, eax
                                                mov editstream.dwCookie,
eax
                                                mov
editstream.pfnCallback, desplazamiento StreamInProc
```

```
                                                invoca SendMessage,
hwndRichEdit, EM_STREAMIN, SF_TEXT, addr editstream
                                                invoca SendMessage,
hwndRichEdit, EM_SETMODIFY, FALSE, 0
                                                invocar CloseHandle, hFile
                                                mov FileOpened, TRUE
                                        . Más
                                                invocar el cuadro de
mensajes, hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                        . Endif
                                . Endif
                        . Elseif ax == IDM_CLOSE
                                invocar CheckModifyState, hWnd
                                . Si eax == TRUE
                                        invocar SetWindowText,
hwndRichEdit, 0
                                        mov FileOpened, FALSO
                                . Endif
                        . Elseif ax == IDM_SAVE
                                invocar CreateFile, addr FileName
GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0
                                . Si eax! = INVALID_HANDLE_VALUE
   @ @:
                                        mov hFile, eax
                                        mov editstream.dwCookie, eax
                                        mov editstream.pfnCallback,
desplazamiento StreamOutProc
                                        invoca SendMessage, hwndRichEdit,
EM_STREAMOUT, SF_TEXT, addr editstream
                                        invoca SendMessage, hwndRichEdit,
EM_SETMODIFY, FALSE, 0
                                        invocar CloseHandle, hFile
                                . Más
                                        invocar el cuadro de mensajes,
hWnd, OpenFileFail addr, addr AppName, MB_OK o MB_ICONERROR
                                . Endif
                        . Elseif ax == IDM_COPY
                                invoca SendMessage, hwndRichEdit,
WM_COPY, 0,0
                        . Elseif ax == IDM_CUT
                                invoca SendMessage, hwndRichEdit,
WM_CUT, 0,0
                        . Elseif ax == IDM_PASTE
                                invoca SendMessage, hwndRichEdit,
WM_PASTE, 0,0
                        . Elseif ax == IDM_DELETE
                                invoca SendMessage, hwndRichEdit,
EM_REPLACESEL, TRUE, 0
                        . Elseif ax == IDM_SELECTALL
                                mov chrg.cpMin, 0
                                mov chrg.cpMax, -1
                                invoca SendMessage, hwndRichEdit,
EM_EXSETSEL, 0, addr chrg
                        . Elseif ax == IDM_UNDO
                                invoca SendMessage, hwndRichEdit,
EM_UNDO, 0,0
                        . Elseif ax == IDM_REDO
```

```
                                        invoca SendMessage, hwndRichEdit,
EM_REDO, 0,0
                                . Elseif ax == IDM_OPTION
                                        invocar DialogBoxParam, hInstance,
IDD_OPTIONDLG, hWnd, addr OptionProc, 0
                                . Elseif ax == IDM_SAVEAS
                                        invocar RtlZeroMemory, addr de n, sizeof
OFN
                                        mov ofn.lStructSize, sizeof OFN
                                        impulsar hWnd
                                        ofn.hwndOwner pop
                                        impulsar hInstance
                                        pop ofn.hInstance
                                        mov ofn.lpstrFilter, ASMFilterString
desplazamiento
                                        mov ofn.lpstrFile, AlternateFileName
desplazamiento
                                        mov byte ptr [AlternateFileName], 0
                                        mov ofn.nMaxFile, sizeof AlternateFileName
                                        mov ofn.Flags, OFN_FILEMUSTEXIST o
OFN_HIDEREADONLY o OFN_PATHMUSTEXIST
                                        invocar GetSaveFileName, addr OFN
                                        . Si eax! = 0
                                                invocar CreateFile,
AlternateFileName addr, GENERIC_WRITE, FILE_SHARE_READ, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
                                                . Si eax! =
INVALID_HANDLE_VALUE
                                                        jmp @ B
                                                . Endif
                                        . Endif
                                . Elseif ax == IDM_FIND
                                        . Si hsearch == 0
                                                invocar CreateDialogParam,
hInstance, IDD_FINDDLG, hWnd, addr SearchProc, 0
                                        . Endif
                                . Elseif ax == IDM_REPLACE
                                        . Si hsearch == 0
                                                invocar CreateDialogParam,
hInstance, IDD_REPLACEDLG, hWnd, addr ReplaceProc, 0
                                        . Endif
                                . Elseif ax == IDM_GOTOLINE
                                        . Si hsearch == 0
                                                invocar CreateDialogParam,
hInstance, IDD_GOTODLG, hWnd, addr GoToProc, 0
                                        . Endif
                                . Elseif ax == IDM_FINDNEXT
                                        invocar lstrlen, addr FindBuffer
                                        . Si eax! = 0
                                                invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                                mov eax, findtext.chrg.cpMin
                                                . Si eax! = Findtext.chrg.cpMax
                                                        impulsar
findtext.chrg.cpMax
                                                        pop findtext.chrg.cpMin
                                                . Endif
                                                mov findtext.chrg.cpMax, -1
```

```
                                                mov findtext.lpstrText,
desplazamiento FindBuffer
                                                invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, FR_DOWN, addr FindText
                                                . Si eax! = -1
                                                        invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                . Endif
                                        . Endif
                                . Elseif ax == IDM_FINDPREV
                                        invocar lstrlen, addr FindBuffer
                                        . Si eax! = 0
                                                invoca SendMessage, hwndRichEdit,
EM_EXGETSEL, 0, addr findtext.chrg
                                                mov findtext.chrg.cpMax, 0
                                                mov findtext.lpstrText,
desplazamiento FindBuffer
                                                invoca SendMessage, hwndRichEdit,
EM_FINDTEXTEX, 0, addr FindText
                                                . Si eax! = -1
                                                        invoca SendMessage,
hwndRichEdit, EM_EXSETSEL, 0, addr findtext.chrgText
                                                . Endif
                                        . Endif
                                . Elseif ax == IDM_EXIT
                                        invoca SendMessage, hWnd, WM_CLOSE, 0,0
                                . Endif
                        . Endif
                . Elseif uMsg == WM_CLOSE
                        invocar CheckModifyState, hWnd
                        . Si eax == TRUE
                                invocar DestroyWindow, hWnd
                        . Endif
                . Elseif uMsg == WM_SIZE
                        mov eax, lParam
                        mov edx, eax
                        y EAX, 0FFFFh
                        shr edx, 16
                        invocar MoveWindow, hwndRichEdit, 0,0, eax, edx, TRUE
                . Elseif uMsg == WM_DESTROY
                        invocar PostQuitMessage, NULL
                . Más
                        invocar DefWindowProc, hWnd, uMsg, wParam, lParam
                        ret
                . Endif
                xor eax, eax
                ret
        WndProc ENDP
        poner fin a empezar a
```

Análisis:

La primera acción antes de llamar a llamar a WinMain FillHiliteInfo. Esta función lee el contenido de wordfile.txt y analiza el contenido.

```
        FillHiliteInfo proc utiliza EDI
                Búfer local [1024]: BYTE
```

```
LOCAL pTemp: DWORD
LOCAL BlockSize: DWORD
invocar RtlZeroMemory, addr ASMSyntaxArray, sizeof ASMSyntaxArray
```

Inicializar ASMSyntaxArray a cero.

```
invocar GetModuleFileName, hInstance, addr buffer, sizeof tampón
        invocar lstrlen, addr tampón
        mov ecx, eax
        diciembre ecx
        Lea edición, tampón
        añadir EDI, ECX
        enfermedades de transmisión sexual
        mov al, "\"
        REPNE SCASB
        EPC
        inc edi
        mov byte ptr [edi], 0
        invocar lstrcat, addr buffer, addr WordFileName
```

Construir el nombre de ruta completo de wordfile.txt: Supongo que es siempre en la misma carpeta que el programa.

```
invocar GetFileAttributes, addr tampón
        . Si eax! = -1
```

Puedo utilizar este método como una forma rápida de comprobar si existe un archivo.

```
mov BlockSize, 1024 * 10
                invocar HeapAlloc, hMainHeap, 0, BlockSize
                mov pTemp, eax
```

Asignar el bloque de memoria para almacenar las palabras. Por defecto a 10 km. La memoria se asigna desde el montón predeterminado.

```
@ @:
                invocar GetPrivateProfileString, ASMSection addr, C1Key addr,
addr ZeroString, pTemp, de bloque, addr tampón
                . Si eax! = 0
```

Yo uso **GetPrivateProfileString** para recuperar el contenido de cada tecla en el wordfile.txt. La clave se inicia desde C1 a C10.

```
inc eax
                        . Si eax == BlockSize, el búfer es demasiado pequeño
                                añadir BlockSize, 1024 * 10
                                invocar HeapReAlloc, hMainHeap, 0, pTemp,
BlockSize
                                mov pTemp, eax
                                jmp @ B
                        . Endif
```

Comprobación de que el bloque de memoria es lo suficientemente grande. Si no es así, se incrementa el tamaño de 10K hasta que el bloque es suficientemente grande.

```
mov edx, offset ASMColorArray
```

invocar ParseBuffer, hMainHeap, pTemp, eax, edx, addr ASMSyntaxArray

Pasar las palabras, el mango bloque de memoria, el tamaño de los datos leídos de wordfile.txt, la dirección de la dword color que se utiliza para Selecciona los palabras y la dirección de **ASMSyntaxArray.**

Ahora, vamos a examinar lo que hace **ParseBuffer.** En esencia, esta función acepta el tampón que contiene las palabras que se hilighted, analiza a las palabras individuales y tiendas de cada uno de ellos en una amplia estructura de **WORDINFO** que se puede acceder rápidamente desde **ASMSyntaxArray.**

```
Proc ParseBuffer utiliza edi esi hHeap: DWORD, pBuffer: DWORD, nSize: DWORD,
ArrayOffset: DWORD, pArray: DWORD
        Búfer local [128]: BYTE
        LOCAL InProgress: DWORD
        mov InProgress, FALSO
```

InProgress es la bandera que se utiliza para indicar si el proceso de digitalización ha comenzado. Si el valor es FALSE, no hemos encontrado un carácter no espacio en blanco todavía.

```
esi lea, tampón
        MOV EDI, pBuffer
        invocar CharLower, edición
```

puntos ESI a nuestro buffer local que contendrá la palabra que hemos analizado de la lista de palabras. puntos de EDI a la cadena de la lista de palabras. Para simplificar la búsqueda más tarde, convertir todos los caracteres en minúsculas.

```
mov ecx, nSize
SearchLoop:
        o ecx, ecx
        jz final
        cmp byte ptr [edi], ""
        je EndOfWord
        cmp byte ptr [edi], 9; ficha
        je EndOfWord
```

Busque en la lista la palabra completa en el buffer, en busca de los espacios en blanco. Si un espacio en blanco se encuentra, tenemos que determinar si se marca el final o el principio de una palabra.

```
mov InProgress, TRUE
        mov al, byte ptr [edi]
        mov byte ptr [esi], al
        inc esi
SkipIt:
        inc edi
        diciembre ecx
        jmp SearchLoop
```

Si el byte bajo el escrutinio no es un espacio en blanco, lo copiamos a la memoria intermedia para la construcción de una palabra y luego continuar el recorrido.

```
EndOfWord:
```

```
cmp InProgress, TRUE
je WordFound
jmp SkipIt
```

Si un espacio en blanco se encuentra, compruebe el valor de **InProgress.** Si el valor es TRUE, podemos asumir que el espacio en blanco marca el final de una palabra y podamos proceder a poner la palabra actualmente en el búfer local (apuntado por ESI) en una estructura **WORDINFO.** Si el valor es FALSE, continuamos la exploración hasta que un personaje no espacio en blanco se encuentra.

```
WordFound:
        mov byte ptr [esi], 0
        empujar ecx
        invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, sizeof WORDINFO
```

Cuando el final de una palabra se encuentra, añadimos 0 al buffer para que la palabra una cadena ASCIIZ. A continuación, asignar un bloque de memoria del montón del tamaño de **WORDINFO** de esta palabra.

```
impulsar esi
        mov esi, eax
        asumir esi: ptr WORDINFO
        invocar lstrlen, addr tampón
        mov [esi]. WordLen, eax
```

Se obtiene la longitud de la palabra en el buffer local y almacenarlo en el miembro **WordLen** de la estructura **WORDINFO,** para ser utilizado como una comparación rápida.

```
impulsar ArrayOffset
        pop [esi]. pcolor
```

Guarde la dirección de la DWORD que contiene el color que se utiliza para Selecciona los miembros de la palabra en **pcolor.**

```
inc eax
        invocar HeapAlloc, hHeap, HEAP_ZERO_MEMORY, eax
        mov [esi]. pszWord, eax
        mov edx, eax
        invocar lstrcpy, edx, addr tampón
```

Asignar memoria del montón para almacenar la palabra misma. En este momento, la estructura **WORDINFO** está listo para ser insertado en la lista apropiada ligados.

```
mov eax, pArray
        movzx edx, byte ptr [buffer]
        SHL edx, 2, multiplique por 4
        add eax, edx
```

pArray contiene la dirección de **ASMSyntaxArray.** Queremos pasar a la dword que tiene el mismo índice como el valor del primer carácter de la palabra. Por eso, pusimos el primer carácter de la palabra en edx luego multiplicar edx por 4 (debido a que cada elemento en **ASMSyntaxArray** es de 4 bytes de tamaño) y luego añadir el desplazamiento a la dirección de **ASMSyntaxArray.** Tenemos la dirección de la DWORD correspondiente en eax.

```
. Si dword ptr [eax] == 0
                mov dword ptr [eax], esi
```

```
        . Más
                push dword ptr [eax]
                pop [esi]. Nextlink
                mov dword ptr [eax], esi
        . Endif
```

Compruebe el valor de DWORD. Si es 0, significa que actualmente no existe una palabra que empiece con este carácter en la lista. Por lo tanto poner la dirección de la estructura actual de esa **WORDINFO** DWORD.

Si el valor en el dword no es 0, significa que hay al menos una palabra que comienza con este carácter en la matriz. Por lo tanto insertar esta estructura **WORDINFO** a la cabeza de la lista enlazada y actualizar sus miembros Nextlink para que apunte a la estructura **WORDINFO** siguiente.

```
   esi pop
        pop ecx
        esi lea, tampón
        mov InProgress, FALSO
        jmp SkipIt
```

Después de la operación se ha completado, se inicia el ciclo siguiente hasta el final del búfer que se alcance.

```
   invoca SendMessage, hwndRichEdit, EM_SETTYPOGRAPHYOPTIONS,
TO_SIMPLELINEBREAK, TO_SIMPLELINEBREAK
                invoca SendMessage, hwndRichEdit,
EM_GETTYPOGRAPHYOPTIONS, 1,1
                . Si eax == 0; significa este mensaje no se procesa
                        mov RichEditVersion, 2
                . Más
                        mov RichEditVersion, 3
                        invoca SendMessage, hwndRichEdit,
EM_SETEDITSTYLE, SES_EMULATESYSEDIT, SES_EMULATESYSEDIT
                . Endif
```

Después de que el control RichEdit se crea, es necesario determinar la su versión. Este paso es necesario ya que **EM_POSFROMCHAR** comporta de manera diferente de RichEdit 2.0 y 3.0 y **EM_POSFROMCHAR** es crucial para nuestra rutina de resaltado de sintaxis. Nunca he visto una forma documentada de comprobar la versión del control RichEdit lo que tengo que usar una solución. En este caso, me puse una opción que es específica a la versión 3.0 e inmediatamente recuperar su valor. Si yo puedo recuperar el valor, supongo que la versión de control es de 3,0.

Si utiliza el control RichEdit versión 3.0, te darás cuenta de que la actualización del color de la fuente de un archivo de gran tamaño tiene un tiempo bastante largo. Este problema parece ser específica a la versión 3.0. He encontrado una solución: hacer que el control de emular el comportamiento del sistema de control de edición mediante el envío de mensajes **EM_SETEDITSTYLE.**

Después de que se puede obtener la información de la versión, se procede a la subclase el control RichEdit. Ahora vamos a examinar el nuevo procedimiento de ventana para el control RichEdit.

```
   NewRichEditProc proc hWnd: DWORD, uMsg: DWORD, wParam: DWORD, lParam:
DWORD
```

```
.....
.....
. Si uMsg == WM_PAINT
        impulsar edición
        impulsar esi
        invocar HideCaret, hWnd
        invocar CallWindowProc, OldWndProc, hWnd, uMsg, wParam,
lParam
        push eax
```

Nosotros nos encargamos de mensaje **WM_PAINT.** En primer lugar, ocultar el cursor con el fin de evitar algunos feos gfx después de que el resaltado de. Después de que pase el mensaje al procedimiento original de richedit dejar que se actualice la ventana. Cuando regresa **CallWindowProc,** el texto se actualiza con su color habitual / fondo. Ahora es nuestra oportunidad de hacer resaltado de sintaxis.

```
MOV EDI, ASMSyntaxArray desplazamiento
        invocar GetDC, hWnd
        mov hdc, eax
        invocar SetBkMode, hdc, TRANSPARENTE
```

Guarde la dirección de ASMSyntaxArray en EDI. A continuación, se obtiene el identificador de contexto de dispositivo y configurar el modo de fondo del texto a la transparencia por lo que el texto que vamos a escribir se utiliza el color de fondo predeterminado.

```
invoca SendMessage, hWnd, EM_GETRECT, 0, addr rect
        invoca SendMessage, hWnd, EM_CHARFROMPOS, 0, addr rect
        invoca SendMessage, hWnd, EM_LINEFROMCHAR, eax, 0
        invoca SendMessage, hWnd, EM_LINEINDEX, eax, 0
```

Se desea obtener el texto visible por lo que primero tiene que obtener el rectángulo de formato mediante el envío de mensajes **EM_GETRECT** al control RichEdit. Ahora que tenemos el rectángulo de delimitación, se obtiene el índice de carácter más cercano a la esquina superior izquierda del rectángulo con **EM_CHARFROMPOS.** Una vez que tenemos el índice de caracteres (el primer carácter visible en el control), podemos empezar a hacer resaltado de sintaxis a partir de esa posición. Sin embargo, el efecto podría no ser tan bueno como cuando comenzamos desde el primer carácter de la línea de que el personaje es pulg Es por eso que necesito para obtener el número de línea que el carácter visible primera es mediante el envío de mensajes de **EM_LINEFROMCHAR.** Para obtener el primer carácter de esa línea, puedo enviar mensajes **EM_LINEINDEX.**

```
mov txtrange.chrg.cpMin, eax
        mov aPartirDe, eax
        invoca SendMessage, hWnd, EM_CHARFROMPOS, 0, addr
rect.right
        mov txtrange.chrg.cpMax, eax
```

Una vez que tenemos el índice del primer carácter, la guarde para futuras referencias en la variable aPartirDe. A continuación se obtiene el último índice carácter visible mediante el envío de **EM_CHARFROMPOS,** pasando por la esquina inferior derecha del rectángulo de formato en **lParam.**

```
impulsar rect.left
        pop RealRect.left
        impulsar rect.top
        pop RealRect.top
```

```
                    impulsar rect.right
                    pop RealRect.right
                    impulsar rect.bottom
                    pop RealRect.bottom
                    invocar CreateRectRgn, RealRect.left, RealRect.top,
RealRect.right, RealRect.bottom
                    mov hrgn, eax
                    invocar SelectObject, hdc, hrgn
                    mov hOldRgn, eax
```

Mientras que hace resaltado de sintaxis, me di cuenta de una desagradable efecto secundario de este método: si el control RichEdit tiene un margen (se puede especificar el margen mediante el envío de **EM_SETMARGINS** mensaje al control RichEdit), **DrawText** escribe sobre el margen. Así que necesita para crear una región de recorte, el tamaño del rectángulo de formato, llamando **CreateRectRgn.** La salida de las funciones GDI será recortada a la "escritura" zona.

A continuación, necesitamos Selecciona los comentarios de primero y sacarlos de nuestro camino. Mi método es la búsqueda de ";" y seleccionalo el texto con el color de sus comentarios hasta el retorno de carro es hallado. No voy a analizar aquí la rutina: es bastante largo y complicado. Baste decir aquí que, cuando todos los comentarios sean hilighted, las sustituimos con 0s en el buffer de modo que las palabras en los comentarios no serán procesados / hilighted más tarde.

```
   mov ecx, BufferSize
                    esi lea, tampón
                    . Mientras ecx> 0
                              mov al, byte ptr [esi]
                              . Si al == "" | | == al 0Dh | | Al == "/" | | Al == "" |
| Al == "|" | | == al "+" | | Al == "-" | | Al == "*" | | Al == "y" | | Al == "<" | |
Al == ">" | | == al "=" | | Al == "( "| | Al ==") "| | Al ==" {"| | == al"} "| | Al
==" ["| | Al =="] "| | Al ==" ^ "| | Al == "" | | == al 9
                                        mov byte ptr [esi], 0
                              . Endif
                              diciembre ecx
                              inc esi
                    . Endw
```

Una vez que los comentarios están fuera de nuestro camino, separar las palabras en el buffer mediante la sustitución de los "Separador" personajes con 0. Con este método, no necesitamos preocuparnos acerca de los caracteres de separación durante el procesamiento de las palabras en la memoria intermedia más: sólo hay un carácter de separación, es NULL.

```
   esi lea, tampón
                    mov ecx, BufferSize
                    . Mientras ecx> 0
                              mov al, byte ptr [esi]
                              . Si al! = 0
```

Buscar en el búfer para el primer carácter que no es nulo, es decir, el primer carácter de una palabra.

```
   empujar ecx
                                        invocar lstrlen, esi
                                        push eax
                                        mov edx, eax
```

Obtener la longitud de la palabra y la puso en edx

```
   movzx eax, byte ptr [esi]
                                        . Si al> = "A" al && <= "Z"
                                                sub al, "A"
                                                añadir otros, "a"
                                        . Endif
```

Convierte el carácter a minúscula (si es un carácter en mayúscula)

```
   SHL eax, 2
                                        add eax, edición, edición contiene el puntero
a la matriz de punteros WORDINFO
                                        . Si dword ptr [eax]! = 0
```

Después de eso, nos saltamos a la DWORD correspondiente en ASMSyntaxArray y comprobar si el valor DWORD que es 0. Si es así, podemos pasar a la siguiente palabra.

```
   mov eax, dword ptr [eax]
                                                asumir eax: ptr WORDINFO
                                                . Mientras eax! = 0
                                                        . Si edx == [eax]. WordLen
```

Si el valor en el valor DWORD es distinto de cero, que apunta a la lista enlazada de estructuras **WORDINFO.** Procesamos a caminar la lista enlazada, comparando la longitud de la palabra en nuestro buffer local con la longitud de la palabra en la estructura **WORDINFO.** Esta es una prueba rápida antes de comparar las palabras. En caso de salvar a algunos ciclos de reloj.

```
   pushad
                                                invocar lstrcmpi,
[eax]. pszWord, esi
                                                . Si eax == 0
```

Si las longitudes de las dos palabras son iguales, se procede a compararlos con **lstrcmpi.**

```
   POPAD
                                                mov ecx,
esi
                                                lea edx,
tampón
                                                sub ecx,
edx
                                                añadir
ecx, aPartirDe
```

Se construye el índice de caracteres con la dirección del primer carácter de la palabra correspondiente en el buffer. En primer lugar, obtener su desplazamiento relativo de la dirección inicial de la memoria intermedia a continuación, añadir el índice de caracteres del carácter visible primero a ella.

```
   pushad
                                                . Si
RichEditVersion == 3

invoca SendMessage, hWnd, EM_POSFROMCHAR, addr rect, ecx
                                                . Más
```

invoca SendMessage, hWnd, EM_POSFROMCHAR, ecx, 0

mov ecx, eax

y ECX, 0FFFFh

mov rect.left, ecx

SHR EAX, 16

mov rect.top, eax

. Endif
POPAD

Una vez que conocemos el índice de carácter del primer carácter de la palabra que se hilighted, se procede a obtener las coordenadas de la misma mediante el envío de mensajes **EM_POSFROMCHAR.** Sin embargo, este mensaje se interpreta de manera diferente por RichEdit 2.0 y 3.0. Para RichEdit 2.0, **wParam** contiene el índice del carácter y **lParam** no se utiliza. Devuelve la coordenada en eax. Para RichEdit 3.0, **wParam** es el puntero a una estructura **POINT** que se llenará con la coordenada y **lParam** contiene el índice de carácter.

Como puede ver, pasando los argumentos equivocados para **EM_POSFROMCHAR** puede causar estragos en su sistema. Es por eso que hay que diferenciar entre las versiones de control RichEdit.

mov edx, [eax]. pcolor

invocar

SetTextColor, hdc, dword ptr [edx]

invocar

DrawText, hdc, esi, -1, rect addr, 0

Una vez que llegamos la coordenada para empezar, nos fijamos el color del texto con el que se especifica en la estructura **WORDINFO.** Y luego proceder a sobrescribir la palabra con el nuevo color.

Como las últimas palabras, este método puede mejorarse de varias maneras. Por ejemplo, obtener todo el texto a partir de la primera a la última línea visible. Si las líneas son muy largas, el rendimiento puede doler de la transformación de las palabras que no son visibles. Usted puede optimizar esta obteniendo la línea de texto muy visibles por línea. Asimismo, el algoritmo de búsqueda puede ser mejorada mediante el uso de un método más eficiente. No me malinterpreten: el método de resaltado de sintaxis utilizada en este ejemplo es rápido, pero puede ser más rápido. :)
