27

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGON

"INGENIERÍA INVERSA DE SOFTWARE E INFORMACIÓN"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

ROBERTO MARQUEZ BECERRIL

ASESOR :
ING. GLADIS FUENTES CHAVEZ

MÉXICO

2002

DEDICATORIAS Y AGRADECIMIENTOS

Doy gracias a **Dios** porque nunca me abandono en todas las situaciones que hubo en mi vida y estuvo conmigo en cada momento.

A ti Mama porque haz estado en toda mi trayectoria, siempre cuidándome, apoyándome en las buenas y en las malas y dándome consejos que siempre ocupe.

A mis Hermanos porque de ellos aprendí muchas cosas y he tomado lo mejor de ellos para ser una mejor persona, además de que casi toda mi vida convivi con ellos y son especiales para mi.

A Ing. Juan Moy Lara muchas gracias por la gran amistad y los grandes ratos que hemos convivido y los consejos que me haz brindado.

A mis Padres porque siempre estuvieron al pendiente, nunca me dejaron solo y me formaron buenos principios.

A ti Papa porque siempre me supiste guiar, me diste una gran herencia que son mis estudios y siempre haz estado orgulloso de mí.

A Ing. Gladis Fuentes Chávez gracias a la amistad que me haz brindado en el transcurso desde que te conozco y todo el apoyo que me haz ofrecido en mi base profesional y laboral.

A Ing. Adonai Ríos Rodríguez que puedo decir, es una gran persona que estuvo a lo largo de mi carrera, conviviendo y soportándome a lo largo de esta, es un gran amigo para mí.

A Ing. Oswaldo Reyes Nogueron gracias a ti aprendí muchas cosas nuevas que supe aprovechar en todo momento.

A Ing. Abel Herrera Corredor quien iba a pensar que con uno de mis amigos iba a estar en el ámbito laboral y fue una experiencia muy padre.

A Ing. Emmanuel Alcántara otra de las personas que convivió conmigo y siempre estuvo al pendiente.

A Ing. Ángel Castro porque siempre nos motivo a hacer cosas nuevas y a nunca sobajarnos, ni dejarnos de nadie.

A mi Asesora que puedo decir, fuiste la única persona que se quiso aventar al ruedo y apoyarme en todos los aspectos tanto emocionales como intelectuales, nunca pensé que fueras a batallar tanto conmigo y nunca te cansaste y estuviste al pendiente de mi, gracias por dejarme ser en mi tesis y compartir conmigo toda su gran experiencia que tienes y ubicarme bien en todos los aspectos.

A todas las personas que han estado en mi vida y han formado parte de ella, algunas amistades nuevas y otras ni tan nuevas pero ahí siguen, gracias a todos.

A mis Revisores Ing. Silvia Vega, Mat. Luis Ramírez, Ing. Ricardo Gutiérrez e Ing. Rodolfo Vázquez, por todo su apoyo.

Ing. Roberto Márquez porque siempre supiste salir adelante pese a las cosas que pasaban frente a ti, supiste tomar las cosas malas con inteligencia y ocuparlas en tu beneficio y tuviste el coraje de salir adelante en las cosas adversas, nunca cambies.

ÍNDICE

CAPITULO IV: LEYES DE PROTECCIÓN CONTRA LA PIRATERÍA

INTRODUCCIÓN

La ingeniería inversa se cataloga como una de las formas en que una persona puede resolver varios aspectos relevantes.
Todo esto conlleva a aspectos positivos o aspectos negativos dependiendo del uso que se le aplique, este método se utiliza no solamente en procesos de cómputo sino en varias áreas donde existan métodos adversos, en algunas ramas donde se puede utilizar esta técnica son: Medicina, Educación, Mecánica, Bioingeniería, Matemáticas, Graficación y Procesos lógicos entre otros.

Existe un sin fin de áreas donde podrías aplicar técnicas inversas para obtener mejores resultados o simplificar métodos.

Este campo no es muy estudiado, porque en algunos casos existen reglas o procesos de leyes que no permiten explorar algunas cosas, en la rama del software o sobre los datos, estos métodos son poco sugeridos ya que muchos programas tienen leyes que los protegen y tiene derechos de autor, y se podría decir que en estos casos se estaría cometiendo algunos delitos en los cuales estaría la piratería de recursos.

El gran uso de las técnicas como el desensamblado y el desencriptado de la información, son dos de las herramientas más poderosas que existen en la actualidad para el análisis de cualquier aplicación.

Sin embargo cada día se implementan un sin fin de utilidades que nos ayudan a proteger él contenido de estas, y he ahí que los métodos de protección y desprotección han aumentado en cuanto a técnicas las cuales cada vez se hacen mas eficientes en cualquier aspecto, sin embargo en la historia del software han surgido grandes herramientas en la protección de éstas.

Los casos más comunes son la implementación de rutinas, las cuales detectan los famosos descompiladores, estas cadenas ó rutinas tienen ciertas instrucciones, la mayoría proviene de secuencias en ensamblador y pueden ser insertadas mediante el lenguaje que se este ocupando o usando un método poco común el cual primero se genera el desensamblado y luego se inserta el fragmento de código que va a permitir la protección de nuestra aplicación.

Actualmente en la ingeniería inversa, la mayoría de su porcentaje se emplea para fines de lucro, ya que esto se debe a los altos costos que existen en los productos finales de cada fabricante, he ahí la problemática de la piratería y las técnicas antipiratería.

Estos costos sobre producto no son costeables por individuo, de hecho toda la gente prefiere usar copias piratas, en lugar de pagar estos altos costos.

Gracias a los grandes avances de la tecnología los costos por productos piratas casi son insignificantes, y van ganando mucho territorio en todo el mundo.

Los centros de investigación tiene que lidiar con muchas cosas, primero, implementar nuevas defensas para sus productos, tales como:

- La creación de nuevos algoritmos.
- La implantación de nuevas técnicas de encriptado.
- Métodos inteligentes detectores de debuggers.
- Camuflaje de información.
- Y la más común la fuga de información de la misma empresa.

En este trabajo de titulación se abarcan las cosas más significativas que existen en el mercado y las problemáticas que hay en cuanto a la piratería.

En la ingeniería inversa es muy interesante analizar las cosas en forma de partir de un producto final, ya que me he encontrado al estar analizando aplicaciones con muchas cosas desconocidas, que de hecho si eres el dueño del mismo programa que estas estudiando veras que puedes ver diferentes aspectos de programación.

Analizando todos estos temas, que tal vez sean muy complejos, pero a la vez divertidos y fascinantes, encuentras muchas terminologías y muchos conceptos, además es muy interesante ver las propiedades de otras aplicaciones y empezar a estudiar los posibles fallos que podría tener, si requiere de un sistema de protección o reconstruir una sección.

También se muestran partes muy interesantes, la primera, y fundamental, es la forma que tiene la Ingeniería Inversa y toda la terminología que esto implica conocer, existen muchos conceptos dependiendo del tema que se trate de Ingeniería Inversa, en este caso como estamos basados en la programación y software sus definiciones son muy amplias.

Además de las definiciones, es necesario saber porque se debe de aplicar los procesos inversos, si son necesarios y en que se deben aplicar.

Casi la mayoría de los programas tienen Ingeniería Inversa de acuerdo el lenguaje en que fueron creados, en algunos lenguajes, su inversa la proporciona el lenguaje ensamblador y otros solo son creados para interpretar instrucciones.

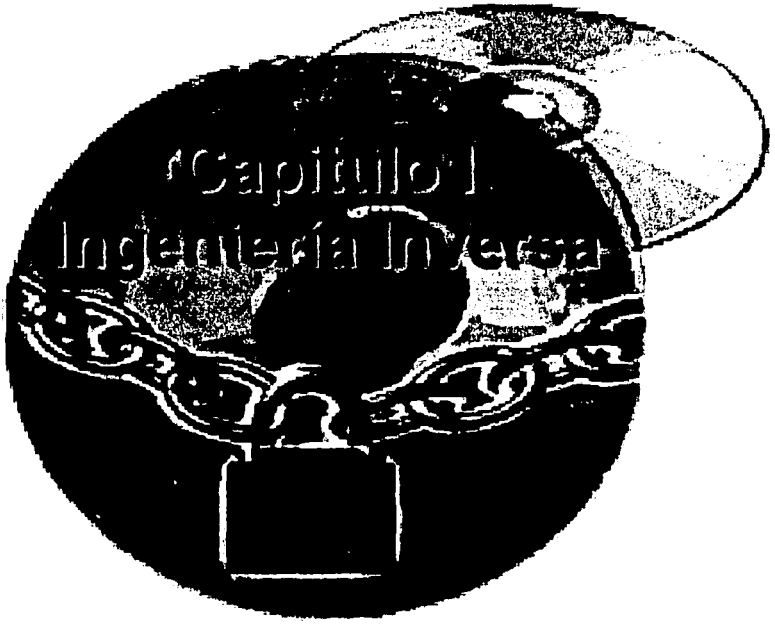
Aquí se retoman aquellos compiladores que no son intérpretes de lenguaje si no que en realidad crean un código ejecutable, a todos estos programas ejecutables después de ser desensamblados se les puede insertar código adicional el cual podría servir para mejorar características de el, o simplemente proteger su sistema de datos.

Las referencias del lenguaje ensamblador, los conceptos y las rutinas que se explican, son fundamentales para el entendimiento idóneo de los temas relacionados con la ingeniería inversa.

Los conocimientos básicos, medios y avanzados sobre el lenguaje ensamblador van a brindar una serie de ventajas en el análisis de los programas que se estén estudiando ya que se podría decir que es la base de la ingeniería inversa.

También se manejan las diferentes formas de protección que hay, no son todas las que existen pero son las mas comerciales o las mas populares que se ofrecen en el mercado, creo que el saber como funciona estas técnicas de protección anti-pirateria, nos da un amplio margen de saber como funciona estas técnicas, aparte de saber como funciona nos muestran muchas cosas relevantes que podríamos implementar en nuestras aplicaciones y tal vez mejorar estas técnicas o crear nuestro propio kit de protección. Estos elementos y técnicas analizados se pueden retomar para ejemplificar los diferentes tipos de investigación educativa.

En la actualidad hablar de leyes que rigen la propiedad intelectual, es un tema complejo y con poco fundamento, pero que a su vez se va modernizando, ya que no se tiene un margen concreto que especifique bien las multas que se deben de requerir, existen asociaciones que se interesan sobre los derechos de autor y sobre el software, entre otros, dichas asociaciones son organismos mundiales que ayudan que las garantías del autor sean respetadas, esto se podrá apreciar en el último capítulo de ésta tesis.


Capítulo I
Ingeniería Inversa

INGENIERÍA INVERSA

La ingeniería inversa se basa en quitar, remover o suspender una o mas técnicas de protección de algunas aplicaciones, ya sean comerciales, educacionales u otras. Muchos consideran esto como un arte.

Algunos términos como "Cracking " o "Reverse Engineering" son específicos del idioma ingles y no tiene una correcta traducción al español. Por lo que el termino "Reverse Engineering" se Traduce Como "Ingeniería Inversa".

Con el uso de diferentes técnicas podemos descifrar todo el mundo que envuelve al software en su totalidad y esto nos lleva a diferentes campos de conocimientos en los cuales lo único que nos detiene es nuestra imaginación.

Cuando utilizar la Ingeniería Inversa?. Es necesario cuando se tiene un listado de código no estructurado y no documentado y se debe de obtener la documentación completa del programa.

1.1 REINGENIERÍA DEL SOFTWARE

La Reingeniería del Software es la etapa que surge de aplicar las técnicas de Inteligencia Artificial y matemática sofisticada al análisis automatizado y modificación del código fuente de programas, para abreviarlo y hacerlo mas eficiente.

Actualmente la creación y modificación de los programas de computadora es una tarea principalmente manual, difícil e imprevisible.
Por lo que los programas grandes suelen ser más complejos y difíciles e impredecibles de depurar.

Aunque la técnica todavía está en su infancia, la reingeniería del software está empezando a tomar auge en algunas tareas de programación, particularmente las tareas menos creativas y más repetitivas con el fin de automatizarlas.

Estos programas de reingeniería, escritos en lenguajes especialmente diseñados, operan en el código fuente de los programas y realizan una gran variedad de análisis y modificaciones.

1.2 PORQUE APLICAR REINGENIERÍA DE SOFTWARE

Cuando una aplicación ha servido para las necesidades del negocio de una compañía durante varios años, se vuelve inestable debido a las correcciones, adaptaciones y mejoras que se realizaron.

Esto deriva que cada vez que se intenta efectuar un camino se produzcan efectos colaterales graves e inesperados, como la perdida de datos, lentitud en los procesos, etc.
Por esta razón es conveniente utilizar la reingeniería del software, para evitarnos ciertos percances y pérdidas en nuestra información.

1.3 MODELO DE PROCESOS DE REINGENIERÍA DEL SOFTWARE

Puesto que la reingeniería es una suma de tareas que requieren por lo general mucho tiempo, ésta se divide en procesos separados que llevan acabo secuencial mente. Los procesos fundamentales de la reingeniería del software son los siguientes:

- **Análisis de Inventario**
  - ✓ Se realiza un inventario de todas las aplicaciones disponibles.
  - ✓ Se ordena ésta información de acuerdo a su antigüedad, importancia en el negocio, mantenibilidad actual y otros criterios.
- **Reestructuración de documentos.**
  - ✓ Pueden usarse 2 opciones, dependiendo de la más utilizable o conveniente para el negocio en cuestión.
  - ✓ Opción 1: Puede evitarse la documentación de programas estáticos con poca probabilidad de experimentar cambios.
  - ✓ Opción 2: Se documenta solamente lo que se modifica, y con el tiempo se obtendrá una valiosa colección de documentación de cambios realizados.
- **Reestructuración del código.**
  - ✓ Se analiza el código fuente utilizando una herramienta de reestructuración, como los decompiladores o debuggers. Las violaciones de las estructuras de programación estructurada se indican y entonces se reestructura el código.
- **Reestructuración de datos.**
  - ✓ Se deben tener en cuenta cuando, se suceden por reglas de negocio u otras causas de reestructuración de datos, ya que inevitablemente ésta producirá una reestructuración de código.

- **Ingeniería progresiva.**

  - ✓ Es un mundo ideal, las aplicaciones se reconstruyen utilizando "motor de reingeniería automatizado". Se insertaría el viejo programa y lo analizaría, reconstruiría y después regeneraría mejores aspectos de calidad de software.

1.4 REESTRUCTURACIÓN.

La reestructuración del software modifica el código fuente y los datos en un intento de hacerlos adecuados para futuros cambios. Tiende a centrarse en los detalles de diseño de módulos y en estructuras de datos locales definidas dentro de los módulos.

La reestructuración brinda los siguientes beneficios:

- ✓ Se obtienen programas de mayor calidad, mejor documentación y menos complejidad.
- ✓ Reduce la frustración entre ingenieros que deban trabajar con el programa, mejorando la productividad y facilitando el aprendizaje.
- ✓ Reduce el esfuerzo requerido para llevar acabo el mantenimiento.
- ✓ Hace que el software sea más sencillo de comprobar y de depurar.

1.5 TERMINOLOGÍA

A continuación se va a mostrar una serie de conceptos los cuales se manejarán a lo largo de este capitulo, ya que va hacer mas ameno para el lector tener por anticipado las terminologías, ya que en las lecturas posteriores podrá entender, no al 100%, pero no perderá la secuencia de la lectura.

- **Interfaz de programación de aplicaciones (Application Programming Interface "API")**

Conjunto de estándares o convenciones mediante los cuales los programas pueden llamar servicios de red o sistemas operativos específicos. Los programas para realizar ciertas funciones llaman librerías estándares (*.dll) que existen en prácticamente todas las computadoras y con esto un programador agiliza las funciones de sus programas.

- **Punto de interrupción o de Ruptura (Breackpoint)**

Ubicación en un programa donde este se detiene para dejar que el usuario decida qué hará enseguida o punto donde el programa se detiene para recurrir a la función de alguna librería.

Cuando nos referimos a estos, será que estamos buscando el punto en que el programa llama a cierta función API para llevar acabo otra función.
A esto también se le conoce como breakpoint on execution (Punto de ruptura durante la ejecución). Esto es sumamente usado en algunos programas desensambladores.

- **Desensamblador (Disassembler)**

Un desensamblador toma el código de un modulo (*.exe, *.dll, *.vxd etc.), y los analiza para producir una lista en ensamblador, sin importan en que lenguaje fue escrito el módulo.

Lo desensamblado puede ser estudiado y con un buen desensamblador su representación puede ser modificada, como renombrar variables, parámetros stack[1], etc.

Un solo desensamblador como tal no permite ejecutar el código paso a paso, o ver el interior de un programa en ejecución real. Por esta razón una lista desensamblada en ocasiones se le llama como código muerto.

- **Decompilador (Decompiler)**

Un decompilador toma el código de un módulo e intenta producir el equivalente en un lenguaje de alto nivel.

Hay que tomar en cuenta que para la mayoría de los lenguajes de alto nivel una exacta y acertada representación es imposible de producir.

Los decompiladores para lenguajes que compilan en código fuente propio, como C, C++, etc. generalmente no son usados en el ámbito de la ingeniería inversa.
La razón para esto es que no son demasiados exactos, en cambio una lista bien desensamblada provee una mejor solución para poder estudiar nuestro objetivo.

Los decompiladores son más bien usados para lenguajes que compilan en pseudo-código[2] (P-code). Como ejemplo se pueden citar aplicaciones de Visual Basic, programas de instalación y múltiples lenguajes de macros.

---

[1] Stack= Pila. El ultimo elemento que entra, es el primero en salir (Se definirá más adelante)
[2] Interpretación lógica de lo que se pretende diseñar, antes de generar un código

- **Depurador (Debugger)**

Un depurador es una utilería que nos ayuda a los programadores a encontrar y corregir errores de sintaxis y de otro tipo en el código fuente.

Un depurador te permite detener la ejecución de un programa e ir por el código instrucción por instrucción. Los depuradores vienen en muy distintas formas: la mayoría de los lenguajes de programación modernos vienen con un depurador integrado, el cual permite ver el código fuente en el mismo lenguaje en el que fue escrito.

Para hacer esto el depurador utiliza los archivos del código fuente, y cierta información de depuración contenida en el mismo ejecutable.

- **Candado (Dongle)**

Pequeña pieza de hardware que se conecta a un puerto.

Son utilizadas por algunos programas como candados de protección contra copiado; si no se conecta el candado simplemente el programa no funciona. Otros dispositivos de este tipo posibilitan la transferencia infrarroja de datos o conectividad de red.

- **Vaciar (Dump)**

Transferir el contenido de la memoria virtual a una impresora o a un dispositivo de almacenamiento en disco.

- **Editor Hexadecimal (Hex editor)**

Generalmente no permiten hacer otra cosa más que modificar la imagen en disco o modificar la estructura de los ejecutables portables.

Usualmente no se utiliza un editor hexadecimal antes de haber desensamblado o depurado nuestro objetivo y haber decidido en que lugar va ser modificado.

- **Empacado / Comprimido (Packed / Compressed)**

Los compresores de ejecutables portables empacan los datos del ejecutable dado y después envuelven en código los datos comprimidos.

Cuando el programa es ejecutado, el código desempacador o cargador como se llama comúnmente, desempaca los datos comprimidos dentro de la memoria RAM. Y es entonces cuando brinca al POE[3] del archivo exe en memoria y el programa entonces corre de forma normal.

[3] Punto Original de Entrada, Original Entry Point.

Los beneficios de empacar un exe son obviamente que el tamaño de éste es usualmente reducido en pequeña proporción. Pero comúnmente los empacadores son utilizados como métodos de anti-ingeniería inversa[4].

Los programas empacados no pueden ser desensamblados antes de que sean desempacados, sin embargo, existen herramientas que permiten tomar el código fuente de la memoria RAM.

- **Encriptación (Encryption)**

Encriptar es un proceso de convertir un mensaje de texto cifrado o encriptado utilizando una clave, de manera que parezca que el mensaje sólo contiene basura. Sin embargo, el destinatario designado puede aplicar la clave para desencriptar y leerlo.

- **Indicador (Flag)**

Variable que sirve para mostrar el estado de un programa, archivo o dato. Un indicador es un registro de base de datos, por ejemplo puede ser verdadero si otros campos del registro muestran que ya se venció la fecha de alquiler de una cinta de video o sinónimo de atributo de archivo.

- **Cargador (loader)**

Un cargador es el término que se le da a cualquier entidad que sea responsable de preparar e iniciar un proceso.

Cuando se inicia cualquier aplicación en Windows, el cargador de Windows lee la imagen en disco de la aplicación, analiza gramaticalmente la estructura de la misma, la prepara en memoria y después pasa el mando al proceso de punto de entrada.

En términos de ingeniería inversa un cargador es comúnmente descrito como un programa que realiza un parche en memoria de una aplicación, para poder remover alguna protección.

- **Modo Protegido (Protected Mode)**

En el modo protegido los registros de la memoria limitada en modo real contiene apuntadores a registros adicionales con lo que se supera la limitación de memoria de 640 kb.

Además los apuntadores contienen información de protección de memoria, lo cual permite al procesador protegerla.

[4] Concepto referente a evitar la extracción de código, por medio de un programa ejecutable

Con soporte del sistema operativo este permite hacer multitarea por referencia en lo cual los programas puedes coexistir con seguridad en el mismo espacio de memoria.

Como opción predeterminada los procesadores Intel inician en modo real y requieren que el software los cambie al modo protegido.

- **Modo Real (Real Mode)**

El modo Operativo de los microprocesadores Intel en que las ubicaciones de memoria se asignan directamente de acuerdo a un conjunto limitado de registros, produciendo un tamaño total máximo de memoria de 1Mb y en la practica 640 Kb debido a la asignación de parte de la memoria al uso de los dispositivos periféricos.

- **Módulo (Module)**

Es un programa, unidad o sección capaz de funcionar por si misma. Por ejemplo un programa integrado, el usuario puede usar el módulo de procesamiento de texto como si fuera un programa separado e independiente.

- **Secuencia de Comandos (Script)**

Serie de instrucciones parecidas a una macro y escritas en texto simple, que le indican a un programa como ejecutar un procesamiento especifico; por ejemplo como conectarse a un sistema de correo electrónico.

Algunos programas integran capacidades de elaboración scripts. El usuario debe aprender a usar scripts mediante algún lenguaje de programación relativamente sencillo.

Algunos programas escriben el script de forma automática grabando las teclas que el usuario oprime y los comandos que selecciona conforme llevan a cabo el procedimiento.

- **Parchador (Patcher)**

Parchar es el proceso de modificar la imagen o archivo del objetivo en particular. Generalmente es la forma más rápida para remover protecciones simples y mundanas y generalmente no se necesita un conocimiento experto más que entender el lenguaje ensamblador.

- **Pila (Stack)**

Se utiliza en programación estructurada de datos en la cual los primeros componentes insertados son los últimos que se retiran.

La estructura de datos "Ultimo entrar – Primero en salir" (LIFO) se emplea en programas que usan estructuras de control, un pila permite que una computadora encuentre lo que estaba haciendo cuando se bifurco o salto a otro procedimiento.

- **Algoritmo (Algorithm)**

Serie de instrucciones o pasos de que consta un procedimiento ideado para resolver un problema especifico.

- **Código autoverificante (Self Checking Code)**

Código en el cual la representación de cada carácter se construye obedeciendo a reglas especificas. Ciertas combinaciones de los elementos con los que se construye el juego de caracteres no obedecen a estas reglas.

Tales combinaciones se denominan caracteres prohibidos y pueden ser reconocidas o rechazadas como erróneas si aparecen en un mensaje.

Usado para comprobar la integridad de un programa ya compilado, comprobando si el código de este ha sido alterado por algún agente externo como un virus.

- **Código Maquina (Machine Language Code)**

Sistema de codificación adoptado en el diseño de un ordenador para representar su repertorio de instrucciones.

Las diversas operaciones realizables se representan mediante códigos numéricos de función, a todas las posiciones de memoria se les asignan números para permitir el direccionamiento de los datos almacenados en tales posiciones, también denominado código de ordenador.

- **Compresión (Pack)**

Colocación de más de un elemento de información en una misma unidad de memoria, con el objeto de ahorrar espacio de memoria.

- **Descomprimir (Unpack)**

Recuperar un dato original de una posición de memoria en la cual ha sido comprimido juntamente con otros datos.

- **Desplazamiento (Offset or Displacement)**

Referido a un valor a añadir a una dirección base para producir una segunda dirección. Por ejemplo si B representa 100, entonces la expresión B+5, significara 105. El 5 de la expresión es el desplazamiento (offset). Especificar direcciones

usando un desplazamiento (offset) se denomina direccionamiento relativo porque la dirección resultante es relativa a algún otro punto.

- **Desvío, capacidad de (Trapping)**

Una característica de los programas monitores que efectúan verificaciones automáticas del funcionamiento de otros programas.

La trampa está diseñada con el fin de detectar incidentes insólitos o inesperados durante el funcionamiento de un programa, e iniciará una bifurcación incondicional a un cierto programa de diagnóstico o rutina de recuperación de errores.

- **Dirección (Address)**

La parte de una instrucción que especifica en qué posición de la memoria se encuentra almacenado u operando.

- **Posición de Memoria (Memory Location)**

Cualquier lugar de memoria de un ordenador capaz de alojar una unidad de información.

Se expresa habitualmente en función de la unidad básica de almacenamiento utilizada en un determinado sistema informático, por ejemplo en las unidades de memoria organizadas en palabras, cada una de estas es una unidad de almacenamiento.

La ubicación de cada posición de memoria está identificada por subdirecciones.

- **Volcado de Memoria (Memory Dumping)**

Técnica utilizada durante el funcionamiento de un programa con el fin de asegurar que en el caso de avería de la maquina o de alguna interrupción de la tarea, será posible reanudar la ejecución del programa sin necesidad de empezar de nuevo.

La Técnica consiste en escribir periódicamente el programa y sus datos, más el contenido de las zonas de trabajo.

- **Compilador (Compiler)**

Un complejo programa que convierte en código máquina las instrucciones de ordenador escritas en lenguaje fuente. Tras la compilación el ordenador es capaz de leer el programa objeto resultante y actuar basándose en él.

Para producir el programa objeto, el compilador traduce cada una de las sentencias del lenguaje fuente en sus equivalentes en código máquina incorpora

al programa objeto cualesquiera subrutinas de biblioteca solicitadas por el usuario, establece los enlaces de interconexión entre las partes del programa.

Los compiladores se diferencian de los programas ensambladores en que, de ordinario, los compiladores generan más de una instrucción en código máquina por cada sentencia fuente, mientras que las instrucciones dadas en un lenguaje ensamblador se corresponden con las del código máquina.

- **Crack**

A continuación se mostraran algunas definiciones:

- ✓ Romper la seguridad de un sistema. El término fue acuñado a mediados de los 80 por hackers que querían diferenciarse de los que solo tenían como propósito burlar sistemas de seguridad.

Mientras que el propósito de los crackers es burlar los sistemas de seguridad, los hackers se interesan más en ganar conocimiento sobre sistemas informáticos y usar este conocimiento para actos más honorables.

Aunque los hackers están de acuerdo en que hay una gran diferencia entre lo que ellos hacen y lo que hacen los crackers, la mayoría de la gente no ha comprendido o entendido las diferencias, por eso los dos términos hack y crack son muy a menudo intercambiados.

- ✓ Copiar software comercial ilegalmente burlando las protecciones anti-copia y las técnicas de protección o registro que sean usadas por el software.

- **Craker**

- ✓ Persona o cosa que tiene una calidad excepcional.

- ✓ El que rompe la seguridad de un sistema.

La utilización de ambos neologismos[5] refleja una fuerte repulsión contra el robo y vandalismo perpretado por los círculos de crackers. Aunque se supone que cualquier hacker auténtico ha jugado con algún tipo de crackeo y conoce muchas de las técnicas básicas, se supone que cualquiera que haya pasado la etapa inicial ha desterrado el deseo de hacerlo con excepción de razones prácticas inmediatas, por ejemplo, si es necesario pasar por alto algún sistema de seguridad para completar algún tipo de trabajo. Por lo tanto, hay mucho menos en común entre el mundo de los hackers.

[5] Vocablos o lenguajes.

Los crackers tienden a agruparse en grupos pequeños, muy secretos y privados, aunque los crackers a menudo se definen a sí mismos como hackers, la mayor parte de los auténticos hackers los consideran una forma de vida inferior.
Consideraciones éticas aparte, los hackers consideran que cualquiera que no sea capaz de imaginar una forma más interesante de jugar con su ordenador que romper los sistemas de alguien ha de ser bastante perdedor.

- ✓ Se denomina cracker al avezado pirata que consigue desproteger totalmente algún programa.

Los crackers son muy orgullosos y normalmente ponen su marca de fábrica, en algunos ordenadores, e incluso la acompañan de efectos de animación en pantalla y una esplendorosa música. Las apropiaciones que llevan a cabo estos piratas están teñidas de caracteres profundamente artísticos. Su actividad más destacada consiste en la desprotección del software ajeno y la posterior apropiación del mismo.

No es necesario insistir en la dificultad que encierra la desprotección de programas; sin embargo, resulta difícil hacerse una idea exacta de hasta qué punto se requieren habilidades especiales en la elaboración de las marcas necesarias para saltarse protecciones ajenas.

El cracker es ese mismo individuo en el terreno de la informática: un artista, un magnífico ladrón especializado.

La habilidad de este tipo de piratería no es diplomática, no estratégico-política.

El cracker es un cerebro de la ingeniería informática, una mente laberíntica y fértil.

- ➢ **Dirección Absoluta (Absolute Address)**

La dirección real o verdadera de una posición de memoria, expresada mediante el sistema de numeración del código de máquina.

Se la conoce también por: dirección verdadera, dirección directa, dirección de máquina, dirección real, dirección específica.

- ➢ **Dirección Base (Base Address)**

1.- En modificación de direcciones, dícese de aquella dirección a la que se agrega un modificador para obtener la dirección variable de un operando.

2.- Durante el ensamblado o la carga de un programa, dirección sumada a la componente de dirección de cada instrucción al objeto de obtener direcciones absolutas.

- **Dirección Real (Actual Address Or Real Address)**

Una dirección de la memoria principal; se contrapone a dirección virtual, cuya dirección real se obtiene consultando una tabla o sumándole un dato.

Entre los sinónimos de dirección real se cuentan dirección directa, dirección absoluta, dirección específica y dirección verdadera.

- **Dirección Virtual (Virtual Address)**

Dirección que alude a una posición de memoria, pero que es preciso traducir mediante una conversión de direcciones con el fin de obtener una dirección real correspondiente a una posición de memoria principal.

Su significado es análogo al de dirección relativa; pero se aplica a ordenadores que dispongan de medios de memoria virtual.

- **Direccionamiento Indirecto (Indirect Addressing)**

Técnica de programación en la cual la zona de dirección de una instrucción hace referencia a otra posición, la cual contiene otra dirección.

Esta última dirección puede especificar un operando, pero también puede especificar una dirección anterior. También denominado direccionamiento de niveles múltiples.

- **Direccionamiento Relativo (Relative Addressing)**

Un sistema de programación en el cual las direcciones han sido escritas de modo que no aluden sin más a direcciones absolutas de la memoria; en lugar de eso, en el momento de cargar el programa se le suma a la componente de dirección base, con el fin de crear números que sí designen posiciones absolutas.

Así una subrutina compuesta por 20 instrucciones podría estar redactada con una secuencia de 20 palabras que comenzasen en la dirección base R, y continuan en la R+1, R+2,..., hasta R+19. Si se tomase R igual a 1200, en la codificación absoluta la subrutina ocuparía en la memoria las palabras 1200 a 1219.

Este método le permite al programador escribir un programa en varias secciones o segmentos independientes, sin tener que prestar atención a las direcciones absolutas necesarias.

- **Ensamblado (Assembly)**

Operación que se efectúa sobre un programa redactado en lenguaje simbólico, cuyo resultado es un programa completo en lenguaje máquina.

En esencia, el ensamblado consta de traducción de los códigos de operación y direcciones dados simbólicamente a su correspondiente expresión en lenguaje máquina, montaje del programa en lenguaje máquina resultante a partir de los fragmentos que lo componen, por ejemplo, la inclusión de rutinas de biblioteca, la consolidación de segmentos del programa, ajustes en los enlaces, etc.

El ensamblado se diferencia de la compilación en que la correspondencia entre las instrucciones en lenguaje máquina producidas a partir de las instrucciones relativas es una correspondencia, mientras que la compilación produce normalmente muchas instrucciones de máquina a partir de una pseudo-instrucción.

- **Ensamblador (Assembler)**

Programa que opera sobre otro programa redactado en lenguaje simbólico con el fin de generar un programa en lenguaje máquina en el proceso de ensamblado. Conocido también por programa ensamblador, rutina de ensamblado.

- **Juego de Instrucciones (Instruction Set)**

Repertorio de órdenes o mandatos de que dispone un el lenguaje de un ordenador particular o que componen un sistema de programación. Conocido también por el código de instrucciones, código de máquina y código de ordenes.

- **Lenguaje Máquina (Machine Language)**

En su sentido más estricto se refiere a las instrucciones escritas en código máquina, que un ordenador puede obedecer inmediatamente, sin traducción alguna. Se utiliza más para aludir a cualquier de las instrucciones simbólicas destinadas a ser ejecutadas por un sistema informático.

- **Mochila (Dongle)**

Sistema de protección que se basa en la colocación de una pastilla con chip y memoria en una de las salidas del ordenador, frecuentemente en los puertos serie o paralelo.

- **Pascal (Pascal)**

Un lenguaje de alto nivel desarrollado a finales del decenio de 1960, inspirado en el ALGOL[6]. El Pascal es un lenguaje estructurado, provisto de características algorítmicas diseñadas para lograr una rápida ejecución del programa resultante.

[6] Lenguaje de programación antecesor de Pascal

- **Parche (Patch)**

Un remedio temporal para corregir un error (bug) de un programa. Un parche es una porción de código objeto que es insertado en el programa ejecutable. Normalmente, el parche queda ubicado como codificación fuera de línea y se penetra en él mediante una instrucción de bifurcación incondicional situada en la parte de la rutina que está siendo alterada.

- **Phreaking**

Cercano a hackear, usando una computadora u otro dispositivo engañar a una sistema telefónico. Típicamente, phreaking se usa para realizar llamadas gratis o para cargarlas a otra cuenta.

- **Pirata**

Usuario de informática que utiliza el software ajeno sin ningún respeto a la moral establecida o a la legislación vigente. Esto supone un 99,9% de los usuarios.

- **Posición de Memoria (Memory Location)**

Cualquier lugar de la memoria de un ordenador capaz de alojar una unidad de información.

Se expresa habitualmente en función de la unidad básica de almacenamiento utilizada en un determinado sistema informático; por ejemplo, en las unidades de memoria organizadas en palabras, cada una de éstas es una unidad de almacenamiento. La ubicación de cada posición de memoria está identificada por su dirección.

- **Programa (Program)**

Conjunto de instrucciones preparadas al objeto de resolver un determinado problema mediante un ordenador.

- **Programa Ejecutivo (Executive Program)**

Un programa ejecutivo o sistema ejecutivo, consta por lo general de cierto número de rutinas complejas residentes en todo o en parte en la memoria principal de un ordenador; su función es controlar y supervisar cierto número de funciones básicas de control.

En los ordenadores de tiempo compartido suele considerarse que el programa ejecutivo forma parte del hardware, pues en general, resulta imposible hacerlos funcionar sin tal programa.

Mediante un programa ejecutivo se podrían controlar, por ejemplo, las siguientes funciones: gestión e interpretación de todos los mensajes de control y todas las señales recibidas en la consola del ordenador o transmitidas desde ella; supervisión y control del sistema de compartición de tiempos, para asegurarse de que las unidades periféricas se pueden hacer funcionar simultáneamente en modo asincrónico, permitiendo así que todos los elementos del equipo sean utilizados con la máxima eficiencia.

Controlar el funcionamiento simultáneo de varios programas en un ambiente multiprogramación, conmutando automáticamente el control entre un programa y otro de acuerdo con prioridades especificadas, permitir que se preste servicio a unidades periféricas mediante interrupciones automáticas y otros acontecimientos.

- **Programa Monitor (Monitor Program)**

1.- Cualquier rutina encargada de supervisar el desarrollo del trabajo en un sistema informático.

2.- Más específicamente, rutina o programa destinado a supervisar y controlar el funcionamiento de los programas en el ordenador, asumiendo funciones propias de un programa ejecutivo.

3.- Rutina utilizada para proporcionar información sobre el desarrollo de un programa, con el fin de diagnóstico y depuración.

- **Protección (Protection)**

1.- Técnicas destinadas a impedir la interferencia entre unidades de software o la superposición de zonas de datos en un sistema de multiprocesado.

2.- Técnica para impedir o dificultar el uso no autorizado de software o datos; mediante contraseñas o por refinados medios de hardware.

- **Punto de Entrada (Entry Point)**

La primera instrucción que es preciso obedecer en una rutina o programa. Una rutina o programa puede tener varios puntos en función de las diferentes condiciones de entrada u operaciones a realizar.

- **Registro (Register / Record)**

1.- Unidad de datos que representa una transacción particular o un elemento básico de un archivo.

2.- Una posición de memoria especial. Por lo común, de capacidad equivalente al tamaño de palabra del ordenador de que se trate y que tiene propiedades especiales y útiles para efectuar operaciones aritméticas o lógicas.

Por ejemplo, en algunos ordenadores la única forma de poder realizar operaciones aritméticas es que al menos uno de los operandos se encuentre almacenado en un registro especial.

Un ordenador puede tener varios registros, cada uno de los cuales ha sido concebido para realizar funciones específicas.

- **Rutina (Routine)**

Se utiliza como palabra sinónima de programa, aunque muchas veces solamente expresa una parte de un programa.

Se puede decir, por ejemplo, que un programa consta de una rutina de entrada, una rutina principal, de rutinas de error y de una rutina de salida.
De hecho, el término rutina es aplicable a cualquiera de los principales procedimientos que desempeñen un papel bien definido en el funcionamiento de un programa o sistema.

- **Suma de Verificación (Check Sum)**

Suma generada a partir de los dígitos individuales de un número, utilizada formando parte de comprobación por suma. A veces se emplea como sinónimo de suma de comprobación aleatoria.

- **Instrucciones de movimientos de datos**

A continuación muestro una tabla con las diferente instrucciones que puede haber en el lenguaje ensamblador. Con el significado de cada instrucción.

MOV	Destino, fuente	La única instrucción que utiliza todos los tipos de direccionamiento
XCHG	Destino, fuente	Intercambia los contenidos de destino y fuente
XLAT	Tabla, fuente	Carga el registro AL con el byte direccionado por (BX+AL)
LAHF		Carga las flags S, Z, A, P y C en AH
SAHF		Guarda AH en el registro de flags
LDS	Destino, fuente	Transfiere un puntero de 32 bits al registro DS y al registro destino
LES	Destino, fuente	Transfiere un puntero de 32 bits al registro ES y al registro destino
LEA	Destino, fuente	Transfiere el offset de fuente (una dirección) a destino (un registro)
PUSH	Fuente	Guarda fuente en el stack (en la dirección SS:SP)
POP	Destino	Recupera del stack (dirección SS:SP-1) y guarda en registro destino
PUSHF		Almacena el registro de flags desde el stack
POPF		Recupera el registro de flags desde el stack
PUSHA		Almacena los reg DI,SI,BP,SP,BX,DX,CX,AX desde el stack

POPA		Recupera los reg DI,SI,BP,SP,BX,DX,CX,AX desde el stack
IN	Origen	Carga desde un puerto origen un byte o word en AL o AX
OUT	Destino	Escribe Al o AX en el puerto destino (direcciona, inmediato o DX)

➢ Las operaciones aritméticas

ADD	Destino, fuente	Suma fuente + destino y guarda el resultado en destino
ADC	Destino, fuente	Suma fuente + destino + Carry y guarda el resultado en destino
SUB	destino, fuente	Resta destino - fuente y guarda el resultado en destino
SUB	Destino, fuente	Resta destino - fuente - Carry y guarda el resultado en destino
MUL	Fuente	Multiplica AL o AX fuente y guarda el resultado en DX:AX
IMUL	Fuente	Igual que la anterior pero con números enteros con signo
DIV	Fuente	Divide DX:AX fuente y guarda cociente en AX y resto en DX
IDIV	Fuente	Igual que la anterior pero con números enteros con signo
AND	Destino, fuente	Opera destino AND fuente y guarda resultado en destino
OR	Destino, fuente	Opera *destino* OR fuente y guarda el resultado en destino
XOR	Destino, fuente	Opera *destino* XOR fuente y guarda el resultado en destino
NOT	Destino	El NOT cambia todos los 1 en 0 y los 0 en 1 de destino.
NEG	Destino	NEG realiza el complemento a 2 de destino
INC	Destino	Incremente an 1 el contenido de destino
DEC	Destino	Decrementa en 1 el contenido de destino
DAA / DAS		Efectúa el ajuste decimal en suma o resta del registro AL
AAA/AAD/ AAM/AAS		Ajustan el registro AL a valor decimal desempaquetado, para aplicar en operaciones suma, resta, multiplicación y división.

➢ Instrucciones de rotación

RCL	Destino, contador	Rota destino a través de carry a la izquierda contador veces
RCR	Destino, contador	Rota destino a través de carry a la derecha contador veces
ROL	Destino, contador	Rota destino a la izquierda contador veces
ROR	Destino, contador	Rota destino a la derecha contador veces
SAL	Destino, contador	Desplaza destino a la izquierda contador veces y rellena con ceros
SAR	Destino, contador	Desplaza destino a la derecha contador veces y rellena con bit SF
SHR	Destino, contador	Desplaza destino a la derecha contador veces y rellena con ceros

➢ Instrucciones de comparación

CMP	Destino, fuente	Compara fuente y destino. Modifica las flags V, Z, S, C, P y AC
TEST	Destino, fuente	AND entre fuente y destino. Ninguno de los operandos cambia.

TEST modifica las mismas flags que CMP pero siempre deja a V = 0 y C = 0.

➢ Instrucciones de strings

CMPS	string_destino, string_fuente	Compara las dos cadenas de a bytes o words
CMPSB	string_destino, string_fuente	Origen y destino indicados por DS:SI y ES:DI (bytes)
CMPSW	string_destino, string_fuente	Origen y destino indicados por DS:SI y ES:DI (words)
LODS	string_fuente	Mueve un byte o una word desde fuente a AL o AX
LODSB	string_fuente	Origen indicado por DS:SI (mueve un byte a AL)
LODSW	string_fuente	Origen indicado por DS:SI (mueve una word a AX)
STOS	string_destino	Mueve un byte o una word al destino desde AL o AX
STOSB	string_destino	Destino indicado por ES:DI (mueve AL a un byte)
STOSW	string_destino	Destino indicado por ES:DI (mueve AX a una word)
MOVS	string_destino, string_fuente	Mueve un byte o word de fuente a destino
MOVSB	string_destino, string_fuente	Origen y destino indicados por DS:SI y ES:DI (un byte)
MOVSW	string_destino, string_fuente	Origen y destino indicados por DS:SI y ES:DI (una word)
SCAS	string_destino	Compara la cadena de destino con AL o AX
SCASB	string_destino	Destino indicado por ES:DI (compara AL con un byte)
SCASW	string_destino	Destino indicado por ES:DI (compara AX con una word)

➢ Instrucciones de repetición

LOOP	offset	Decrementa CX. Si CX no es cero, salta a offset (IP = IP + offset)
LOOPZ	offset	Decrementa CX, Si CX <> 0 y Z = 1 , salta a offset (IP = IP + offset)
LOOPNZ	offset	Decrementa CX, Si CX <> 0 y Z = 0 , salta a offset (IP = IP + offset)

En todos los casos, si no se produce el salto, se ejecuta la próxima instrucción

REP	instrucción	Decrementa CX y repite la siguiente instrucción MOVS o STOS hasta que CX=0
REPZ	instrucción	Igual que REP, pero para CMPS y SCAS. Repite si la flag Z queda en 1 (igualdad)
REPNZ	instrucción	Igual que REPZ, pero repite si la flag Z queda en 0 (las cadenas son distintas)

➢ Instrucciones de salto

CALL	destino	Llama a procedimiento. IP <-- offset de destino y CS <-- segmento de destino
RET	valor	Retorna desde un procedimiento (el inverso de CALL), valor es opcional
INT	número	Llamado a interrupción. CS:IP <-- vector de INT.Las flags se guardan en el stack
INTO		Llama a la INT 4 si la flag de overflow (V) está en 1 cuando se ejecuta la instrucción
IRET		Retorna de interrupción al programa restaurando flags
JMP	dirección	Salta incondicionalmente al lugar indicado por dirección
JA	offset	Salta a IP + offset si las flags C=0 Y Z=0 (salta si primer operando es mayor)
JAE	offset	Salta a IP + offset si la flag C=0 (salta si primer operando es mayor o igual)
JB	offset	Salta a IP + offset si las flags C=1 (salta si primer operando es menor)(igual a JC)
JBE	offset	Salta a IP + offset si las flags C=1 o Z=1 (salta si primer operando es menor o igual)

JZ	offset	Salta a IP + offset si las flags Z=1 (salta si primer operando es igual al segundo) (=JE)
JG	offset	Salta a IP + offset si las flags S=V Y Z=0 (salta si primer operando es mayor)
JGE	offset	Salta a IP + offset si las flags S=V (salta si primer operando es mayor o igual)
JL	offset	Salta a IP + offset si las flags S<>V (salta si primer operando es menor)
JLE	offset	Salta a IP + offset si las flags S<>V o Z=1(salta si primer operando es menor o igual)
JNC	offset	Salta a IP + offset si la flag C=0 (salta si no hay carry)
JNZ	offset	Salta a IP + offset si la flag Z=0 (salta si no son iguales o no es cero)
JNO	offset	Salta a IP + offset si la flag V=0 (salta si no hay overflow)
JNP	offset	Salta a IP + offset si la flag P=0 (salta si no hay paridad -o la paridad es impar =JPO)
JNS	offset	Salta a IP + offset si la flag S=0 (salta si no hay bit de signo)
JO	offset	Salta a IP + offset si la flag V=1 (salta si hay desbordamiento -overflow)
JP	offset	Salta a IP + offset si la flag P=1 (salta si la paridad es par) (=JPE)
JS	offset	Salta a IP + offset si la flag S=1 (salta si el signo es negativo)
JCXZ	offset	Salta a IP + offset si la flag CX=0 (salta si el registro CX es cero)

Las instrucciones de saltos por Above o Below se refieren entre dos valores sin signo (JA, JAE, JB y JBE), mientras que las Greater y Less se refieren a la relación entre dos valores con signo (JG, JGE, JL y JLE).

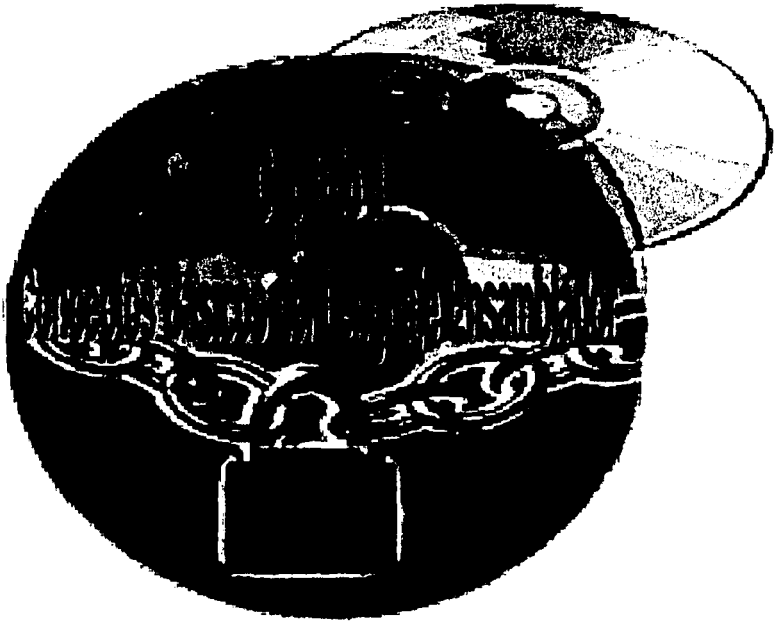
- **Instrucciones que afectan flags**

CLC/CMC/STC	pone a cero, complementa y pone en 1 la flag C (carry)
CLD/STD	pone a cero, uno la flag de dirección (D=0 hace que SI y DI se incrementen)
CLI/STI	deshabilita y habilita las interrupciones por hardware enmascarables

- **Instrucciones misceláneas**

NOP	no-operación: el procesador pasa a la instrucción siguiente sin hacer nada
CBW	Convierte el byte de AL en palabra (AX), copiando el bit 7 a todo el registro AH
CWD	Convierte word en double-word, copiando bit 15 de AX a todo el registro DX
HLT	el procesador se detiene hasta que llegue un Reset o una interrupción por hard.

Todas estas instrucciones y conceptos forman parte esencial sobre el manejo o desarrollo de aplicaciones en ensamblador, las cuales nos van a ser muy útiles para comprender y darle sentido a los contenidos de los siguientes capítulos.



Los programas de Win32 corren en modo protegido, disponible desde el 80286. Pero ahora el 80286 es historia. Así que ahora debemos interesarnos en el 80386 y sus descendientes. Windows corre cada programa de 32 bits en espacios virtuales de memoria separados.

Eso significa que cada programa de Win32 tiene sus propios 4 GB[7] de memoria en el espacio de direcciones. Como sea, esto no significa que cada programa de Win32 tenga 4 GB de memora física, si no que el programa puede direccionar cualquier dirección en ese rango.

Windows hará cualquier cosa que sea necesaria para hacer que la memoria y las referencias del programa sean válidas. Por supuesto, el programa debe adherirse a las reglas impuestas por Windows, si no, causará un error de protección general. Cada programa está solo en su espacio de direcciones.

Esto contrasta con la situación en Win16. Todos los programas de Win16 podían verse unos a otros. No es lo mismo en Win32. Esto reduce la posibilidad de que un programa escriba sobre el código datos de otros programas.

El modelo de la memoria es también drásticamente diferente al de los antiguos días del mundo de 16-bits. Bajo Win32 ya no necesitamos meternos nunca más con el modelo o los segmentos de memoria. Hay un solo tipo de modelo de memoria: El modelo plano flat.

Ahora, no hay solamente segmentos de 64k. La memoria es un largo y continuo espacio de 4 GB. Eso también significa que no tendrás que jugar más con los registros de los segmentos. Ahora puedes usar cualquier registro de segmento para direccionar a cualquier punto en el espacio de la memoria. Eso es una genial ayuda para los programadores. Esto hace la programación de ensamblador para Win32 tan fácil como C.

Cuando programas bajo Win32, debes tener en cuenta unas cuantas reglas importantes.

- ✓ Windows usa (esi, edi, ebp y ebx) internamente y no espera que los valores en esos registros cambien. Así que recuerda esta regla primero: si usas cualquiera de estos cuatro registros.

- ✓ En tu función callback nunca olvides restaurarlos antes de regresar el control a Windows. Una función callback[8] es una función escrita por ti que Windows llama cuando algún evento específico se produce.

[7] Capacidad de almacenamiento en información o procesos (Giga Byte).
[8] Llamada a una referencia.

El ejemplo más obvio es el procedimiento de ventana. Esto no significa que no puedas usar estos cuatro registros; sí puedes. Solo asegúrate de restaurarlos antes de pasarle el control a Windows.

Ejemplo:

Aquí hay un esqueleto de un programa. Si no entiendes algo de los códigos, estos se explicaran más adelante.

```
.386
.MODEL Flat, STDCALL
.DATA
  <Tu data (información) inicializada>
  ......
.DATA?
  <Tu data NO inicializada>
  ......
.CONST
  <Tus constantes>
  ......
.CODE
  <Etiqueta>
   <Tu código>
  .....
   end <Etiqueta>
```

Vamos a analizar el programa esqueleto.

.386

Esto es una directiva para el ensamblador, que le dice que vamos a usar el conjunto de instrucciones del 80386. También puedes usar .486, .586 pero es mas seguro utilizar .386. Hay actualmente dos formas casi idénticas para cada modelo de CPU. .386/.386p, .486/.486p. Esas versiones de "p" son necesarias cuando tu programa usa instrucciones privilegiadas. Las instrucciones privilegiadas son las instrucciones reservadas por el CPU del sistema operativo cuando están en modo protegido. Solamente pueden ser usadas por un código privilegiado, así como los VXD[9].

```
.MODEL FLAT, STDCALL
```

.MODEL es una directiva para el ensamblador que especifica el modelo de memoria de tu programa.

Bajo Win32, hay un solo tipo de memoria, la PLANA(FLAT). STDCALL Indica al ensamblador la conversión de paso de los parámetros. La conversión de paso de parámetros especifica el orden que debe seguirse para pasar parámetros, izquierda a derecha o derecha a izquierda, y también equilibrará la pila después de una llamada.

Bajo Win16, hay dos tipos de conversiones para las llamadas a funciones: C y PASCAL. La conversión para pasar parámetros de derecha a izquierda en cada

[9] Virtual Device Drivers, controladores de dispositivos virtuales.

llamada, es decir, el parámetro de más a la derecha es empujada primero a la pila. La rutina que hace la llamada es la responsable de equilibrar el marco de la pila después de la llamada. Por ejemplo, si vamos a llamar a una función con nombre foo(int primera_parte, int segunda_parte, int tercera_parte) en lenguaje C la conversión sería así en ASM[10]:

```
push [tercera_parte]      ; Empuja el tercer parámetro
push [segunda_parte]      ; Seguido por el segundo
push [primera_parte]      ; Y aquí se pone de primer call foo
add  sp, 12               ; La llamada equilibra el marco de la pila
```

La conversión de llamadas en PASCAL es totalmente al revés de la conversión de C. Pasa los parámetros de izquierda a derecha y la rutina que llama es responsable de equilibrarse después de la llamada.

El sistema Win16 adopta la conversión de PASCAL porque produce códigos más pequeños. La conversión de C es eficiente cuando no sabes cuántos parámetros serán pasados a la función como es el caso de wsprintf(). En el caso de wsprintf(), no hay manera de determinar cuántos parámetros serán empujados por esta función a la pila, así que no se puede hacer el balanceo de la pila.

STDCALL es la conversión híbrida entre C y PASCAL. Pasa parámetros de derecha a izquierda, pero la llamada es la responsable por el balance de la pila después de la llamada. La plataforma de Win32 usa el modelo STDCALL exclusivamente. Excepto en un caso: wsprintf(). Debes usar la conversión de llamada C con la función wsprintf().

```
.DATA
.DATA?
.CONST
.CODE
```

Estas 4 directivas son las llamadas secciones. No tienes segmentos en Win32, pero puedes dividir todo el espacio de direcciones en secciones lógicas. El comienzo de una sección demuestra el fin de la otra sección previa. Hay dos grupos de secciones: data y code. Las secciones Data están divididas en tres categorías:

.DATA Esta sección contiene la información inicializada de tu programa.
.DATA? Esta sección contiene la información no inicializada de tu programa. A veces quieres solamente prelocalizar alguna memoria pero no quieres iniciarla. Ese es el propósito de esta sección. La ventaja de la información no inicializada es que no toma espacio en el ejecutable. Por ejemplo, si tu localizas 10,000 bytes en tu sección .DATA?, tu ejecutable no se infla 10,000 bytes. Su tamaño se mantiene muy homogéneo. Tu sólo le dices al ensamblador cuánto espacio necesita cuando el programa se carga en la memoria, eso es todo.
.CONST Esta sección contiene declaraciones de constantes usadas por el programa. Las constantes nunca pueden ser modificadas. Sólo son constantes.

[10] Abreviación sobre el lenguaje ensamblador

No tienes que usar las tres secciones en tu programa. Declara solo las secciones que quieres usar.
Existe solo una sección para el código CODE. Aquí es donde tu código reside.

```
<etiqueta>
end <etiqueta>
```

Donde está la <etiqueta> se puede usar cualquier nombre como etiqueta para especificar la extensión de tu código. Ambas etiquetas deben ser idénticas. Todos tus códigos deben residir entre <etiqueta> y end <etiqueta>.

2.1 MESSAGEBOX

Windows tiene preparado una gran cantidad de recursos para sus programas. En el centro de esta concepción se ubica la API de Windows.
La API de Windows es una enorme colección de funciones muy útiles que residen en el propio sistema Windows, listas para ser usadas por cualquier programa de Windows. Estas funciones están almacenadas en varias librerías de enlace dinámico, DLL tales como kernel32.dll, user32.dll y gdi32.dll.
Kernel32.dll contiene las funciones de la API que tienen que ver con el manejo de memoria y de los procesos. User32.dll controla los aspectos de la interface de usuario de tu programa. Gdi32.dll es la responsable de las operaciones gráficas. Además de estas tres funciones principales, hay otras DLLs que nuestros programas pueden emplear, siempre y cuando tengas la suficiente información sobre las funciones de la API que te interesan.
Los programas de Windows se enlazan dinámicamente a estas DLLs, es decir, las rutinas de las funciones de la API no están incluidas en el archivo ejecutable del programa de Windows. Con el fin de que tu programa pueda encontrar en tiempo de ejecución las funciones de la API deseadas, esa información tiene que estar incluida en el archivo ejecutable.
Esta información se encuentra dentro de archivos .LIB. Debes enlazar tus programas con las librerías de importación correctas o no serán capaces de localizar las funciones de la API.
Cuando un programa de Windows es cargado en la memoria, Windows lee la información almacenada en el programa. Esa información incluye el nombre de las funciones que el programa usa y las DLLs donde residen esas funciones. Cuando Windows encuentra esa información en el programa, cargará las DLLs y ejecutará direcciones de funciones fijadas en el programa de manera que las llamadas transfieran el control a la función correcta.
Hay dos categorías de funciones de la API: Una para ANSI[11] y la otra para Unicode. Los nombres de las funciones de la API para ANSI terminan con "A", por ejemplo, MessageBoxA. Los de Unicode terminan con "W". Windows 95 soporta ANSI y Windows NT Unicode[12].
Generalmente estamos familiarizados con las cadenas ANSI, que son arreglos de caracteres terminados en NULL. Un carácter ANSI tiene un tamaño de 1 byte. Si

[11] El estándar del lenguaje C++ que a sido desarrollado (American Nacional Standard Institute)
[12] Es un estándar para ampliar el margen de ANSI

bien el código ANSI es suficiente para los lenguajes europeos, en cambio no puede manejar algunos lenguajes orientales que tienen millares de caracteres únicos. Esa es la razón por la cual apareció UNICODE. Un carácter UNICODE tiene un tamaño de 2 bytes, haciendo posible tener 65536 caracteres únicos en las cadenas.
Sin embargo, la mayoría de las veces, usarás un archivo include que puede determinar y seleccionar las funciones de la API apropiadas para tu plataforma. Sólo referencia los nombres de las funciones de la API sin su sufijo.
Ejemplo:
Presentaré abajo el esqueleto del programa.

```
.386
.model flat, stdcall
.data
.code
start:
end start
```

La ejecución se inicia inmediatamente después de la etiqueta especificada después de la directiva end. En el esqueleto de arriba la ejecución iniciará en la primera instrucción inmediatamente debajo de la etiqueta start. La ejecución procederá instrucción por instrucción hasta encontrar algunas instrucciones de control de flujo tales como jmp, jne, je, ret, etc. Esas instrucciones redirigen el flujo de ejecución a algunas otras instrucciones. Cuando el programa necesite salir a Windows, deberá llamar a una función de la API, ExitProcess.

```
ExitProcess proto uExitCode:DWORD
```

A la línea anterior la llamamos prototipo de la función. Un prototipo de función define los atributos de una función para que el ensamblador o enlazador pueda teclear y chequear. El formato de un prototipo de función es:

```
FunctionName PROTO ParameterName]:DataType,[ParameterName]:DataType,...
```

En pocas palabras, el nombre de la función seguido por la palabra clave PROTO y luego por la lista de tipos de datos de los parámetros separados por comas. En el ejemplo de arriba de ExitProcess, se define ExitProcess como una función que toma sólo un parámetro del tipo DWORD. Los prototipos de funciones son muy útiles cuando usas sintaxis de llamadas de alto nivel, como invoke. Puedes pensar en invoke como una llamada simple con chequeo. Por ejemplo, si haces:

```
call ExitProcess
```

Sin meter en la pila un valor dword, el ensamblador o enlazador no será capaz de capturar ese error por ti. Sólo lo notarás luego cuando tu programa se bloque. Pero si usas:

```
Invoke ExitProcess
```

El enlazador te informará que olvidaste meter a la pila un valor dword y gracias a esto evitarás un error. Recomiendo que uses invoke en vez de call. La sintaxis de invoke es como sigue:

```
invoke expresión [,argumentos]
```

Expresión puede ser el nombre de una función o el nombre de un punto de función. Los parámetros de la función están separados por comas. Muchos de los prototipos de funciones para las funciones de la API se conservan en archivos include. Si usas el MASM32[13], la encontrarás en la carpeta MASM32/include. Los archivos include tienen extensión .inc y los prototipos de función para las funciones en una DLL se almacenan en archivos .inc con el mismo nombre que la DLL. Por ejemplo, ExitProcess es exportado por kernel32.lib de manera que el prototipo de función para ExitProcess está almacenado en kernel32.inc. También puedes crear prototipos para tus propias funciones.
ExitProcess, el parámetro uExitCode es el valor que quieres que el programa regrese a Windows después de que el programa termina. Puedes llamar ExitProcess de esta manera:

```
invoke ExitProcess, 0
```

Pon esa línea inmediatamente abajo de la etiqueta de inicio, y obtendrás un programa win32 que saldrá inmediatamente a Windows, pero no obstante será un programa válido.

```
.386
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
.data
.code
invoke ExitProcess, 0
start:
end start
```

La opción casemap:none dice a MASM que haga a las etiquetas sensibles a mayúsculas o minúsculas, así que ExitProcess y exitprocess son diferentes.
Una nueva directiva, include. Esta directiva es seguida por el nombre de un archivo que quieres insertar en el lugar donde se encuentra la directiva. En el código anterior, cuando MASM procese la línea.
Include \masm32\include\windows.inc, abrirá windows.inc que está en el directorio \MASM32\include y procesará el contenido de windows.inc como si pegaras ahí el contenido de windows.inc.
Windows.inc contiene definiciones de constantes y estructuras que necesitas en la programación de win32. No contiene ningún prototipo de función. windows.inc es totalmente incomprensible.
De windows.inc tus programas obtendrán las definiciones de constantes y estructuras. Pero para los prototipos de la funciones, necesitarás incluir otros

[13] Desensamblador o Ensamblador de programas ejecutables

archivos include. Puedes generar a partir de librerías de importación los archivos include que contienen sólo prototipos de funciones. A continuación mencionare los pasos para generar los archivos include: Baja las librerías para el paquete MASM32. Contiene una colección de librerías de importación que necesitas para programar para win32. También baja la utilidad l2inc.

- ✓ Desempaca ambos paquetes dentro del mismo directorio. Si instalaste MASM32, desempácalos dentro del directorio MASM32\Lib
- ✓ Ejecutar l2inca.exe con los siguientes conmutadores[14]:
  l2inca /M *.lib.

l2inca.exe extraerá información de las librerías de importación y creará archivos include llenos de prototipos de funciones.
Mueve los archivos include a la carpeta de archivos de este tipo. Los usuarios de MASM32 deberán moverlos a la carpeta MASM32\include.
En nuestro ejemplo, llamamos la función exportada por kernel32.dll, así que necesitamos incluir los prototipos de las funciones de kernel32.dll. Ese archivo es kernel32.inc. Si lo abres con un editor de texto, verás que está lleno de prototipos de funciones de kernel32.dll. Si no incluyes kernel32.inc, puedes llamar todavía a call ExitProcess pero sólo con una sintaxis simple de llamada. No podrás usar invoke para invocar la función. El punto aquí es: para invocar una función tienes que poner el prototipo de la función en alguna parte del código fuente. En el ejemplo anterior, si incluyes kernel32.inc, puedes definir los prototipos de funciones para ExitProcess en cualquier parte del código fuente antes del comando invoke para que trabaje. Los archivos include están ahí para liberarte del trabajo que significa escribirlas a cada momento que necesites llamar funciones con el comando invoke.
Ahora encontramos una nueva directiva, includelib. La cual no trabaja como include. Es la única manera que tiene tu programa de decirle al ensamblador que importe las librerías que necesita. Cuando el ensamblador ve la directiva includelib, pone un comando del enlazador dentro del mismo archivo objeto que genera, de manera que el enlazador sepa cuáles librerías necesita importar tu programa que deben ser enlazadas. Sin embargo, no estás obligado a usar includelib. Puedes especificar los nombres de las librerías de importación en la línea de comando del enlazador, es muy tedioso y la línea de comando sólo soporta 128 caracteres.
Ahora salvamos el ejemplo bajo el nombre msgbox.asm. Asumiendo que ml.exe está un tú entorno path[15], ensamblamos msgbox.asm con:
ml /c /coff /Cp msgbox.asm
/c dice a MASM que sólo ensamble, que no invoque a link.exe. Muchas veces no querrás llamar automáticamente a link.exe ya que quizás tengas que ejecutar algunas otras tareas antes de llamar a link.exe.
/coff dice a MASM que genere un archivo objeto en formato COFF[16]. MASM usa una variación de COFF.

---

[14] Referencia entre múltiples conexiones
[15] Ruta que se recorre desde el directorio Raíz a un subdirectorio especifico cuando se trata de localizar un archivo
[16] Common Object File Format = Formato de Archivo Objeto Común

Coff es usado bajo Unix como su propio formato de archivo objeto y ejecutable.
Cp dice a MASM que preserve la sensibilidad a la diferencia entre mayúsculas y minúsculas de los identificadores usados. Si usas el paquete MASM32, puedes poner option casemap:none en la cabeza del código fuente, justo debajo de la directiva .model para alcanzar el mismo efecto.
Después de haber ensamblado satisfactoriamente msgbox.asm, obtendrás msgbox.obj. msgbox.obj es un archivo objeto. Un archivo objeto está a sólo un paso del archivo ejecutable. Contiene las instrucciones o datos en forma binaria. Sólo necesitas que el enlazador fije algunas direcciones en él.
Entonces enlacemos:

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

/SUBSYSTEM:WINDOWS informa a Link qué tipo de ejecutable es este programa.
/LIBPATH:<path to import library> dice a Link dónde se encuentran las librerías de importación. Si usas MASM32, estarán en el archivo MASM32\lib.
Link lee en el archivo objeto y lo fija con las direcciones de las librerías de importación. Cuando el proceso termina obtienes msgbox.exe.
Encontrarás que no hace nada. Bien, todavía no hemos puesto nada interesante en él. Sin embargo, es un programa de Windows.
Vamos a ponerle ahora una caja de mensaje DialogBox. Su prototipo de función es:

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

hwnd: Es el manejador de la ventana padre. Puedes pensar en el manejador como un número que representa la ventana a la cual te refieres. Su valor no es tan importante para ti. Sólo recuerda que representa la ventana. Cuando quieres hacer algo con la ventana, debes referirte a ella por su manejador.
lpText: Es el puntero[17] al texto que quieres desplegar en el área cliente de la caja de mensaje.
En realidad, un puntero es la dirección de lago: Puntero a cadena de texto==Dirección de esa cadena.
lpCaption: Es un puntero al encabezado de la caja de mensaje.
uType: Especifica el icono, el número y el tipo de botones de la caja de mensajes.
Modifiquemos msgbox.asm para que incluya la caja de mensaje.

```
.386
.model flat,stdcall
option casemap:none
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
.data
MsgBoxCaption  db "HOLA MUNDO",0
MsgBoxText     db "PROGRAMA EN WIN32",0
```

[17] Variable que contiene la dirección de otra variable. Valor que representa la dirección o posición de memoria de un objeto.

```
.code
start:
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK
invoke ExitProcess, NULL
end start
```

Verás un cuadro de mensaje desplegando el texto "PROGRAMA EN WIN32".
Veamos de nuevo el código fuente.
Definimos dos cadenas terminadas en cero en la sección .data. Recuerda que toda cadena ANSI en Windows debe terminar en NULL (0 hexadecimal).
Usamos dos constantes, NULL y MB_OK. Esas constantes están documentadas en windows.inc. Así que nos referiremos a ellas por nombres y no por valores, esto facilita la lectura de nuestro código fuente. El operador addr es usado para pasar la dirección de una etiqueta a la función. Es válido sólo en el contexto de la directiva invoke. No puedes usarla para asignar la dirección de un registro o variable, por ejemplo. En vez de esto puedes usar offset en el ejemplo anterior. Sin embargo hay algunas diferencias entre los dos:
addr no puede manejar referencias delante de ella mientras que offset si puede. Por ejemplo, si la etiqueta está definida en una parte más adelante del código fuente que la directiva invoke, entonces addr no trabajará.

```
invoke MessageBox,NULL, addr MsgBoxText,addr MsgBoxCaption,MB_OK
MsgBoxCaption  db "HOLA MUNDO",0
MsgBoxText     db "PROGRAMA EN WIN32",0
```

MASM reportará error. Si usas offset en vez de addr en el recorte de código de arriba, MASM lo ensamblará.
addr puede manejar variables locales[18] mientras que offset no puede. No conocerás su dirección durante el tiempo de ejecución. offset es interpretado por el ensamblador durante el tiempo de ensamblaje. Así que es natural que offset no trabaje para variables locales. addr puede manejar variables locales debido a que el ensamblador chequea primero si la variable referida por addr es global o local. Si es una variable global, pone la dirección de la variable dentro del archivo objeto. En este aspecto, trabaja como offset. Si la variable es local, genera una secuencia de instrucciones como la siguiente antes de llamar a la función:

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

Puesto que lea puede determinar la dirección de una etiqueta en tiempo de ejecución.

2.2 UNA VENTANA SIMPLE

Los programas de Windows realizan la parte pesada del trabajo de programación a través funciones API para sus GUI[19].
Esto beneficia a los usuarios y a los programadores. A los usuarios, porque no tienen que aprender cómo navegar por la GUI de cada nuevo programa, ya que

[18] Una variable local es un espacio reservado en algún lugar de la pila
[19] Graphic User Interface = Interface de Usuario Grafica

las GUIs de los programas Windows son semejantes. A los programadores, porque tienen ya a su disposición las rutinas GUI, probadas y listas para ser usadas. El lado negativo para los programadores es la creciente complejidad involucrada. Con el fin de crear o de manipular cualquiera de los objetos GUI, tales como ventanas, menús o iconos, los programadores deben seguir una receta estricta. Pero esto puede ser superado a través de programación modular o siguiendo el paradigma de OOP[20].

A continuación describiré los pasos requeridos para crear una ventana sobre el escritorio:

- ✓ Obtener el manejador de instancia del programa.
- ✓ Lograr la línea de comando, esta no se requiere a menos que el programa vaya a procesar la línea de comando.
- ✓ Registrar la clase de ventana, requerido, al menos que vayan a usarse tipos de ventana predefinidos. MessageBox o una caja o de diálogo. Crear la ventana.
- ✓ Mostrar la ventana en el escritorio, requerido al menos que se quiera mostrar la ventana inmediatamente. Refrescar el área cliente de la ventana.
- ✓ Introducir un bucle infinito, que cheque los mensajes de Windows.
- ✓ Si llega un mensaje, es procesado por una función especial, que es responsable por la ventana. Quitar el programa si el usuario cierra la ventana.

Como puedes ver, la estructura de un programa de Windows es más compleja que la de un programa de DOS[21], ya que el mundo de Windows es totalmente diferente al mundo de DOS. Los programas de Windows deben ser capaces de coexistir pacíficamente uno junto a otro. Por eso deben seguir reglas estrictas, como programador debes ser más estricto con tus estilos y hábitos de programación.

A continuación se muestra el código fuente de nuestro programa de ventana simple. Antes de entrar en los detalles de la programación Win32 ASM, adelantaré algunos puntos delicados que facilitarán la programación.

Se deberían poner todas las constantes, estructuras y prototipos de funciones de Windows en un archivo include e incluirlo al comienzo de nuestro archivo .asm, esto nos ahorrará esfuerzos y gran cantidad de tecleado. Generalmente, el archivo include más completo para MASM es windows.inc de hutch. Puedes definir tus propias constantes y estructuras pero deberías ponerlas en un archivo include separado.
Usa la directiva includelib para especificar la librería de importación usada en tu programa. Por ejemplo, si tu programa llama a MessageBox, deberías poner la línea:
includelib user32.lib al comienzo de tu archivo .asm. Esta directiva dice a MASM que tu programa hará uso de funciones es esa librería de importación. Si tu programa llama funciones en más de una librería, entonces hay que agregar una

[20] Object Oriented Programming = Programación Orientada a Objetos.
[21] Disk Operating System = Sistema Operativo de Disco

línea includelib para cada librería que se vaya a usar. Usando la directiva includelib no tendrás que preocuparte de las librerías de importación en el momento de enlazar. Puedes usar el conmutador del enlazador /LIBPATH para decirle a Link[22] donde están todas las librerías. Cuando declares prototipos de funciones de la API, estructuras o constantes en un archivo include, trata de emplear los nombres originales usados en archivos include de Windows, cuidando siempre la diferencia entre mayúsculas y minúsculas. Esto te liberará de dolores de cabeza cuando necesites buscar información sobre algún elemento en la referencia de la API de Win32.
Usa un archivo makefile para automatizar el proceso de ensamblaje. Esto te liberará de tener que teclear más de la cuenta.

```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib      ; llamadas a las funciones en user32.lib y kernel32.lib
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
.DATA                                  ; data inicializada
ClassName db "SimpleWinClass",0        ; el nombre de nuestra clase de  ventana
AppName db "Our First Window",0        ; el nombre de nuestra ventana
.DATA?                                 ; data no inicializada
hInstance HINSTANCE ?                  ; Manejador de instancia de nuestro programa
CommandLine LPSTR ?
.CODE                                  ; Aquí comienza nuestro código
start:
invoke GetModuleHandle, NULL  ; obtener el manejador de instancia del programa.
                              ; En Win32, hmodule==hinstance
mov hInstance,eax
invoke GetCommandLine    ; Obtener la línea de comando. No hay que llamar esta función
                         ; si el programa no procesa la línea de comando
mov CommandLine,eax
invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT  ; llamar la función principal
invoke ExitProcess       ; quitar nuestro programa. El código de salida es devuelto en eax desde WinMain.
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX                  ; crear variables locales en la pila (stack)
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX     ; Llenar los valores de los miembros de wc
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInstance
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,NULL
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke RegisterClassEx, addr wc                ; registrar nuestra clase de ventana
  invoke CreateWindowEx,NULL,\
         ADDR ClassName,\
         ADDR AppName,\
         WS_OVERLAPPEDWINDOW,\
```

[22] Enlace o referencia a algo.

```
            CW_USEDEFAULT,\
            CW_USEDEFAULT,\
            CW_USEDEFAULT,\
            CW_USEDEFAULT,\
            NULL,\
            NULL,\
            hInst,\
            NULL
    mov  hwnd,eax
    Invoke ShowWindow, hwnd,CmdShow    ; desplegar nuestra ventana en el escritorio
    Invoke UpdateWindow, hwnd          ; refrescar el área cliente
    .WHILE TRUE                        ; Introducir en bucle de mensajes
            Invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            Invoke TranslateMessage, ADDR msg
            Invoke DispatchMessage, ADDR msg
    .ENDW
    mov   eax,msg.wParam               ; Regresar el código de salida en eax  ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY               ; si el usuario cierra nuestra ventana
      Invoke PostQuitMessage,NULL      ; quitar nuestra aplicación
    .ELSE
      Invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; Procesar el mensaje por defecto
      ret
    .ENDIF
    xor eax,eax
    ret
WndProc endp
end start
```

Análisis:

Puede parecer desconcertante que un simple programa de Windows requiera tanto código. Pero muchas de estas rutinas son verdaderamente un código plantilla que puede copiarse de un archivo de código fuente a otro. O si prefieres, podrías ensamblar algunas de estas rutinas en una librería para ser usadas como rutinas de prólogo o de epílogo. Puedes escribir solamente las rutinas en la función WinMain. En realidad, esto es lo que hacen los compiladores en C: permiten escribir las rutinas en WinMain sin que tengas que preocuparte de otros asuntos rutinarios y domésticos. Todo lo que hay que hacer es tener una función llamada WinMain, si no los compiladores C no serán capaces de combinar tus rutinas con el prólogo y el epilogo. No existen estas restricciones con lenguaje ensamblador. Puedes usar otros nombres de funciones en vez de WinMain o no emplear esta función en ninguna parte.
En este caso se analizara el programa:

```
.386
.model flat,stdcall
option casemap:none
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 primeras tres líneas son necesarias. .386 dice a MASM que intentamos usar en nuestro programa el conjunto de instrucciones para los procesadores 80386 o

superiores. .model flat, stdcall a MASM que nuestro programa usa el modelo de direccionamiento de memoria plana (flat). También usaremos la conversión de paso de parámetros stdcall como la conversión por defecto del programa.
Lo siguiente constituye el prototipo para la función WinMain. Ya que llamaremos más tarde a WinMain, debemos definir su prototipo de función primero para que podamos invocarle.
Debemos incluir windows.inc al comienzo del código fuente, windows.inc contiene estructuras y constantes importantes que son usadas por nuestro programa. El archivo include, windows.inc, es un archivo de texto. Puedes abrirlo con cualquier editor de texto. Por favor, nota que windows.inc no contiene todas las estructuras y constantes.

Nuestro programa llama las funciones API que residen en user32.dll CreateWindowEx, RegisterWindowClassEx, por ejemplo y kernel32.dll ExitProcess, así que debemos enlazar nuestro programa a esas librerías de importación. La próxima cuestión es: ¿cómo podemos saber cuál librería debe ser enlazada con nuestro programa? La respuesta es: debes saber donde residen las funciones API llamadas por el programa. Por ejemplo, si llamas una función API en gdi32.dll, debes enlazarla con gdi32.lib. Esta es la manera como lo hace MASM. El método de TASM[23] para importar librerías a través del enlace es mucho más simple: sólo hay que enlazar un archivo: import32.lib.

```
.DATA
   ClassName db "SimpleWinClass",0
   AppName  db "Our First Window",0
.DATA?
hInstance HINSTANCE ?
CommandLine LPSTR ?
```

Siguen las secciones "DATA".
En .DATA, declaramos dos cadenas de caracteres terminadas en cero cadenas ASCII: ClassName, que es el nombre de nuestra clase de ventana, y AppName, el nombre de nuestra ventana, las dos variables están inicializadas.
En .DATA?, son declaradas tres variables: hInstance manejador de instancia de nuestro programa, CommandLine, línea de comando de nuestro programa, y CommandShow, estado de nuestro programa en su primera aparición. Los tipos no familiares de datos, HINSTANCE y LPSTR, realmente son nombres nuevos para DWORD. Puedes verificarlo en windows.inc. Todas las variables en la sección .DATA? no están inicializadas, es decir, no tienen que tener ningún valor especifico al inicio, pero queremos reservarle un espacio para su usarlas en el futuro.

```
.CODE
 start:
   invoke GetModuleHandle, NULL
   mov   hInstance,eax
   invoke GetCommandLine
   mov   CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
```

[23] Método que enlaza archivos de tipo .Lib.

```
    invoke ExitProcess,eax
    .....
end start
```

.CODE contiene todas las instrucciones. Tus instrucciones deben residir entre <etiqueta inicio> (start) y terminar en <etiqueta inicio> (end start). El nombre de las etiquetas es importante. Puedes llamarla de la forma que quieras siempre y cuando no violes las convenciones para los nombres de MASM.

Nuestra primera instrucción llama a GetModuleHandle para recuperar el manejador de instancia de nuestro programa. Bajo Win32, el manejador de la instancia y el manejador del módulo son una y la misma cosa. Se puede pensar en el manejador de instancia como el ID[24] de nuestro programa. Es usado como parámetro en algunas funciones API que nuestro programa debe llamar, así que generalmente es una buena idea obtenerlo al comienzo del programa.
Realmente bajo win32, el manejador de instancia es la dirección lineal de nuestro programa en la memoria.
Al regresar a una función de Win32, el valor regresado, si hay alguno, puede encontrarse en el registro eax. Todos los demás valores son regresados a través de variables pasadas en la lista de parámetros de la función que va a ser llamada.
Cuando se llama a una función Win32, casi siempre preservará los registros de segmento y los registros ebx, edi, esi y ebp. Al contrario, los registros eax, ecx y edx son considerados como registros dinámicos y siempre sus valores son indeterminados e impredecibles cuando retorna una función Win32.

No esperes que los valores de eax, ecx, edx sean preservados durante las llamadas a una función API.
La línea inferior establece que: cuando se llama a una función API, se espera que regrese el valor en eax. Si cualquiera de las funciones que creamos es llamada por Windows, también debe seguir la siguiente regla: preservar y restablecer los valores de los registros de segmentos ebx, edi, esi y ebp cuando la función regrese, si no el programa se quebrará de inmediato, esto incluye el procedimiento de ventana y las funciones callback de ventanas.
La llamada a GetCommandLine es innecesaria si el programa no procesa la línea de comando. En el siguiente ejemplo muestro como llamarla en caso que sea necesario en un programa. Lo siguiente es la llamada a WinMain. Aquí recibe cuatro parámetros: el manejador de instancia de nuestro programa, el manejador de instancia de la instancia previa del programa, la línea de comando y el estado de la ventana en su primera aparición. Bajo Win32, no hay instancia previa. Cada programa está aislado en su espacio de direcciones, así que el valor de hPrevInst siempre es 0. Esto es uno de los restos de los días de Win16 cuando todas las instancias de un programa corrían en el mismo espacio de direcciones y una instancia necesitaba saber si era la primera. En win16, si hPrevInst es NULL entonces es la primera instancia.
Esta función no tiene que ser declarada como WinMain. En realidad, hay completa libertad a este respecto. Ni siquiera hay que usar siempre una función equivalente

[24] Numero de Identificación Único.

a WinMain. Se puede pegar el código dentro de la función WinMain inmediatamente después de GetCommandLine y el programa funcionará perfectamente.
Al regresar de WinMain, eax tiene el código de salida. Pasamos el código de salida como parámetro de ExitProcess, que terminará nuestra aplicación.

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

La línea de arriba forma la declaración de la función WinMain. Que los parámetros: tipo que siguen a la directiva PROC, son parámetros que WinMain recibe desde la instrucción que hace la llamada. Puedes referirte a estos parámetros por nombre en vez de la manipulación de la pila. Además, MASM generará los códigos de prólogo y epílogo para la función. Así que no tenemos que preocuparnos del marco de la pila cuando la función enter y exit.

```
LOCAL wc:WNDCLASSEX
LOCAL msg:MSG
LOCAL hwnd:HWND
```

La directiva LOCAL localiza memoria de la pila para las variables locales usadas en la función. El conjunto de directivas, LOCAL debe estar ubicado inmediatamente abajo de la directiva PROC.
La directiva LOCAL es seguida inmediatamente por <nombre de la variable local>:<tipo de variable>. Así que LOCAL wc:WNDCLASSEX le dice a MASM que localice memoria de la pila con un espacio equivalente al tamaño de la estructura WNDCLASSEX para la variable llamada wc. Podemos hacer referencia a wc en nuestro código sin ninguna dificultad en la manipulación de la pila. El aspecto negativo de esto es que las variables locales no pueden ser usadas fuera de la función porque ellas son creadas para ser destruidas inmediatamente cuando la función retorna a la rutina desde la cual fue llamada. Otra contrapartida es que no se pueden inicializar variables locales automáticamente porque ellas son localizadas dinámicamente en la memoria de la pila cuando la función es introducida. Hay que asignarlas manualmente con los valores deseados después de las directivas LOCAL.

```
mov  wc.cbSize,SIZEOF WNDCLASSEX
mov  wc.style, CS_HREDRAW or CS_VREDRAW
mov  wc.lpfnWndProc, OFFSET WndProc
mov  wc.cbClsExtra,NULL
mov  wc.cbWndExtra,NULL
   push  hInstance
   pop  wc.hInstance
mov  wc.hbrBackground,COLOR_WINDOW+1
mov  wc.lpszMenuName,NULL
mov  wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
mov  wc.hIcon,eax
mov  wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
mov  wc.hCursor,eax
   invoke RegisterClassEx, addr wc
```

En el código anterior se pueden apreciar líneas muy complejas en cuanto a concepto. Toma varias líneas de instrucciones realizar la operación ahí implicada.

El concepto detrás de todas estas líneas es el de clase de ventana (window class). Una clase de ventana no es más que un anteproyecto o especificación de una ventana. Define algunas de las características importantes de una ventana tales como un icono, su cursor, la función que se responsabiliza de ella, su color etc.

Una ventana se crea a partir de una clase de ventana, este es una especie de concepto orientado a objeto, si se quiere crear más de una ventana con las mismas características, lo razonable es almacenar todas estas características en un solo lugar y referirse a ellas cuando sea necesario. Este esquema salva gran cantidad de memoria evitando duplicación de código. Hay que recordar que Windows fue diseñado cuando los chips de memoria eran disparatados ya que una computadora tenía apenas 1 MB de memoria. Windows debía ser muy eficiente al usar recursos de memorias escasos. El punto es: si defines tu propia ventana, debes llenar las características de tu ventana en una estructura WNDCLASS o WNDCLASSEX y llamar a RegisterClass o RegisterClassEx antes de crear la ventana. Sólo hay que registrar la clase de ventana una vez para cada tipo de ventana que se quiera crear desde una clase de ventana.
Windows tiene varias clases de ventanas predefinidas, tales como botón y caja de edición. Para estas ventanas, no tienes que registrar una clase de ventana, sólo hay que llamara a CreateWindowEx con el nombre de la clase predefinido.
El miembro más importante en WNDCLASSEX es lpfnWndProc. lpfn se concibe como un puntero largo a una función. Bajo Win32, no hay puntero cercano o lejano, si no sólo puntero, debido al nuevo modelo de memoria FLAT.
Pero esto también es otro de los restos de los días de Win16. Cada clase de ventana debe estar asociada con la función llamada procedimiento de ventana.
El procedimiento de ventana es la función responsable por el manejo de mensajes de todas las ventanas creadas a partir de la clase de ventana asociada. Windows enviará mensajes al procedimiento de ventana para notificarle sobre eventos importantes concernientes a la ventana de la cual el procedimiento es responsable, tal como el uso del teclado o la entrada del ratón. Le toca al procedimiento de ventana responder inteligentemente a cada evento que recibe la ventana. Seguro que pasarás bastante tiempo escribiendo manejadores de evento en el procedimiento de ventana.
A continuación se describen los miembros de WNDCLASSEX:

```
WNDCLASSEX STRUCT DWORD
 cbSize         DWORD ?
 style          DWORD   ?
 lpfnWndProc    DWORD   ?
 cbClsExtra     DWORD   ?
 cbWndExtra     DWORD   ?
 hInstance      DWORD   ?
 hIcon          DWORD   ?
 hCursor        DWORD   ?
 hbrBackground  DWORD   ?
 lpszMenuName   DWORD   ?
 lpszClassName  DWORD   ?
 hIconSm        DWORD   ?
WNDCLASSEX ENDS
```

Donde:

cbSize: Tamaño de la estructura WNDCLASSEX en bytes. Podemos usar el operador SIZEOF para obtener este valor.
style: El estilo para las ventanas creadas a partir de esta clase. Se pueden combinar varios tipos de estilo combinando el operador "or".
lpfnWndProc: La dirección del procedimiento de ventana responsable para las ventanas creadas a partir de esta clase.
cbClsExtra: Especifica el número de bytes extra para localizar la siguiente estructura de clase de ventana. El sistema operativo inicializa los bytes poniéndolos en cero. Puedes almacenar aquí datos específicos de la clase de ventana.
cbWndExtra: Especifica el número de bytes extra para localizar the window instance. El sistema operativo inicializa los bytes poniéndolos en cero. Si una aplicación usa la estructura WNDCLASS para registrar un cuadro de diálogo creado al usar la directiva CLASS en el archivo de recurso, debe poner este miembro en DLGWINDOWEXTRA.
hInstance: Manejador de instancia del módulo.
hIcon: Manejador del icono. Se obtiene llamando a LoadIcon.
hCursor: Manejador del cursor. Se obtiene llamando a LoadCursor.
hbrBackground: Color de fondo de la ventana creada a partir de esta clase.
lpszMenuName: Manejador del menú por defecto para la ventana creada a partir de esta clase.
lpszClassName: Nombre para esta clase de ventana.
hIconSm: Manejador del icono pequeño asociado con la clase de ventana. Si este miembro es NULL, el sistema busca el recurso de icono especificado por el miembro hIcon para un icono de tamaño apropiado para ser usado como icono pequeño.

```
invoke 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 basada en la clase de ventana propuesta. Cabe señalar que hay 12 parámetros para 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
```

A continuación se describen los parámetros del código anterior.
Donde:

dwExStyle: Estilos extra de ventana. Es el nuevo parámetro agregado a la antigua función CreateWindow. Aquí puedes poner estilos nuevos para Windows 95 y NT. Puedes especificar tu estilo de ventana ordinario en dwStyle pero si quieres algunos estilos especiales tales como topmost window[25], debes especificarlos aquí. Puedes usar NULL si no quieres usar estilos de ventana extra.
lpClassName: Dirección de la cadena ASCII que contiene el nombre de la clase de ventana que quieres usar como plantilla. La Clase puede ser una clase registrada por ti mismo o una clase de ventana predefinida. Como ya se menciono, todas las ventanas que creas deben estar basadas en una clase de ventana.
lpWindowName: Dirección de la cadena ASCII que contiene el nombre de la ventana. Será mostrada en la barra de título de la ventana. Si este parámetro es NULL, la barra de título de la ventana aparecería en blanco.
dwStyle: Estilos de la ventana. Aquí puedes especificar la apariencia de la ventana. Si utilizas NULL la ventana no tendrá el menú de sistema, ni botones, minimizar o maximizar, y tampoco el botón cerrar ventana, la ventana no sería de mucha utilidad. Necesitarás presionar Alt+F4 para cerrarla. El estilo de ventana más común es WS_OVERLAPPEDWINDOW. Un estilo de ventana sólo es una bandera de un bit[26]. Así que puedes combinar varios estilos usando el operador "or" para alcanzar la apariencia deseada de la ventana.
El estilo WS_OVERLAPPEDWINDOW es realmente la combinación de muchos estilos de ventana comunes empleando este método.
Son las **X,Y:** Las coordenadas de la esquina izquierda superior de la ventana. Normalmente este valor debería ser CW_USEDEFAULT, es decir, deseas que Windows decida por ti dónde poner la ventana en el escritorio.
nWidth, nHeight: El ancho y el alto de la ventana en píxeles. También puedes usar CW_USEDEFAULT para dejar que Windows elija por ti el ancho y la altura apropiada.
hWndParent: El manejador de la ventana padre de la ventana, si existe. Este parámetro dice a Windows si esta ventana es una hija de alguna otra ventana y, si lo es, cual ventana es el padre, que no se trata del tipo de interrelación padre e hija de la MDI[27]. Las ventanas hijas no están restringidas a ocupar el área cliente de la ventana padre. Esta interrelación es únicamente para uso interno de Windows. Si la ventana padre es destruida, todas las ventanas hijas serán destruidas automáticamente. Es realmente simple. Como en nuestro ejemplo sólo hay una ventana, especificamos este parámetro como NULL.

[25] N ventanas en el Tope.
[26] Digito Binario, la unidad más pequeña de información, que tiene dos valores posibles 1 y 0.
[27] Multiple Document Interface = Interface de Documento Multiple.

hMenu: Manejador del menú de la ventana. NULL si la clase menú va a ser usada. Observa el miembro de la estructura WNDCLASSEX, lpszMenuName. lpszMenuName especifica el menú por defecto para la clase de ventana. Toda ventana creada a partir de esta clase de ventana tendrá por defecto el mismo menú, a menos que especifiques un nuevo menú imponiéndolo a una ventana especifica a través de su parámetro hMenu. hMenu es realmente un parámetro de doble propósito. En caso de que la ventana que quieras crear sea de un tipo predefinido como un control, tal control no puede ser propietario de un menú. hMenu es usado entonces más bien como un ID de control. Windows puede decidir si hMenu es realmente un manejador de menú o un ID de control revisando el parámetro lpClassName. Si es el nombre de una clase de ventana predefinida, hMenu es un ID de control, si no entonces es el manejador del menú de la ventana.
hInstance: El manejador de instancia para el módulo del programa que crea la ventana.
lpParam: Puntero opcional a la estructura pasada a la ventana. Es usada por la ventana MDI para pasar los datos CLIENTCREATESTRUCT. Normalmente, este valor es puesto en NULL, que significa que ningún dato es pasado vía CreateWindow(). La ventana puede recibir el valor de este parámetro llamando a la función GetWindowLong.

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

El regreso satisfactorio de CreateWindowEx, devuelve el manejador de ventana en eax. Debemos conservar este valor para usarlo luego. La ventana que creamos no es desplegada inmediatamente. Debes llamar a ShowWindow con el manejador de ventana y el estado de despliegue deseado para la ventana y desplegarla sobre el monitor. Luego puedes llamar a UpdateWindow con el fin de que tu ventana vuelva a dibujarse sobre el área cliente. Esta función es útil cuando se desea actualizar el contenido del área cliente. Aunque puedes omitir esta llamada.

```
.WHILE TRUE
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
      invoke TranslateMessage, ADDR msg
      invoke DispatchMessage, ADDR msg
.ENDW
```

En este momento nuestra ventana está desplegada en la pantalla. Pero no puede recibir entrada del mundo exterior. Así que tendremos que informarle de los eventos relevantes. Hacemos esto con un bucle de mensajes. Sólo hay un bucle de mensaje para cada módulo. Este bucle de mensaje chequea continuamente los mensajes de Windows llamando a GetMessage. GetMessage pasa a Windows un puntero a una estructura MSG[28]. Esta estructura MSG será llenada con información sobre el mensaje que Windows quiere enviar a una ventana en el módulo. La función GetMessage no regresará hasta que haya un mensaje para

[28] Estructura que invoca mensaje.

una ventana en el módulo. Durante ese tiempo, Windows puede darle el control a otros programas. Esto es lo que forma el esquema de multitareas cooperativas de la plataforma Win16. GetMessage regresa FALSE si el mensaje WM_QUIT es recibido en el bucle de mensaje, lo cual terminará el programa y cerrará la aplicación.
TranslateMessage es una útil función que toma la entrada desde el teclado y genera un nuevo mensaje (WM_CHAR) que es colocado en la cola de mensajes. El mensaje WM_CHAR viene acompañado del valor ASCII de la tecla presionada, el cual es más fácil de manipular que los códigos brutos de lectura de teclado. Se puede omitir esta llamada si el programa no procesa los golpes de tecla.
DispatchMessage envía los datos del mensaje al procedimiento de ventana responsable por la ventana específica a la cual va dirigido el mensaje.

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

Si termina el bucle de mensaje, el código de salida es almacenado en el miembro wParam de la estructura MSG. Puedes almacenar el código de salida en eax para regresarlo a Windows. Para estos momentos, Windows no usa el valor regresado, pero es mejor hacerlo por si acaso y para jugar con las reglas.

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

Este es nuestro procedimiento de ventana. No tienes que llamarlo necesariamente WndProc. El primer parámetro, hWnd es el manejador de la ventana hacia el cual el mensaje está destinado. uMsg es el mensaje, que uMsg no es una estructura MSG, realmente sólo es un número, Windows define cientos de mensajes, muchos de los cuales carecen de interés para nuestro programa. Windows enviará un mensaje apropiado a la ventana en caso de que ocurra algo relevante a la ventana. El procedimiento de ventana recibe el mensaje y reacciona a él inteligentemente. WParam y lParam sólo son parámetros extra a ser utilizados por algunos mensajes. Algunos mensajes envían datos junto con ellos en adición al mensaje propiamente dicho. Estos datos son pasados al procedimiento de ventana por medio de lParam y wParam.

```
   .IF uMsg==WM_DESTROY
     invoke PostQuitMessage,NULL
   .ELSE
     invoke DefWindowProc,hWnd,uMsg,wParam,lParam
     ret
   .ENDIF
   xor eax,eax
   ret
WndProc endp
```

Aquí viene la parte crucial. Es donde reside gran parte de la inteligencia de los programas. El código que responde a cada mensaje de Windows está en el procedimiento de ventana. El código debe chequear el mensaje de Windows para ver si hay un mensaje que sea de interés. Si lo es, se hace algo que se desee en respuesta a ese mensaje y luego se regresa cero en eax. Si no es así, debe

llamarse a DefWindowProc, pasando todos los parámetros recibidos para su procesamiento por defecto. DefWindowProc es una función de la API que procesa los mensajes en los que tu programa no está interesado.
El único mensaje que debes responder es WM_DESTROY. Este mensaje es enviado a tu procedimiento de ventana cada vez que la ventana se va a cerrar. En el momento que tu procedimiento de diálogo reciba este mensaje, tu ventana estará ya removida del escritorio. Este mensaje sólo es una notificación de que tu ventana ha sido destruida y de que debes prepararte para regresar a Windows. En respuesta a esto, puedes ejecutar alguna tarea doméstica antes de regresar a Windows. No queda más opción que quitar cuando la ventana llega a este estado. Si quieres tener la oportunidad de detener el usuario cuando éste intente cerrar la ventana, debes procesar el mensaje WM_CLOSE.
Ahora, regresando a WM_DESTROY, después de ejecutar las tareas domésticas, debes llamar al PostQuitMessage que enviará el mensaje WM_QUIT de vuelta a tu módulo. WM_QUIT hará que GetMessage regrese el valor NULL en eax, el cual terminará el bucle de mensajes y devolverá el control a Windows. Puedes enviar el mensaje WM_DESTROY a tu propio procedimiento de ventana llamando a la función DestroyWindow.

2.3 COLOCANDO TEXTOS

El texto en Windows es un tipo de objeto GUI. Cada carácter está compuesto por numerosos píxeles o puntos, que están dentro de un patrón son distintos. Por eso hablamos de "colocar" en vez de "escribir". Normalmente, pintas texto en tu propia área cliente, realmente, puedes también pintar fuera del área cliente. En Windows, poner texto en la pantalla es algo radicalmente distinto a DOS. En DOS, puedes pensar en la pantalla como una dimensión de 80x25. Pero en Windows, la pantalla es compartida por varios programas. Algunas reglas deben ser reforzadas para evitar que los programas escriban sobre la pantalla de otros. Windows asegura esto limitando el área para pintar de cada ventana a su área cliente solamente. El tamaño del área cliente de una ventana tampoco es constante. El usuario puede cambiarla en cualquier momento. Así que hay que determinar dinámicamente las dimensiones del área cliente de las ventanas.
Antes de que puedas pintar algo sobre el área cliente, debes pedir permiso a Windows. Eso es correcto, ya no tienes el control total sobre el monitor como lo tenías con DOS. Debes pedir permiso a Windows para pintar tu propia área cliente. Windows determinará el tamaño de tu área cliente, de la fuente, los colores y otros atributos GDI[29] y regresará un manejador del contexto de dispositivo a tu programa.
Luego puedes emplear tu contexto de dispositivo como un pasaporte para pintar tu área cliente.
¿Qué es un contexto de dispositivo? Es sólo una estructura de datos que Windows mantiene en su interior. Un contexto de dispositivo está asociado con un dispositivo en particular, tal como una impresora o un monitor de video. Para un

[29] Dinamic Interface Graphics = Interface Gráfica Dinámica.

monitor de video, un contexto de dispositivo está normalmente asociado con una ventana particular en el monitor.
Algunos valores en el contexto de dispositivo son atributos gráficos como colores y fuentes entre otros. Estos son valores por defecto que se pueden cambiar a voluntad. Existen para ayudar a reducir la carga de tener que especificar estos atributos en todas las llamadas a funciones GDI.
Puedes pensar en un contexto de dispositivo como un ambiente por defecto preparado para ti por Windows. Luego puedes anular algunos de los elementos establecidos por defecto si quieres.
Cuando un programa necesita pintar, debe obtener un manejador al contexto de dispositivo. Normalmente, hay varias maneras de realizar esto.

Llamar a BeginPaint en respuesta al mensaje WM_PAINT.
Llamar a GetDC en respuesta a otros mensajes.
Llamar a CreateDC para crear tu propio contexto de dispositivo

Hay algo que debes recordar para después de que tengas el manejador del contexto de dispositivo, y que debes realizar para el procesamiento de cualquier mensaje: no obtener el manejador en respuesta a un mensaje y emplearlo como respuesta a otro.
Windows envía mensajes WM_PAINT a la ventana para notificar que es ahora el momento de volver a pintar su área cliente. Windows no salva el contenido del área cliente de una ventana. En vez de eso, cuando ocurre una situación que garantiza que se va a volver a pintar el área cliente, tal como cuando una ventana ha sido cubierta por otra y luego descubierta, Windows pone el mensaje WM_PAINT en la cola de mensajes de ese programa. Es responsabilidad de Windows volver a pintar su propia área cliente. Debes reunir toda la información sobre cómo volver a pintar el área cliente en la sección WM_PAINT de tu procedimiento de ventana, así que tu procedimiento de ventana puede volver a pintar tu área cliente cuando llega el mensaje WM_PAINT.
Otro concepto que debes tener en consideración es el de rectángulo inválido. Windows define un rectángulo inválido como el área rectangular más pequeña que el área cliente necesita para volver a ser pintada. Cuando Windows detecta un rectángulo inválido en el área cliente de una ventana, envía un mensaje WM_PAINT a esa ventana. En respuesta al mensaje WM_PAINT, la ventana puede obtener una estructura paintstruct que contiene, entre otras cosas, la coordenada del rectángulo inválido. Puedes llamar a BeginPaint en respuesta al mensaje WM_PAINT para validar el rectángulo inválido. Si no procesas el mensaje WM_PAINT, al menos debes llamar a DefWindowProc o a ValidateRect para validar el rectángulo inválido, si no Windows te enviará repetidamente el mensaje WM_PAINT.
Estos son los pasos que deberías realizar en respuesta a un mensaje WM_PAINT:

- ✓ Obtener un manejador al contexto de dispositivo con BeginPaint.
- ✓ Pintar el área cliente. Liberar el manejador del contexto de dispositivo con EndPaint.

No tienes que validar explícitamente el rectángulo inválido. Esto es realizado automáticamente por la llamada a BeginPaint, entre las llamadas a BeginPaint y

EndPaint, puedes llamar cualquiera de las funciones GDI para pintar tu área. Casi todas ellas requieren el manejador del contexto de dispositivo como parámetro.

Ejemplo:

Escribiremos un programa que despliega una cadena con el texto "PROGRAMA EN WIN 32" en el centro del área cliente.

```
.386
.model flat,stdcall
option casemap:none
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
.DATA
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
OurText  db "PROGRAMA EN WIN 32",0
.DATA?
hInstance HINSTANCE ?
CommandLine LPSTR ?
.CODE
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,NULL
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke 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
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
    .WHILE TRUE
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR msg
        invoke 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
  .IF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_PAINT
    invoke BeginPaint,hWnd, ADDR ps
    mov   hdc,eax
    invoke GetClientRect,hWnd, ADDR rect
    invoke DrawText, hdc,ADDR OurText,-1, ADDR rect, \
        DT_SINGLELINE or DT_CENTER or DT_VCENTER
    invoke EndPaint,hWnd, ADDR ps
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax, eax
  ret
WndProc endp
end start
```

Análisis:

La mayoría del código es el mismo que el del ejemplo anterior, por lo que sólo explicaré los cambios importantes.

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

Estas variables locales son usadas por las funciones GDI en tu sección WM_PAINT. hdc es usado para almacenar el manejador al contexto de dispositivo regresado por la llamada a BeginPaint. ps es una estructura PAINTSTRUCT. Normalmente no tienes que usar los valores en ps. Es pasado a la función BeginPaint y Windows la llena con valores apropiados. Luego pasa ps a la función EndPaint cuando terminas de pintar el área cliente. rect es una estructura RECT definida así:

```
RECT Struct
  left      LONG ?
  top       LONG ?
  right     LONG ?
  bottom    LONG ?
RECT ende
```

left y top son las coordenadas de la esquina izquierda superior de un rectángulo. right y bottom son las coordenadas de la esquina derecha inferior. Debe recordarse que: El origen de los ejes "x" e "y" está en la esquina superior izquierda. Entonces el punto y=10 está debajo del punto y=0.

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

En respuesta al mensaje WM_PAINT, llamas a BeginPaint pasando como parámetros al manejador de la ventana que quieres pintar y una estructura PAINTSTRUCT no inicializada. Después de una llamada exitosa, eax contiene el manejador al contexto de dispositivo. Luego llamas a GetClientRect para recobrar la dimensión del área cliente. La dimensión es regresada en la variable rect variable que tú pasas a DrawText como uno de sus parámetros. La sintaxis de DrawText es:

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

DrawText: es una función de la API de alto nivel para salida de texto. Maneja algunos detalles tales como ajuste de línea, centramiento, etc. así que puedes concentrarte sólo en la cadena que quieres pintar. Su hermana de bajo nivel, TextOut, será analizado después. DrawText formatea una cadena de texto para fijar dentro de los límites de un rectángulo. Emplea la fuente seleccionada en el momento, color y fondo, en el contexto de dispositivo para dibujar texto. Las líneas son ajustadas para fijarla dentro de los límites del rectángulo. Regresa la altura del texto de salida en unidades de dispositivo, en nuestro caso, píxeles. Veamos sus parámetros:
Hdc: manejador al contexto de dispositivo
lpString: El puntero a la cadena que quieres dibujar en el rectángulo. La cadena debe estar terminada en NULL o si no tendrás que especificar su largo en el parámetro de texto, nCount.
nCount: El número de caracteres para salida. Si la cadena es terminada en cero, nCount debe ser -1. De otra manera nCount debe contener el número de caracteres en la cadena que quieres dibujar.
lpRect: El puntero al rectángulo, una estructura de tipo RECT, donde quieres dibujar la cadena. que este rectángulo también es un rectángulo recortante, es decir, no podrás dibujar la cadena fuera del rectángulo.
uFormat: El valor que especifica como la cadena es desplegada en el rectángulo. Usamos tres valores combinados por el operador "or":
DT_SINGLELINE: especifica una línea de texto
DT_CENTER: centra el texto horizontalmente.
DT_VCENTER: centra el texto verticalmente.
Debe ser usado con DT_SINGLELINE.
Después de terminar de pintar el área cliente, debes llamar a la función EndPaint para liberar el manejador del contexto de dispositivo. Eso es todo. Podemos hacer un sumario de los puntos relevantes:
Llamas a BeginPaint y EndPaint en respuesta el mensaje WM_PAINT. Haces lo que gustes con el área cliente de la ventana entre las llamadas a las funciones BeginPaint y EndPaint. Si quieres volver a pintar tu área cliente en respuesta a otros mensajes, tienes dos posibilidades:

- ✓ Usar el par GetDC-ReleaseDC y pintar entre estas dos llamadas
- ✓ Llamar a InvalidateRect o a UpdateWindow para invalidar toda el área cliente, forzando a Windows a que ponga un mensaje WM_PAINT en la cola de mensajes de tu ventana y pinte durante la sección WM_PAINT

2.4 SISTEMA DE COLORES

El sistema de colores de Windows está basado en valores RGB[30], R=red (rojo), G=Green (verde), B=Blue (azul). Si quieres especificar un color en Windows, debes establecer el color que desees en términos de estos tres colores mayores. Cada valor de color tiene un rango desde 0 a 255, un valor de un byte. Por ejemplo, si quieres un color rojo puro, deberías usar 255, 0, 0. O si quieres un color blanco puro, debes usar 255, 255, 255. Puedes ver en los ejemplos que obtener el color que necesitas es muy difícil con este sistema ya que tienes que tener una buena comprensión de como mezclar y hacer corresponder los colores. Para el color del texto y del fondo, usas SetTextColor y SetBkColor, que requieren un manejador al contexto del dispositivo y un valor RGB de 32-bit. La estructura dvalor RGB de 32-bit está definida así:

```
RGB_value struct
  unused  db 0
  blue    db ?
  green   db ?
  red     db ?
RGB_value ends
```

No se emplea el primer byte y que debería ser cero. El orden de los restantes tres bytes es inverso, es decir azul, verde y rojo. Sin embargo, no usaremos esta estructura ya que es embarazoso inicializarla y usarla. Más bien crearemos una macro. La macro recibirá tres parámetros: los valores rojo, verde y azul. Esto producirá el valor RGB 32-bit deseado y lo almacenará en eax. La macro es como sigue a continuación:

```
RGB macro red,green,blue
  xor  eax,eax
  mov  ah,blue
  shl  eax,8
  mov  ah,green
  mov  al,red
endm
```

Puedes poner esta macro en el archivo include para usarla en el futuro. Puedes crear, una fuente llamando a CreateFont o a CreateFontIndirect, la diferencia entre las dos funciones es que CreateFontIndirect recibe sólo un parámetro: un puntero a la estructura lógica de la fuente, LOGFONT. CreateFontIndirect es la más flexible de las dos, especialmente si tus programas necesitan cambiar de fuentes con frecuencia. Sin embargo, como en nuestro ejemplo crearemos sólo una fuente para demostración, podemos hacerlos con CreateFont. Después de llamada a CreateFont, regresará un manejador a la fuente que debes seleccionar dentro del contexto de dispositivo. Después de eso, toda función de texto de la API usará la fuente que hemos seleccionado dentro del contexto de dispositivo.

[30] Red, Green, Blue

Ejemplo:

```
.386
.model flat,stdcall
option casemap:none
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 red,green,blue
      xor eax,eax
      mov ah,blue
      shl eax,8
      mov ah,green
      mov al,red
endm
.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
TestString  db "PROGRAMA EN WIN 32",0
FontName db "script",0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
.code
 start:
   invoke GetModuleHandle, NULL
   mov   hInstance,eax
   invoke GetCommandLine
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,NULL
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke 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
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
   invoke UpdateWindow, hwnd
   .WHILE TRUE
            invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            invoke TranslateMessage, ADDR msg
            invoke 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
  .IF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_PAINT
    invoke BeginPaint,hWnd, ADDR ps
    mov   hdc,eax
    invoke CreateFont,24,16,0,0,400,0,0,0,OEM_CHARSET,\
                      OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,\
                      DEFAULT_QUALITY,DEFAULT_PITCH or FF_SCRIPT,\
                      ADDR FontName
    invoke SelectObject, hdc, eax
    mov   hfont,eax
    RGB   200,200,50
    invoke SetTextColor,hdc,eax
    RGB   0,0,255
    invoke SetBkColor,hdc,eax
    invoke TextOut,hdc,0,0,ADDR TestString,SIZEOF TestString
    invoke SelectObject,hdc, hfont
    invoke EndPaint,hWnd, ADDR ps
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax,eax
  ret
WndProc endp
end start
```

Análisis:

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

CreateFont creará la fuente lógica que más coincida con los parámetros dados y con los datos de fuentes disponibles. Esta función tiene más parámetros que cualquier otra función en Windows. Regresa un manejador a la fuente lógica a ser usada por la función SelectObject. Examinaremos sus parámetros en detalle.

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

nHeight: La altura deseada para los caracteres. 0 significa el tamaño por defecto.
nWidth: El ancho deseado para los caracteres. Normalmente este valor debería ser 0 que permite a Windows coordinar el ancho y el alto. Sin embargo, en nuestro

ejemplo, el ancho por defecto hace difícil la lectura del texto, así que usaremos mejor un ancho de 16.
nEscapement: Especifica la orientación del próximo carácter de salida relativa al previo en décimas de grados. Normalmente, se pone en 0. Si se pone en 900 tendremos que todos los caracteres irán por encima del primer carácter, 1800 los escribirá hacia atrás, o 2700 para escribir cada carácter desde abajo.
nOrientation: Especifica cuánto debería ser rotado el carácter cuando tiene una salida en décimas de grados. Si se pone en 900 todos los caracteres reposaran sobre sus respaldos, 1800 se emplea para escribirlos upside-down, etc.
nWeight: Establece el grosor de las líneas de cada carácter. Windows define los siguientes tamaños:

```
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 itálicas.
cUnderline: 0 para normal, cualquier otro valor para caracteres subrayados.
cStrikeOut: 0 para normal, cualquier otro valor para caracteres con una línea a través del centro.
cCharSet: Especifica la configuración de la fuente. Normalmente debería ser OEM_CHARSET lo cual permite a Windows seleccionar la fuente dependiente del sistema operativo.
cOutputPrecision: Especifica cuando la fuente seleccionada debe coincidir con las características que queremos. Normalmente debería ser OUT_DEFAULT_PRECIS que define la conducta de proyección por defecto de la fuente.
cClipPrecision: Especifica la precisión del corte. La precisión del corte define cómo recortar los caracteres que son parcialmente fuera de la región de recorte. Deberías poder obtenerlo con CLIP_DEFAULT_PRECIS que define la conducta por defecto del recorte.
cQuality: Especifica la cualidad de la salida. La cualidad de salida define cuán cuidadosamente la GDI debe intentar hacer coincidir los atributos de la fuente lógica con los de la fuente física actual. Hay tres posibilidades:
DEFAULT_QUALITY, PROOF_QUALITY y DRAFT_QUAL.
cPitchAndFamily: Especifica la pitch y familia de la fuente. Debes combinar el valor pitch valué y el valor de familia con el operador "or".
lpFacename: Un puntero a una cadena terminada en cero que especifica la tipografía de la fuente.

La descripción anterior no tiene nada de comprensible. Deberías revisar la referencia del API de Win32 API para más detalles.

```
invoke SelectObject, hdc, eax
mov   hfont,eax
```

Después de obtener el manejador lógico de la fuente, debemos usarlo para seleccionar la fuente dentro del contexto de dispositivo llamando a SelectObject. SelectObject pone los nuevos objetos GDI tales como bolígrafos, brochas, y fuentes dentro del contexto de dispositivo a ser usado por las funciones GDI. Este regresa el manejador del objeto reemplazado, el cual deberíamos salvar para su uso posterior a la llamada a SelectObject. Después de llamar a SelectObject, cualquier función de salida de texto, usará la fuente que seleccionamos dentro del contexto de dispositivo.

```
RGB   200,200,50
invoke SetTextColor,hdc,eax
RGB   0,0,255
invoke SetBkColor,hdc,eax
```

Usa la macro RGB para crear un valor de 32-bit RGB para ser usado por SetColorText y SetBkColor.

```
invoke TextOut,hdc,0,0,ADDR TestString,SIZEOF TestString
```

Llama a la función TextOut para dibujar el texto sobre el área cliente. El texto estará en la fuente y el color que especificamos previamente.

```
invoke SelectObject,hdc, hfont
```

Cuando estemos trabajando con fuentes, deberíamos almacenar la fuente original dentro del contexto de dispositivo. Por lo que deberías almacenar siempre el objeto que reemplazaste en el contexto de dispositivo.

2.5 ENTRADA DEL TECLADO

Como normalmente sólo hay un teclado para cada PC, todos los programas de Windows deben compartirlo entre sí. Windows es responsable de enviar los golpes de tecla a la ventana que tiene el foco de entrada.
Aunque puede haber varias ventanas en el monitor, sólo una de ellas tiene el foco de entrada. La ventana que tiene el foco de entrada es la única que puede recibir los golpes de tecla. Puedes diferenciar la ventana que tiene el foco de entrada de las otras ventanas observando la barra de título. La barra de título del programa que tiene el foco está iluminada.
Realmente, hay dos tipos principales de mensajes de teclado, dependiendo de tu punto de vista sobre el teclado. Puedes ver el teclado como una colección de teclas. En este caso, si presionas una tecla, Windows envía un mensaje WM_KEYDOWN a la ventana que tiene el foco de entrada, que notifica que una tecla ha sido presionada. Cuando sueltas la tecla, Windows envía un mensaje WM_KEYUP.

Otra manera de ver el teclado es como un dispositivo de entrada de caracteres. Cuando presionas una tecla, Windows envía un mensaje WM_CHAR a la ventana que tiene el foco de entrada, diciéndole que el usuario envía un carácter a ella. En realidad, Windows envía mensajes WM_KEYDOWN y WM_KEYUP a la ventana que tiene el foco de entrada y esos mensajes serán traducidos a mensajes WM_CHAR por una llamada a TranslateMessage. El procedimiento de ventana puede decidir si procesa los tres mensajes o sólo los mensajes que interesan. Muchas veces, podrás ignorar WM_KEYDOWN y WM_KEYUP ya que la función TranslateMessage en el bucle de mensajes traduce los mensajes. WM_KEYDOWN y WM_KEYUP a mensajes WM_CHAR. Por lo cual nos concentraremos en WM_CHAR.

Ejemplo:

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
char WPARAM 20h                 ; el carácter que el programa recibe del teclado
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,NULL
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke 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
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
  .WHILE TRUE
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR msg
        invoke 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
  .IF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_CHAR
    push wParam
    pop  char
    invoke InvalidateRect, hWnd,NULL,TRUE
  .ELSEIF uMsg==WM_PAINT
    invoke BeginPaint,hWnd, ADDR ps
    mov   hdc,eax
    invoke TextOut,hdc,0,0,ADDR char,1
    invoke EndPaint,hWnd, ADDR ps
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax,eax
  ret
WndProc endp
end start
```

Análisis:

```
char WPARAM 20h                ; el carácter que el programa recibe del teclado
```

Esta es la variable que guardará el carácter recibido del teclado. Como el carácter es enviado en WPARAM del procedimiento de ventana, por simplicidad definimos los tipos de variables como WPARAM. El valor inicial es 20h o el espacio, ya que cuando nuestra ventana refresque su área cliente por primera vez, ahí no habrá carácter de entrada. Así que preferimos desplegar el espacio.

```
  .ELSEIF uMsg==WM_CHAR
    push wParam
    pop  char
    invoke InvalidateRect, hWnd,NULL,TRUE
```

Esto es agregado al manejador del mensaje WM_CHAR en el procedimiento de ventana. Pone el carácter dentro de la variable llamada char y luego llama a InvalidateRect. InvalidateRect hace que le rectángulo específico en el área cliente quede invalidado para forzar a Windows para que envíe el mensaje WM_PAINT al procedimiento de ventana. Su sintaxis es como sigue:

```
InvalidateRect proto hWnd:HWND,\
                     lpRect:DWORD,\
                     bErase:DWORD
```

lpRect: es un puntero al rectángulo en el área cliente que queremos declarar inválida.

Si este parámetro es nulo, toda el área cliente será marcada como inválida.
bErase: es una bandera que dice a Windows si necesita borrar el fondo. Su ventana es TRUE, luego Windows borrará el fondo del rectángulo invalidado cuando se llama a BeginPaint.
Así que la estrategia que usamos aquí es: almacenamos toda la información necesaria involucrada en la acción de pintar el área cliente y generar el mensaje WM_PAINT para pintar el área cliente. Por supuesto, el código en la sección WM_PAINT debe saber de antemano qué se espera de ella. Esto parece una manera indirecta de hacer las cosas, pero así es como lo hace Windows.
Realmente podemos pintar el área cliente durante el proceso del mensaje WM_CHAR llamando el par de funciones GetDC y ReleaseDC. No hay problema. Pero lo gracioso comienza cuando nuestra ventana necesita volver a pintar su área cliente. Como el código que pinta el carácter está en la sección WM_CHAR, el procedimiento de ventana no será capaz de pintar nuestro carácter en el área cliente. Así que la línea de abajo es: poner todo el código y los datos necesarios para que realicen la acción de pintar en WM_PAINT. Puedes enviar el mensaje WM_PAINT desde cualquier lugar de tu código cada vez que quieras volver a pintar el área cliente.

```
invoke TextOut,hdc,0,0,ADDR char,1
```

Cuando se llama a InvalidateRect, envía un mensaje WM_PAINT de regreso al procedimiento de ventana. De esta manera es llamado el código en la sección WM_PAINT. Llama a BeginPaint como es usual para obtener el manejador al contexto del dispositivo y luego llama a TextOut que dibuja nuestro carácter en el área cliente en x=0, y=0. Cuando corres el programa y presionas cualquier tecla, verás un eco del carácter en la esquina izquierda superior del área cliente de la ventana. Y cuando la ventana sea minimizada, al ser maximizada de nuevo tendrá todavía el carácter ahí ya que todo el código y los datos esenciales para volver a pintar son todos activados en la sección WM_PAINT.

2.6 ENTRADA DEL RATÓN

Windows detecta y envía notificaciones sobre las actividades del ratón que son relevantes para las ventanas. Esas actividades incluyen los clicks de los botones, izquierdo y derecho del ratón, el movimiento del cursor del ratón sobre la ventana, doble clicks. A diferencia de la entrada del teclado, que es dirigida a la ventana que tiene el foco de entrada, los mensajes del ratón son enviados a cualquier ventana sobre la cual esté el cursor del ratón, activo o no. Además, también hay mensajes del ratón sobre el área no cliente. Pero la mayoría de las veces, afortunadamente podemos ignorarlas. Podemos concentrarnos en los mensajes relacionados con el área cliente.
Hay dos mensajes para cada botón del ratón: los mensajes WM_LBUTTONDOWN, WM_RBUTTONDOWN y WM_LBUTTONUP, WM_RBUTTONUP. Para un ratón con tres botones, están también WM_MBUTTONDOWN y WM_MBUTTONUP. Cuando el cursor del ratón se mueve sobre el área cliente, Windows envía mensajes WM_MOUSEMOVE a la

ventana debajo del cursor. Una ventana puede recibir mensajes de doble clicks, WM_LBUTTONDBCLK o WM_RBUTTONDBCLK si y sólo si la clase de su ventana tiene activada la bandera correspondiente al estilo CS_DBLCLKS, si no la ventana recibirá sólo una serie de mensajes del tipo botón del ratón arriba o abajo. Para todos estos mensajes, el valor de lParam contiene la posición del ratón. La palabra baja es la coordenada "x", y la palabra alta es la coordenada "y" relativa a la esquina izquierda superior del área cliente de la ventana. wParam indica el estado de los botones del ratón y de las teclas Shift y Ctrl.

Ejemplo:

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MouseClick db 0        ; 0=no click yet
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hitpoint POINT <>
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,NULL
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke 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
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
  .WHILE TRUE
```

```
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
        invoke 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
  .IF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_LBUTTONDOWN
    mov eax,lParam
    and eax,0FFFFh
    mov hitpoint.x,eax
    mov eax,lParam
    shr eax,16
    mov hitpoint.y,eax
    mov MouseClick,TRUE
    invoke InvalidateRect,hWnd,NULL,TRUE
  .ELSEIF uMsg==WM_PAINT
    invoke BeginPaint,hWnd, ADDR ps
    mov   hdc,eax
    .IF MouseClick
      invoke lstrlen,ADDR AppName
      invoke TextOut,hdc,hitpoint.x,hitpoint.y,ADDR AppName,eax
    .ENDIF
    invoke EndPaint,hWnd, ADDR ps
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax,eax
  ret
WndProc endp
end start
```

Análisis:

```
  .ELSEIF uMsg==WM_LBUTTONDOWN
    mov eax,lParam
    and eax,0FFFFh
    mov hitpoint.x,eax
    mov eax,lParam
    shr eax,16
    mov hitpoint.y,eax
    mov MouseClick,TRUE
    invoke InvalidateRect,hWnd,NULL,TRUE
```

El procedimiento de ventana espera a que el botón izquierdo del ratón haga un click. Cuando recibe el mensaje WM_LBUTTONDOWN, lParam contiene la coordenada del botón del ratón en el área cliente. Salva la coordenada en la variable de tipo POINT definida así:

```
POINT STRUCT
  x  dd ?
  y  dd ?
POINT ENDS
```

Y establece la bandera, MouseClick, a TRUE, lo que significa que al menos hay un click del botón izquierdo del ratón sobre el área cliente.

```
mov eax,lParam
and eax,0FFFFh
 mov hitpoint.x,eax
```

Como la coordenada 'x' es la palabra baja de lParam y los miembros de la estructura POINT tiene un tamaño de 32-bits, debemos poner en cero la palabra alta de eax antes de almacenarla en hitpoint.x.

```
shr eax,16
 mov hitpoint.y,eax
```

Como la coordenada "y" es la palabra alta de lParam, debemos ponerla en la palabra baja de eax antes de almacenarla en hitpoint.y. Hacemos esto desplazando el contenido de eax 16 bits a la derecha.
Después de almacenar la posición del ratón, establecemos la bandera, MouseClick, a TRUE con el fin de dejar que el código de pintura en la sección WM_PAINT sepa que hay al menos un click en el área cliente y puede dibujar la cadena en la posición del ratón. Luego llamamos a la función InvalidateRect para que Windows vuelva a pintar toda el área cliente.

```
  .IF MouseClick
  invoke lstrlen,ADDR AppName
  invoke TextOut,hdc,hitpoint.x,hitpoint.y,ADDR AppName,eax
.ENDIF
```

El código de pintura en la sección WM_PAINT debe chequear si MouseClick es uno (TRUE), ya que cuando la ventana fue creada, recibió un mensaje WM_PAINT en ese momento, ningún click del ratón había ocurrido aún, así que no dibujará la cadena en el área cliente. Inicializamos MouseClick a FALSE y cambiamos su valor a TRUE cuando ocurre un click del ratón.
Si ha ocurrido al menos un click de ratón, se dibuja la cadena en el área cliente en la posición del ratón.
Se llama a lstrlen para obtener el tamaño de la cadena a desplegar y envía este tamaño como último parámetro de la función TextOut.

2.7 CREANDO MENÚS

El menú es uno de los componentes más importantes en nuestra ventana. El menú presenta una lista de los servicios que un programa ofrece a un usuario. El usuario ya no tiene que leer el manual incluido con el programa para utilizarlo, ya que puede leerse cuidadosamente el menú para obtener una visión general de las capacidades de un programa particular y comenzar a trabajar con él inmediatamente. Como el menú es una herramienta para obtener el acercamiento del usuario y correr el programa rápidamente, se debería seguir siempre el estándar. Puesto brevemente, los primeros dos elementos del menú deberían ser Archivo y Editar y el último debería ser Ayuda. Puedes insertar tus propios elementos de menú entre Editar y Ayuda. Si un elemento de menú invoca una caja de diálogo, deberías anexar un paréntesis (...) a la cadena del menú.
El menú es un tipo de recurso. Hay varios tipos de recursos, tales como dialog box, string table, icon, bitmap, menú etc. Los recursos son descritos en un archivo

separado llamado archivo de recursos, el cual generalmente tiene extensión .rc. Luego combinas los recursos del archivo fuente durante el estado de enlace. El resultado final es un archivo ejecutable que contiene tanto instrucciones como recursos.
Puedes escribir guiones (scripts) de recursos usando un editor de texto. Estos guiones están compuestos por frases que describen la apariencia y otros atributos de los recursos usados en un programa particular. Aunque puedes escribir guiones de recursos con un editor de texto, esto resulta más bien embarazoso. Una mejor alternativa es usar un editor de recursos que te permita visualizar con facilidad el diseño de los recursos. Usualmente los paquetes de compiladores como Visual C++, Borland C++, etc., incluyen editores de recursos
Y a continuación se describen los recursos de esta forma:

```
Mymenu MENU
{
  [menu list here]
}
```

Los programadores en lenguaje C pueden reconocer que es similar a la declaración de una estructura. MyMenu sería un nombre de menú seguido por la palabra clave MENU y una lista de menú entre llaves. Alternativamente, si quieres puedes usar BEGIN y END en vez de las llaves. Esta sintaxis es más portable para los programadores en Pascal.
La lista de menú puede ser un enunciado MENUITEM o POPUP.
El enunciado MENUITEM define una barra de menú que no invoca un menú emergente, cuando es seleccionado. La sintaxis es como sigue:

```
MENUITEM "&text", ID [,options]
```

Se comienza por la palabra clave MENUITEM seguida por el texto que quieres usar como cadena de texto de la barra de menú. El ampersand (&), hace que el carácter que le sigue sea subrayado.
Después de la cadena de texto está el ID del elemento de menú. El ID es un número que será usado para identificar el elemento de menú respectivo en el mensaje enviado al procedimiento de ventana cuando el elemento de menú es seleccionado. Como tal, cada ID de menú debe ser único entre ellos.
Las opciones disponibles son:
GRAYED: El elemento del menú está inactivo y no genera un mensaje **WM_COMMAND:** El texto está desvanecido.
INACTIVE: El elemento del menú está inactivo y no genera un mensaje **WM_COMMAND:**. El texto es desplegado normalmente.
MENUBREAK: Este elemento y los siguientes aparecen en una línea nueva del menú.
HELP: Este elemento y los siguientes están justificados a la derecha.
Puedes usar una de las opciones de arriba o combinarlas con el operador "or". Sólo recuerda que INACTIVE y GRAYED no pueden ser combinados simultáneamente.

El enunciado POPUP tiene la siguiente sintaxis:

```
POPUP "&text" [,options]
{
  [menu list]
}
```

El enunciado POPUP define una barra de menú que, cuando es seleccionada, despliega una lista de elementos de menú en una ventana emergente. La lista de menú puede ser un enunciado MENUTIEM o POPUP. Hay un tipo especial de enunciado MENUITEM, MENUITEM SEPARATOR, que dibuja una línea horizontal en la ventana emergente.
El paso siguiente, después de haber terminado con el guión de recursos, es hacer la referencia a él en el programa.
Esto se puede hacer de dos maneras. En el miembro lpszMenuName de la estructura WNDCLASSEX. Es decir, si tienes un menú llamado "FirstMenu", puedes asignar este menú a tu ventana de la siguiente manera:

```
.DATA
MenuName  db "FirstMenu",0
.....
.CODE
.....
 mov   wc.lpszMenuName, OFFSET MenuName
.....
```

En el parámetro del manejador del menú de CreateWindowEx más o menos así:

```
.DATA
MenuName  db "FirstMenu",0
hMenu HMENU ?
..... .....
.CODE
.....
invoke LoadMenu, hInst, OFFSET MenuName
mov   hMenu, eax
invoke CreateWindowEx,NULL,OFFSET ClsName,\
        OFFSET Caption, WS_OVERLAPPEDWINDOW,\
        CW_USEDEFAULT,CW_USEDEFAULT,\
        CW_USEDEFAULT,CW_USEDEFAULT,\
        NULL,\
       hMenu,\
        hInst,\
        NULL\
.....
```

Entonces podrías preguntarte, ¿cuál es la diferencia entre estos dos métodos?
Cuando haces referencia al menú en la estructura WNDCLASSEX, el menú llega a ser el menú por defecto para la clase de ventana. Todas las ventanas de esa clase tendrán el mismo menú.
Si quieres que cada ventana creada a partir de la misma clase tenga diferente menú, debes elegir la segunda manera. En este caso, cualquier ventana que se le pase un manejador de menú en su función CreateWindowEx tendrá un menú que reemplazará el menú por defecto definido en la estructura WNDCLASSEX.
Ahora examinaremos como un menú notifica al procedimiento de ventana cuando el usuario selecciona un elemento del menú.

Cuando el usuario selecciona un elemento del menú, el procedimiento de ventana recibirá un mensaje WM_COMMAND. La palabra baja de wParam contendrá el ID del elemento del menú.
Ahora tenemos suficiente información para usar el menú.

Ejemplo:

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

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MenuName db "FirstMenu",0             ; Nombre de nuestro menú en el archivo .RC
Test_string db "You selected Test menu item",0
Hello_string db "Hello, my friend",0
Goodbye_string db "See you again, bye",0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
.const
IDM_TEST equ 1                ; Menu IDs
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push  hInst
  pop   wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,OFFSET MenuName      ; Poner aquí el nombre del Menú
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke 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
```

```
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
            invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .IF ax==IDM_TEST
        invoke MessageBox,NULL,ADDR Test_string,OFFSET AppName,MB_OK
      .ELSEIF ax==IDM_HELLO
        invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
      .ELSEIF ax==IDM_GOODBYE
        invoke MessageBox,NULL,ADDR Goodbye_string, OFFSET AppName, MB_OK
      .ELSE
        invoke DestroyWindow,hWnd
      .ENDIF
    .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
    .ENDIF
    xor   eax,eax
    ret
WndProc endp
end start

Menu.rc
#define IDM_TEST 1
#define IDM_HELLO 2
#define IDM_GOODBYE 3
#define IDM_EXIT 4
FirstMenu MENU
{
 POPUP "&PopUp"
      {
       MENUITEM "&Say Hello",IDM_HELLO
       MENUITEM "Say &GoodBye", IDM_GOODBYE
       MENUITEM SEPARATOR
       MENUITEM "E&xit",IDM_EXIT
      }
 MENUITEM "&Test", IDM_TEST
}
```

Análisis:

Vamos a analizar primero el guión de recursos.

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

Las líneas de arriba definen los IDs de menú usados por el guión de menú. Puedes asignar cualquier valor al ID siempre que sea único en el menú.
FirstMenu MENU
Declarar tu menú con la palabra clave MENU.

```
POPUP "&PopUp"
    {
     MENUITEM "&Say Hello",IDM_HELLO
     MENUITEM "Say &GoodBye", IDM_GOODBYE
     MENUITEM SEPARATOR
     MENUITEM "E&xit",IDM_EXIT
    }
```

Definir un menú emergente con cuatro elementos, donde el tercer elemento es un separador.

```
MENUITEM "&Test", IDM_TEST
```

Definir una barra de menú en el menú principal.
Ahora examinaremos el código fuente.

```
MenuName db "FirstMenu",0           ; Nombre del menú en el archivo de recursos.
Test_string db "You selected Test menu item",0
Hello_string db "Hello, my friend",0
Goodbye_string db "See you again, bye",0
```

MenuName es el nombre del menú en el archivo de recursos.
Puedes definir más de un menú en el archivo de recursos así que debes especificar cual usar.
Las restantes tres líneas definen la cadena de texto a ser desplegada en las cajas de mensaje que son invocadas cuando el elemento de menú apropiado, seleccionado por el usuario

```
IDM_TEST equ 1              ; Menu IDs
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4
```

Definir IDs de elementos de menú para usar 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
  .IF ax==IDM_TEST
     invoke MessageBox,NULL,ADDR Test_string,OFFSET AppName,MB_OK
  .ELSEIF ax==IDM_HELLO
     invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
  .ELSEIF ax==IDM_GOODBYE
     invoke MessageBox,NULL,ADDR Goodbye_string, OFFSET AppName, MB_OK
  .ELSE
     invoke DestroyWindow,hWnd
  .ENDIF
```

En el procedimiento de ventana, procesamos mensajes WM_COMMAND. Cuando el usuario seleccione un elemento de menú, el ID de ese elemento de menú es enviado al procedimiento de ventana en la palabra baja de wParam junto con el mensaje WM_COMMAND. Así que cuando almacenamos el valor de wParam en eax, comparamos el valor en ax con los IDs de menú que definimos previamente y actuamos de acuerdo con ello. En los primeros tres casos, cuando el usuario selecciona los elementos de menú Test, Say Hello, y Say GoodBye, desplegamos una cadena de texto en la caja de mensaje.

Si el usuario selecciona el elemento de menú Exit, llamamos DestroyWindow con el manejador de nuestra ventana como su parámetro que cerrará nuestra ventana. Como puedes ver, especificar el nombre del menú en una clase de ventana es muy fácil y directo. Sin embargo, también puedes usar un método alternativo para cargar un menú en tu ventana. No mostraré aquí todo el código fuente. El archivo de recursos es el mismo en ambos métodos. Hay algunos cambios menores en el archivo fuente que se mostraran a continuación.

```
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hMenu HMENU ?                ; handle of our menu
```

Definir un tipo de variable HMENU para almacenar nuestro manejador de menú.

```
    invoke LoadMenu, hInst, OFFSET MenuName
    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, llamamos a LoadMenu con el manejador de instancia del programa y un puntero al manejador de nuestro menú. LoadMenu regresa el manejador de nuestro menú en el archivo de recursos que se pasó a CreateWindowEx.

2.8 CONTROLES DE VENTANAS HIJAS

Windows provee algunas clases de ventana predefinidas que podemos usar satisfactoriamente dentro de nuestros programas. Muchas veces las usamos como componentes de una caja de diálogo por lo que ellas usualmente son llamadas controles de ventanas hijas. Los controles de ventanas hijas procesan sus propios mensajes de teclado y de ratón y notifican a las ventanas padres cuando sus estados han cambiado. Ellos liberan al programador de enormes cargas, así que deberías usarlas cada vez que sea posible. Las pongo sobre una ventana normal para demostrar cómo puedes crearlas y usarlas, pero en realidad deberías ponerlas en una caja de diálogo.
Ejemplos de clases de ventanas predefinidas son el botón, la caja de lista, la caja de chequeo, botón de radio, edición, etc.
Con el fin de usar el control de ventana hija, debes crearla con CreateWindow o CreateWindowEx. No tienes que registrar la clase de ventana puesto que Windows lo hace por ti. El parámetro nombre de la clase debe ser el nombre de la clase predefinida. Es decir, si quieres crear un botón, debes especificar button como nombre de la clase en CreateWindowEx. Los otros parámetros que debes llenar son la agarradera o manejador de la ventana padre y el ID del control. El ID del control debe ser único entre los controles, el ID del control es el ID de ese control, lo usas para diferenciar entre controles.

Después de que el control fue creado, enviará mensajes de notificación a la ventana padre cuando su estado cambie. Normalmente, creas las ventanas hijas durante el mensaje WM_CREATE de la ventana padre. La ventana hija 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 manejador de ventana en lParam. Cada control de ventana hija tiene su propio código de notificación, así que debes revisar la referencia de la API de Win32 para más información.
También la ventana padre puede enviar órdenes o comandos a la ventana hija, llamando a la función SendMessage. Esta función envía el mensaje especificado acompañado de otros valores en wParam y lParam a la ventana especificada por el manejador de ventana. Es una función extremadamente útil, ya que puede enviar mensajes a cualquier ventana que conozcas su manejador.
Así que después de crear ventanas hijas, la ventana padre debe procesar los mensajes WM_COMMAND para poder recibir códigos de notificación desde las ventanas hijas.

Ejemplo:
Crearemos una ventana que contenga un control de edición y un pushbutton. Cuando pulses el botón, un cuadro de mensaje aparecerá mostrando un texto que hayas escrito en el cuadro de diálogo. Hay también un menú con 4 elementos:
Say Hello -- Pone una cadena de texto en la caja de edición
Clear Edit Box -- Limpia el contenido de la caja de edición
Get Text -- despliega una caja de mensajes con el texto en la caja de edición
Exit -- Cierra el programa.

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MenuName db "FirstMenu",0
ButtonClassName db "button",0
ButtonText db "My First Button",0
EditClassName db "edit",0
TestString db "Wow! I'm in an edit box now",0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwndButton HWND ?
hwndEdit HWND ?
buffer db 512 dup(?)   ; buffer para almacenar el texto recuperado desde la ventana de edición
.const
ButtonID equ 1                  ; El ID del control botón
EditID equ 2                    ; El ID del control de edición
IDM_HELLO equ 1
IDM_CLEAR equ 2
IDM_GETTEXT equ 3
IDM_EXIT equ 4
```

```
.code
start:
  invoke GetModuleHandle, NULL
  mov    hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov   wc.cbSize,SIZEOF WNDCLASSEX
  mov   wc.style, CS_HREDRAW or CS_VREDRAW
  mov   wc.lpfnWndProc, OFFSET WndProc
  mov   wc.cbClsExtra,NULL
  mov   wc.cbWndExtra,NULL
  push  hInst
  pop   wc.hInstance
  mov   wc.hbrBackground,COLOR_BTNFACE+1
  mov   wc.lpszMenuName,OFFSET MenuName
  mov   wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov   wc.hIcon,eax
  mov   wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov   wc.hCursor,eax
  invoke RegisterClassEx, addr wc
  invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName, \
              ADDR AppName, WS_OVERLAPPEDWINDOW,\
              CW_USEDEFAULT, CW_USEDEFAULT,\
              300,200,NULL,NULL, hInst,NULL
  mov   hwnd,eax
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
  .WHILE TRUE
    invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage, ADDR msg
  .ENDW
  mov    eax,msg.wParam
  ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
  .IF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_CREATE
    invoke CreateWindowEx,WS_EX_CLIENTEDGE, ADDR EditClassName,NULL,\
              WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT or\
              ES_AUTOHSCROLL,\
              50,35,200,25,hWnd,8,hInstance,NULL
    mov  hwndEdit,eax
    invoke SetFocus, hwndEdit
    invoke CreateWindowEx,NULL, ADDR ButtonClassName,ADDR ButtonText,\
              WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
              75,70,140,25,hWnd,ButtonID,hInstance,NULL
    mov  hwndButton,eax
  .ELSEIF uMsg==WM_COMMAND
    mov eax,wParam
    .IF lParam==0
      .IF ax==IDM_HELLO
        invoke SetWindowText,hwndEdit,ADDR TestString
      .ELSEIF ax==IDM_CLEAR
        invoke SetWindowText,hwndEdit,NULL
      .ELSEIF  ax==IDM_GETTEXT
        invoke GetWindowText,hwndEdit,ADDR buffer,512
        invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
      .ELSE
        invoke DestroyWindow,hWnd
      .ENDIF
```

```
    .ELSE
      .IF ax==ButtonID
        shr eax,16
        .IF ax==BN_CLICKED
          invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
        .ENDIF
      .ENDIF
    .ENDIF
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax,eax
  ret
WndProc endp
end start
```

Análisis:

```
  .ELSEIF uMsg==WM_CREATE
    invoke CreateWindowEx,WS_EX_CLIENTEDGE, \
            ADDR EditClassName,NULL,\
            WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT\
            or ES_AUTOHSCROLL,\
            50,35,200,25,hWnd,EditID,hInstance,NULL
    mov  hwndEdit,eax
    invoke SetFocus, hwndEdit
    invoke CreateWindowEx,NULL, ADDR ButtonClassName,\
            ADDR ButtonText,\
            WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
            75,70,140,25,hWnd,ButtonID,hInstance,NULL
    mov  hwndButton,eax
```

Creamos los controles cuando procesamos el mensaje WM_CREATE. Llamamos a CreateWindowEx con el estilo de ventana extra, WS_EX_CLIENTEDGE, el cual hace que la ventana cliente se vea con un borde sombreado. El nombre de cada control es un nombre predefinido, edit para el control de edición, button para el control botón. Luego especificamos los estilos de ventanas hijas. Cada control tiene estilos extras además de los estilos de ventana normal. Por ejemplo, los estilos de botón llevan el prefijo BS_ para button style, los estilos del control de edición llevan ES_ para edit style. Tienes que revisar estos estilos en la referencia de la API de Win32. Pones un ID de control en lugar del manejador del menú. Esto no causa problemas ya que el control de una ventana hija no puede tener menú.
Después de crear cada control guardamos su manejador en una variable para su futuro uso.
Se llama a SetFocus para dar foco de entrada a la caja de edición de manera que el usuario pueda teclear texto dentro de ella inmediatamente.
Todo control de ventana hija envía una notificación a su ventana padre con WM_COMMAND.

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

Recuerda que también un menu envía mensajes WM_COMMAND para notificar a su ventana sobre su estado. ¿Cómo puedes diferenciar si los mensajes WM_COMMAND son originados desde un menú o desde un control? He aquí la respuesta:

	Palabra baja de wParam	Palabra alta de wParam	lParam
Menu	Menu ID	0	0
Control	Control ID	Código de notificación	Manejador de ventana hija

Como puedes ver, hay que chequear lParam. Si es cero, el mensaje WM_COMMAND actual es de un menú. No puedes usar wParam para diferenciar entre un menú y un control porque el ID del menú y el ID del control pueden ser idénticos y el código de notificación puede ser cero.

```
.IF ax==IDM_HELLO
        invoke SetWindowText,hwndEdit,ADDR TestString
    .ELSEIF ax==IDM_CLEAR
      invoke SetWindowText,hwndEdit,NULL
    .ELSEIF  ax==IDM_GETTEXT
      invoke GetWindowText,hwndEdit,ADDR buffer,512
      invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
```

Puedes poner una cadena de texto dentro de una caja de edición llamando a SetWindowText. Limpias el contenido de la caja de edición llamando a SetWindowText con NULL. SetWindowText es una función de la API de propósito general. Puedes usar SetWindowText para cambiar el encabezamiento o título de una ventana o el texto sobre un botón.
Para obtener el texto en una caja de edición, usas GetWindowText.

```
.IF ax==ButtonID
    shr eax,16
    .IF ax==BN_CLICKED
      invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
    .ENDIF
  .ENDIF
```

El fragmento de código de arriba tiene que ver con la condición de si el usuario presiona el botón. Primero, chequea la palabra baja de wParam para ver si el ID del control coincide con el del botón. Si es así, chequea la palabra alta de wParam para ver si es el código de notificación BN_CLICKED que se envía cuando el botón es pulsado.
La parte interesante es después que el código de notificación es BN_CLICKED. Queremos obtener el texto de la caja de edición y desplegarlo en la caja de edición. Podemos duplicar el código en la sección IDM_GETTEXT de arriba pero no tiene sentido. Si de alguna manera podemos enviar un mensaje WM_COMMAND con el valor IDM_GETTEXT en la palabra baja de wParam a nuestro procedimiento de ventana, podemos evitar duplicación de código y simplificar nuestro programa. La función SendMessage es la respuesta. Esta función envía cualquier mensaje a cualquier ventana con cualquiera wParam y lParam que decidamos. Así que en vez de duplicar código, llamamos a SendMessage con el manejador de ventana padre, WM_COMMAND,

IDM_GETTEXT, y 0. Esto tiene un efecto idéntico que seleccionar el elemento Get Text de nuestro menú. El procedimiento de ventana no percibirá ninguna diferencia entre los dos. Deberías usar estas técnicas en la medida de lo posible para que tu código sea más organizado. Por último, no olvides poner la función TranslateMessage en el bucle de mensajes, puesto que, como debes teclear algún texto en la caja de edición, tu programa debe traducir la entrada cruda del teclado a texto que pueda ser leído. Si omites esta función, no serás capaz de editar nada en la caja de edición.

2.9 Caja de Diálogo como Ventana Principal

Si llevas a la práctica los ejemplos anteriores, encontrarás que no puedes cambiar el foco de entrada de un control de ventana hija a otra con la tecla Tab. La única manera de realizar eso es haciendo click sobre el control que deseas que gane el foco de entrada. Esta situación es más bien incómoda. Otra cosa que deberías notar es que cambié el color del fondo de la ventana padre a gris en lugar de blanco, como lo había hecho en los ejemplos previos. Esto se hace así para que el color de la ventana hija pueda armonizar con el color del área cliente de la ventana padre. Hay otra manera de salvar este problema pero no es fácil. Tienes que subclasificar todos los controles de ventana hija en tu ventana padre.
La razón de la existencia de tal inconveniente es que los controles de ventana hija están originalmente diseñados para trabajar dentro de cajas de diálogo, no en una ventana normal. Los colores por defecto de los controles de ventanas hijas, como los botones, es gris porque el área cliente de la caja de diálogo normalmente es gris para que armonicen entre sí sin ninguna intervención por parte del programador.
Antes de entrar en detalles, deberíamos saber qué es una caja de diálogo. Una caja de diálogo no es más que una ventana normal diseñada para trabajar con controles de ventanas hijas. Windows también proporciona un administrador interno de cajas de diálogo, responsable por gran parte de la lógica del teclado tal como desplazamiento del foco de entrada cuando el usuario presiona Tab, presionar el botón por defecto si la tecla enter es presionada, etc. Así los programadores pueden ocuparse de tareas de más alto nivel. Las cajas de diálogo son usadas primero como dispositivos de entrada o salida. Como tal, una caja de diálogo puede ser considerada como una caja negra de entrada o salida lo que significa que no tienes que saber cómo funciona internamente una caja de diálogo para usarla, sólo tienes que saber cómo interactuar con ella. Es un principio de la programación orientada a objetos llamado encapsulación u ocultamiento de la información. Si la caja negra es perfectamente diseñada, el usuario puede emplearla sin tener conocimiento de cómo funciona. Lo único es que la caja negra debe ser perfecta, algo difícil de alcanzar en el mundo real. La API de Win32 API también ha sido diseñada como una caja negra.
Las cajas de diálogo han sido diseñadas para reducir la carga de trabajo del programador. Normalmente si tienes que poner controles de ventanas hijas sobre una ventana normal, tienes que subclasificarlas y escribir tú mismo la lógica del teclado. Pero si quieres ponerlas en una caja de diálogo, Windows manejará la

lógica por ti. Sólo tienes que saber cómo obtener la entrada del usuario de la caja de diálogo o como enviar órdenes a ella.
Como el menú, una caja de diálogo se define como un recurso. Escribes un plantilla describiendo las características de la caja de diálogo y sus controles y luego compilas el guión de recursos con un compilador de recursos.
Todos los recursos se encuentran en el mismo archivo de guión de recursos. Puedes emplear cualquier editor de texto para escribir un guión de recursos, pero no lo recomiendo. Deberías usar un editor de recursos para hacer la tarea visualmente ya que arreglar la disposición de los controles en la caja de diálogo es una tarea dura de hacer manualmente. Hay disponibles algunos excelentes editores de recursos. Muchos de las grandes suites de compiladores incluyen sus propios editores de recursos. Puedes usar cualquiera para crear un guión de recursos para tu programa y luego cortar las líneas irrelevantes.

Hay dos tipos principales de cajas de diálogo: modal y no modal, una caja de diálogo no modal te deja cambiar de foco hacia otra ventana. Un ejemplo es el diálogo Find de Word. Hay dos subtipos de caja de diálogo modal: modal de aplicación y modal de sistema. Una caja de diálogo modal de aplicación no permite cambiar el foco a otra ventana en la misma aplicación si no cambiar el foco de entrada a la ventana de otra aplicación, una caja de diálogo modal de sistema no te permite cambiar de foco hacia otra ventana hasta que respondas a la primera.
Una caja de diálogo no modal se crea llamando a la función de la API CreateDialogParam. Una caja de diálogo modal se crea llamando a DialogBoxParam. La única diferencia entre una caja de diálogo de no modal y una modal de sistema es el estilo DS_SYSMODAL. Si quieres incluir el estilo DS_SYSMODAL en una plantilla de caja de diálogo, esa caja de diálogo será modal de sistema.
Puedes comunicarte con cualquier control de ventana hija sobre una caja de diálogo usando la función SendDlgItemMessage. Su sintaxis es:

```
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 quieres obtener el texto de un control de edición, puedes hacer esto:

```
call SendDlgItemMessage, hDlg, ID_EDITBOX, WM_GETTEXT, 256, ADDR text_buffer
```

Con el fin de saber qué mensaje enviar, deberías consultar la referencia de la API de Win32. Windows también provee algunas funciones específicas de la API para controles que permiten obtener y poner datos en los controles rápidamente, por ejemplo, GetDlgItemText, CheckDlgButton etc. Estas funciones específicas para controles son suministradas para conveniencia de los programadores de manera que él no tenga que revisar el significado de wParam y lParam para cada mensaje. Normalmente, deberías usar llamadas a las funciones específicas de la API para

controles cada vez que sean disponibles ya que ellas facilitan el mantenimiento del código fuente. Recurre a SendDlgItemMessage sólo si no hay disponible llamadas a funciones específicas de la API.
El manejador de Windows de cajas de diálogos envía varios mensajes a una función callback particular llamada procedimiento de caja de diálogo que tiene el siguiente formato:

```
DlgProc  proto hDlg:DWORD ,\
               iMsg:DWORD ,\
               wParam:DWORD ,\
               lParam:DWORD
```

El procedimiento de diálogo es similar al procedimiento de ventana excepto por el tipo de valor de retorno, que es TRUE o FALSE en vez de LRESULT. El administrador interno de la caja de diálogo dentro de Windows es el verdadero procedimiento de ventana para la caja de diálogo. Llama a nuestra caja de diálogo con algunos mensajes que recibió. Así que la regla general es que: si nuestro procedimiento de diálogo procesa un mensaje, debe regresar TRUE en eax y si no procesa el mensaje, debe regresar FALSE en eax. Un procedimiento de caja de diálogo no pasa los mensajes no procesados a la llamada DefWindowProc ya que no es realmente un procedimiento de ventana.
Hay dos usos distintos de una caja de diálogo. Puedes usarlas como ventanas principal de tu aplicación o usarla como un dispositivo de entrada. Examinaremos la primera. Usar una caja de diálogo como una ventana principal puede ser interpretado en dos sentidos.
Puedes usar una plantilla de caja de diálogo como la plantilla de clase que registras al llamar a RegisterClassEx. En este caso, la caja de diálogo se comporta como una ventana normal: recibe mensajes del procedimiento de ventana referido por el miembro lpfnWndProc de la clase de ventana class, no a través de un procedimiento de caja de diálogo. El beneficio de esto es que no tienes que crear por ti mismo controles de ventana hija, Windows los crea por ti cuando se crea la caja de diálogo. También Windows maneja la lógica del teclado para ti, por ejemplo se encarga de la orden Tab, etc. Además puedes especificar el cursor y el icono de tu ventana en la estructura de la clase de ventana. Tu programa crea la caja de diálogo sin ninguna ventana padre. Esta aproximación al problema hace innecesario el uso de un bucle de mensajes ya que los mensajes son enviados directamente al procedimiento de ventana de la caja de diálogo.

Ejemplos:

dialog.asm

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName db "DLGCLASS",0
```

```
MenuName db "MyMenu",0
DlgName db "MyDialog",0
AppName db "Our First Dialog Box",0
TestString db "Wow! I'm in an edit box now",0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer 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
.code
start:
  Invoke GetModuleHandle, NULL
  mov   hInstance,eax
  Invoke GetCommandLine
  Invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  Invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hDlg:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,DLGWINDOWEXTRA
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_BTNFACE+1
  mov  wc.lpszMenuName,OFFSET MenuName
  mov  wc.lpszClassName,OFFSET ClassName
  Invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  Invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  Invoke RegisterClassEx, addr wc
  Invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL
  mov  hDlg,eax
  Invoke ShowWindow, hDlg,SW_SHOWNORMAL
  Invoke UpdateWindow, hDlg
  Invoke GetDlgItem,hDlg,IDC_EDIT
  Invoke SetFocus,eax
  .WHILE TRUE
    Invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
    Invoke IsDialogMessage, hDlg, ADDR msg
    .IF eax ==FALSE
      Invoke TranslateMessage, ADDR msg
      Invoke DispatchMessage, ADDR msg
    .ENDIF
  .ENDW
  mov    eax,msg.wParam
  ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
  .IF uMsg==WM_DESTROY
    Invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_COMMAND
    mov eax,wParam
    .IF lParam==0
      .IF ax==IDM_GETTEXT
        Invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR buffer,512
        Invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
      .ELSEIF ax==IDM_CLEAR
```

```
            Invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
          .ELSE
            Invoke DestroyWindow,hWnd
          .ENDIF
        .ELSE
          mov edx,wParam
          shr edx,16
          .IF dx==BN_CLICKED
            .IF ax==IDC_BUTTON
              Invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR TestString
            .ELSEIF ax==IDC_EXIT
              Invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
            .ENDIF
          .ENDIF
        .ENDIF
      .ELSE
        Invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
      .ENDIF
      xor   eax,eax
      ret
WndProc endp
end start
```

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
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our First Dialog Box"
CLASS "DLGCLASS"
BEGIN
  EDITTEXT      IDC_EDIT,  15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
  DEFPUSHBUTTON  "Say Hello", IDC_BUTTON,   141,10,52,13
  PUSHBUTTON     "E&xit", IDC_EXIT,  141,26,52,13, WS_GROUP
END

MyMenu  MENU
BEGIN
  POPUP "Test Controls"
  BEGIN
    MENUITEM "Get Text", IDM_GETTEXT
    MENUITEM "Clear Text", IDM_CLEAR
    MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
    MENUITEM "E&xit", IDM_EXIT
  END
END
```

Análisis:

Vamos a analizar el primer ejemplo.
Este ejemplo muestra como registrar una plantilla de diálogo como una clase de ventana y crear una ventana a partir de esa clase. Simplifica tu programa ya que no tienes que crear tú mismo los controles de ventana hija. Vamos a analizar primero la plantilla de diálogo.

```
MyDialog DIALOG 10, 10, 205, 60
```

Declarar el nombre del diálogo, en este caso, MyDialog seguido por la palabra clave DIALOG. Los siguientes cuatro números son: x, y, ancho, y alto de la caja de diálogo en unidades de caja de diálogo no de píxeles.

```
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
```

Declarar los estilos de la caja de diálogo.

```
CAPTION "Our First Dialog Box"
```

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

```
CLASS "DLGCLASS"
```

Esta línea es crucial. Es esta palabra clave, CLASS, lo que nos permite usar la caja de diálogo como una clase de ventana. Después de la palabra clave está el nombre de la clase

```
BEGIN
  EDITTEXT      IDC_EDIT,  15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
  DEFPUSHBUTTON  "Say Hello", IDC_BUTTON,   141,10,52,13
  PUSHBUTTON     "E&xit", IDC_EXIT,  141,26,52,13
END
```

El bloque de arriba define los controles de ventana hija en la caja de diálogo. Están definidos entre las palabras claves BEGIN y END. Generalmente la sintaxis es la siguiente:

```
control-type  "text"   ,controlID, x, y, width, height [,styles]
```

Los tipos de controles son constantes del compilador de recursos así que tienes que consultar el manual.
Ahora vamos a ensamblar el código fuente. La parte interesante es la estructura de la clase de ventana:

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

Normalmente, este miembro se deja NULL, pero si queremos registrar una plantilla de caja de diálogo como una clase de ventana, debemos poner en este miembro el valor DLGWINDOWEXTRA. El nombre de la clase debe ser idéntico al que sigue a la palabra clave CLASS en la plantilla de la caja de diálogo. Los miembros restantes se inicializan como es usual. Después de que llenas la estructura de la clase de ventana, la registras con RegisterClassEx. Esta es la misma rutina que tienes que hacer con el fin de registrar una clase de ventana normal.

```
Invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL
```

Después de registrar la clase de ventana, creamos nuestra caja de diálogo. En este ejemplo, lo creo como una caja de diálogo modal con la función

CreateDialogParam. Esta función toma 5 parámetros, pero sólo tienes que llenar los primeros dos: el manejador de instancia y el puntero al nombre de la plantilla de la caja de diálogo. el segundo parámetro no es un puntero al nombre de la clase.
En este punto, la caja de diálogo y sus controles de ventana hija son creados por Windows. Tu procedimiento de ventana recibirá el mensaje WM_CREATE como es usual.

```
invoke GetDlgItem,hDlg,IDC_EDIT
invoke SetFocus,eax
```

Después de que es creada la caja de diálogo, quiero poner el foco de entrada a la caja de edición. Si pongo estas instrucciones en la sección WM_CREATE, la llamada a GetDlgItem fallará ya que en ese momento, ya que en ese instante todavía no se han creado los controles de ventana hija. La única manera de hacer esto es llamarlo después de que la caja de diálogo y todos los controles de ventana hija son creados. Así que pongo estas dos líneas después de la llamada a UpdateWindow. La función GetDlgItem obtiene el ID del control y regresa el manejador del control de ventana asociado. Así es como puedes obtener el manejador de un control de ventana hija si tienes su ID.

```
invoke IsDialogMessage, hDlg, ADDR msg
   .IF eax ==FALSE
      invoke TranslateMessage, ADDR msg
      invoke DispatchMessage, ADDR msg
   .ENDIF
```

El programa introduce el bucle de mensajes y antes de que traduzcamos y despachemos mensajes, llamamos a la función IsDialogMessage para permitir que el administrador de la caja de diálogo maneje el teclado lógico de nuestra caja de diálogo. Si la función regresa TRUE, significa que el mensaje es enviado a la caja de diálogo y es procesado por el administrador de la caja de diálogo. **Nota**: Cuando el procedimiento de ventana quiere obtener el texto del control de edición, llama a la función GetDlgItemText en vez de GetWindowText. GetDlgItemText acepta un ID de control en vez de un manejador de ventana. Eso facilita la llamada en el caso de que utilices una caja de diálogo.
Ahora vamos a la segunda aproximación de cómo usar una caja de diálogo como ventana principal. En el siguiente ejemplo, crearé una aplicación de caja de diálogo modal. No encontrarás un bucle de mensaje ni un procedimiento de ventana porque no son necesarios.

dialog.asm (parte 2)

```
.386
.model flat,stdcall
option casemap:none
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
.data
DlgName db "MyDialog",0
```

```
AppName db "Our Second Dialog Box",0
TestString db "Wow! I'm in an edit box now",0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer 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

.code
start:
  invoke GetModuleHandle, NULL
  mov    hInstance,eax
  invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
  invoke ExitProcess,eax
DlgProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
  .IF uMsg==WM_INITDIALOG
    invoke GetDlgItem, hWnd,IDC_EDIT
    invoke SetFocus,eax
  .ELSEIF uMsg==WM_CLOSE
    invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
  .ELSEIF uMsg==WM_COMMAND
    mov eax,wParam
    .IF lParam==0
      .IF ax==IDM_GETTEXT
        invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR buffer,512
        invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
      .ELSEIF ax==IDM_CLEAR
        invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
      .ELSEIF ax==IDM_EXIT
        invoke EndDialog, hWnd,NULL
      .ENDIF
    .ELSE
      mov edx,wParam
      shr edx,16
      .If dx==BN_CLICKED
        .IF ax==IDC_BUTTON
          invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR TestString
        .ELSEIF ax==IDC_EXIT
          invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
        .ENDIF
      .ENDIF
    .ENDIF
  .ELSE
    mov eax,FALSE
    ret
  .ENDIF
  mov eax,TRUE
  ret
DlgProc endp
end start
```

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
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our Second Dialog Box"
MENU IDR_MENU1
BEGIN
  EDITTEXT      IDC_EDIT,  15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
  DEFPUSHBUTTON  "Say Hello", IDC_BUTTON,   141,10,52,13
  PUSHBUTTON     "E&xit", IDC_EXIT,  141,26,52,13
END
IDR_MENU1  MENU
BEGIN
  POPUP "Test Controls"
  BEGIN
    MENUITEM "Get Text", IDM_GETTEXT
    MENUITEM "Clear Text", IDM_CLEAR
    MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
    MENUITEM "E&xit", IDM_EXIT
  END
END
```

A continuación el análisis:

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

Declaramos el prototipo de función para DlgProc de manera que podamos referirnos a él con el operador addr en la línea de abajo:

```
Invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
```

La línea de arriba llama a la función DialogBoxParam que toma 5 parámetros: el manejador de instancia, el nombre de la plantilla de la caja de diálogo, el manejador de la ventana padre, la dirección del procedimiento de de ventana, y los datos específicos del diálogo. DialogBoxParam crea una caja de diálogo modal. No regresará hasta que la caja de diálogo sea destruida.

```
.IF uMsg==WM_INITDIALOG
  Invoke GetDlgItem, hWnd,IDC_EDIT
  Invoke SetFocus,eax
.ELSEIF uMsg==WM_CLOSE
  Invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
```

El procedimiento de la caja de diálogo se observa como un procedimiento de ventana excepto en que no recibe el mensaje WM_CREATE. El primer mensaje que recibe es WM_INITDIALOG. Normalmente puedes poner aquí el código de inicialización. Debes regresar el valor TRUE en eax si procesas el mensaje.
El administrador interno de la caja de diálogo no envía a nuestro procedimiento de diálogo el mensaje WM_DESTROY por defecto cuando WM_CLOSE es enviado a nuestra caja de diálogo. Así que si queremos reaccionar cuando el usuario presiona el botón cerrar en nuestra caja de diálogo, debemos procesar el mensaje WM_CLOSE. En nuestro ejemplo, enviamos el mensaje WM_COMMAND con el valor IDM_EXIT en wParam. Esto tiene el mismo efecto que cuando el usuario selecciona el elemento del menú Exit. EndDialog es llamado en respuesta a IDM_EXIT.
El procesamiento de WM_COMMAND se mantiene igual.

Cuando quieres destruir la caja de diálogo, la única manera es llamar a la función EndDialog. No emplees DestroyWindow; EndDialog, no destruye la caja de diálogo de inmediato. Sólo pone una bandera para ser revisada por el administrador interno de la caja de diálogo y continúa para ejecutar la siguiente instrucción.
Ahora vamos a revisar el archivo de recursos. El cambio notable es que en vez de usar una cadena de texto como nombre de menú usamos un valor, IDR_MENU1. Esto es necesario si quieres agregar un menú a la caja de diálogo creada con DialogBoxParam. En la plantilla de caja de diálogo tienes que agregar la palabra clave MENU seguida por el ID del recurso.
Una diferencia que puedes observar entre los dos ejemplos es la carencia de un icono en el último ejemplo. Sin embargo, puedes poner el icono enviando el mensaje WM_SETICON a la caja de diálogo durante WM_INITDIALOG.

Hay muy poco que decir sobre como usar las cajas de diálogo como entrada-salida de nuestro programa. Tu programa crea la página principal normalmente y cuando quieres mostrar las cajas de diálogo, llamas a CreateDialogParam o DialogBoxParam. Con la llamada a DialogBoxParam, no tendrás que hacer nada mas, sólo procesar los mensajes en el procedimiento de las cajas de diálogo. Con CreateDialogParam, tendrás que insertar la llamada a IsDialogMessage en el bucle de mensajes para dejar a las cajas de diálogo el control sobre la navegación del teclado en tus cajas de diálogo.
Comencemos con las cajas de diálogo comunes. Windows tiene preparadas unas cajas de diálogo predefinidas que pueden ser usadas por tus aplicaciones.
Estas cajas de diálogo existen para proveer un interfaz estándar de usuario. Consisten en cajas de diálogo de archivo, impresión, color, fuente, y búsqueda. Deberías usarlas lo máximo posible. Las cajas de diálogo residen en comdlg32.dll. Para usarlas, tendrás que enlazar el archivo comdlg32.lib. Creas estas cajas de diálogo llamando a la función apropiada en la librería de las cajas de diálogo. Para el archivo de diálogo Abrir, se emplea la función GetOpenFileName, para la caja de diálogo Guardar GetSaveFileName, para dibujar un diálogo es PrintDlg y ya está. Cada una de estas funciones toma como parámetro un puntero a la estructura. Deberás leerlo en la referencia de la API de Win32.
Debajo está la el prototipo de la función GetOpenFileName:

```
GetOpenFileName proto lpofn:DWORD
```

Puedes ver que sólo recibe un parámetro, un puntero a la estructura OPENFILENAME. El valor devuelto es TRUE que significa que el usuario a seleccionado un archivo para abrir, de otra manera devolverá FALSE. Lo siguiente que veremos será la estructura OPENFILENAME.

```
OPENFILENAME STRUCT
lStructSize DWORD ?
hwndOwner HWND ?
hInstance HINSTANCE ?
lpstrFilter LPCSTR ?
lpstrCustomFilter LPSTR ?
nMaxCustFilter DWORD ?
nFilterIndex DWORD ?
lpstrFile LPSTR ?
nMaxFile DWORD ?
lpstrFileTitle LPSTR ?
```

```
nMaxFileTitle DWORD ?
lpstrInitialDir LPCSTR ?
lpstrTitle LPCSTR ?
Flags DWORD ?
nFileOffset WORD ?
nFileExtension WORD ?
lpstrDefExt LPCSTR ?
lCustData LPARAM ?
lpfnHook DWORD ?
lpTemplateName LPCSTR ?
OPENFILENAME ENDS
```

La siguiente tabla muestra el significado de los miembros mas utilizados de la estructura.

lStructSize	El tamaño de la estructura OPENFILENAME, en bytes
hwndOwner	El manejador de la caja de diálogo open file.
hInstance	Manejador de la instancia de la aplicación que crea la caja de diálogo open file.
lpstrFilter	Las cadenas de filtro en formato de pares de cadenas terminadas en null (0). La primera entre las dos es la de descripción. La segunda cadena es el patrón de filtro, por ejemplo: FilterString db "All Files (*.*)",0, "*.*",0 db "Text Files (*.txt)",0,"*.txt",0,0 Date cuenta que sólo el patrón en la segunda cadena de este par es usado actualmente por Windows para filtrar los archivos. También date cuenta de que tienes que poner un 0 extra al final de la cadena de filtro para advertir el final de ésta.
nFilterIndex	Especifica que par de cadenas de filtro que serán usadas inicialmente cuando la caja de diálogo, open file es mostrada por primera vez. El índice es de base 1, que es el primer par 1, el segundo par es 2 y así el resto. En el ejemplo de arriba, si especificamos nFilterIndex como 2, será usado el segundo patrón, "*.txt".
lpstrFile	Puntero al buffer que contiene el nombre del archivo usado para inicializar el control de edición de nombre de archivo en la caja de diálogo. El buffer será como mínimo de 260 bytes de largo. Después de que el usuario seleccione el archivo a abrir, el nombre de archivo con toda la dirección, es almacenado en este buffer. Puedes extraer la información de aquí mas tarde.
nMaxFile	El tamaño del buffer lpstrFile.
lpstrTitle	Puntero al encabezamiento o título de la caja de diálogo open file
Flags	Determina el estilo y caracteristicas de la caja de diálogo.
nFileOffset	Después de que el usuario haya seleccionado el archivo a abrir, este miembro contiene el índice al primer carácter del nombre de archivo actual. Por ejemplo, si el nombre completo con el directorio es "c:\windows\system\lz32.dll", este miembro contendrá el valor 18.
nFileExtension	después de que el usuario seleccione el archivo a abrir, este miembro contiene el índice al primer carácter de la extensión del archivo

Ejemplo:

El siguiente programa muestra una caja de diálogo de abrir archivo cuando el usuario seleccione File-> Open del menu. Cuando el usuario seleccione un archivo en la caja de diálogo, el programa muestra una caja de mensaje mostrando el nombre completo, nombre de archivo, y extensión del archivo seleccionado.

```
.386
.model flat,stdcall
option casemap:none
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 equ 1
IDM_EXIT equ 2
MAXSIZE equ 260
OUTPUTSIZE equ 512
.data
ClassName db "SimpleWinClass",0
AppName  db "Our Main Window",0
MenuName db "FirstMenu",0
ofn   OPENFILENAME <>
FilterString db "All Files",0,"*.*",0
        db "Text Files",0,"*.txt",0,0
buffer db MAXSIZE dup(0)
OurTitle db "-=Our First Open File Dialog Box=-: Choose the file to open",0
FullPathName db "The Full Filename with Path is: ",0
FullName  db "The Filename is: ",0
ExtensionName db "The Extension is: ",0
OutputString db OUTPUTSIZE dup(0)
CrLf db 0Dh,0Ah,0
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,OFFSET MenuName
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
        WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
        CW_USEDEFAULT,300,200,NULL,NULL,\
        hInst,NULL
   mov   hwnd,eax
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
   invoke UpdateWindow, hwnd
   .WHILE TRUE
      invoke GetMessage, ADDR msg,NULL,0,0
      .BREAK .IF (!eax)
      invoke TranslateMessage, ADDR msg
      invoke DispatchMessage, ADDR msg
   .ENDW
   mov    eax,msg.wParam
   ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   .IF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
```

```
    .If ax==IDM_OPEN
      mov ofn.lStructSize,SIZEOF ofn
      push hWnd
      pop  ofn.hwndOwner
      push hInstance
      pop  ofn.hInstance
      mov  ofn.lpstrFilter, OFFSET FilterString
      mov  ofn.lpstrFile, OFFSET buffer
      mov  ofn.nMaxFile,MAXSIZE
      mov  ofn.Flags, OFN_FILEMUSTEXIST or \
        OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
        OFN_EXPLORER or OFN_HIDEREADONLY
      mov  ofn.lpstrTitle, OFFSET OurTitle
      Invoke GetOpenFileName, ADDR ofn
      .if eax==TRUE
        invoke lstrcat,offset OutputString,OFFSET FullPathName
        invoke lstrcat,offset OutputString,ofn.lpstrFile
        invoke lstrcat,offset OutputString,offset CrLf
        invoke lstrcat,offset OutputString,offset FullName
        mov  eax,ofn.lpstrFile
        push ebx
        xor  ebx,ebx
        mov  bx,ofn.nFileOffset
        add  eax,ebx
        pop  ebx
        invoke lstrcat,offset OutputString,eax
        invoke lstrcat,offset OutputString,offset CrLf
        invoke lstrcat,offset OutputString,offset ExtensionName
        mov  eax,ofn.lpstrFile
        push ebx
        xor ebx,ebx
        mov  bx,ofn.nFileExtension
        add eax,ebx
        pop ebx
        invoke lstrcat,offset OutputString,eax
        invoke MessageBox,hWnd,OFFSET OutputString,ADDR AppName,MB_OK
        invoke RtlZeroMemory,offset OutputString,OUTPUTSIZE
      .endif
    .else
      invoke DestroyWindow, hWnd
    .endif
  .ELSE
    invoke DefWindowProc,hWnd,uMsg,wParam,lParam
    ret
  .ENDIF
  xor   eax,eax
  ret
WndProc endp
 end start
```

Análisis del código anterior tenemos:

```
      mov ofn.lStructSize,SIZEOF ofn
      push hWnd
      pop  ofn.hwndOwner
      push hInstance
      pop  ofn.hInstance
```

Rellenamos en la rutina los miembros de la estructura ofn.

```
      mov  ofn.lpstrFilter, OFFSET FilterString
```

Este FilterString es el filtro para el nombre de archivo que especificamos como sigue:

```
FilterString db "All Files",0,"*.*",0
             db "Text Files",0,"*.txt",0,0
```

Date cuenta que las cuatro cadenas terminan en 0. La primera cadena es la descripción de la siguiente cadena. El actual patrón es la cadena par, en este caso, "*.*" y "*.txt". Actualmente podemos especificar aquí cualquier patrón que queramos. Debemos poner un cero extra después de la última cadena de patrón para denotar el final de la cadena de filtro. No olvides esto si no tu caja de diálogo funcionará de forma extraña.

```
mov ofn.lpstrFile, OFFSET buffer
mov ofn.nMaxFile,MAXSIZE
```

Especificamos dónde la caja de diálogo pondrá el nombre del archivo que ha seleccionado el usuario. Date cuenta que tendremos que especificar su tamaño en el miembro nMaxFile. Podemos extraer más tarde el nombre del archivo de este buffer.

```
mov ofn.Flags, OFN_FILEMUSTEXIST or \
  OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
  OFN_EXPLORER or OFN_HIDEREADONLY
```

Flags especifica las características de la caja de diálogo.
Las banderas flags OFN_FILEMUSTEXIST y OFN_PATHMUSTEXIST demandan que el nombre de archivo y dirección que el usuario pone en el control de edición. La bandera OFN_LONGNAMES dice a la caja de diálogo que muestre nombres largos de archivo. La bandera OFN_EXPLORER especifica que la apariencia de la caja de diálogo debe ser parecida a la del explorador. La bandera OFN_HIDEREADONLY oculta el cuadro de selección de solo lectura en la caja de diálogo.
Hay muchas banderas más que puedes usar. Consultando las referencias de la función API de Win32.

```
mov ofn.lpstrTitle, OFFSET OurTitle
```

Especifica el titulo o encabezado de la caja de diálogo.

```
Invoke GetOpenFileName, ADDR ofn
```

Llamada a la función GetOpenFileName. Pasando el puntero a la estructura ofn como parámetro.
En este momento, la caja de diálogo, abrir archivo, es mostrada en la pantalla, la función no volverá hasta que el usuario seleccione un archivo para abrir o presione el botón de cancelar o cierre la caja de diálogo. Devolverá el valor TRUE en eax si el usuario selecciona un archivo para abrir. Si no devolverá FALSE en cualquier otro caso.

```
.If eax==TRUE
Invoke lstrcat,offset OutputString,OFFSET FullPathName
Invoke lstrcat,offset OutputString,ofn.lpstrFile
Invoke lstrcat,offset OutputString,offset CrLf
Invoke lstrcat,offset OutputString,offset FullName
```

En el caso que el usuario seleccione un archivo para abrir, preparamos la cadena de salida que se mostrará en la caja de mensajes. Ubicamos un bloque de memoria en la variable OutputString y entonces usamos la función de la API, lstrcat, para entrelazar las cadenas siempre. Para poner las cadenas en varias líneas, debemos separar cada línea con el par de caracteres que alimentan el retorno de carro (13d o 0Dh) y el avance de línea (10d o 0Ah).

```
mov eax,ofn.lpstrFile
push ebx
xor ebx,ebx
mov bx,ofn.nFileOffset
add eax,ebx
pop ebx
Invoke lstrcat,offset OutputString,eax
```

Las líneas de arriba requieren una explicación. nFileOffset contiene el índice dentro de ofn.lpstrFile. Pero no puedes añadirlo directamente porque nFileOffset es una variable de tamaño WORD y lpstrFile es de tamaño DWORD. Así que tendré que poner el valor de nFileOffset en la palabra baja [low word] de ebx y sumárselo al valor de lpstrFile.

```
Invoke MessageBox,hWnd,OFFSET OutputString,ADDR AppName,MB_OK
```

Mostramos la cadena en la caja de mensajes.

```
Invoke RtlZeroMemory,offset OutputString,OUTPUTSIZE
```

Debemos limpiar el OutputString antes de poder meterle cualquier otra cadena. Así que usamos la función RtlZeroMemory para hacer este trabajo.

2.10 Manejo de Memoria y E/S de Archivos

El manejo de la memoria bajo Win32 desde el punto de vista de las aplicaciones es un poco simple y directo. Cada proceso usa un espacio de 4 GB de direcciones de memoria. El modelo de memoria usado se llama modelo de memoria plana. En este modelo, todos los segmentos de registro o selectores, direccionan a la misma localidad de memoria y el desplazamiento es de 32-bit. También las aplicaciones pueden acceder a la memoria en cualquier punto en su espacio de direcciones sin necesidad de cambiar el valor de los selectores. Esto simplifica mucho el manejo de la memoria. No hay más puntos cercas o lejos.
Bajo Win16, hay dos categorías principales de funciones de la API de memoria: Global y Local, las de tipo Global tienen que ver con la memoria situada en diferentes segmentos, por eso hay funciones para memoria lejana. Las funciones de la API de tipo Local tienen que ver con un montículo de memoria local del proceso así que son las funciones de memoria cercana. Bajo Win32, estos dos tipos son uno y el mismo tipo. Tendrás el mismo resultado si llamas a GlobalAlloc o LocalAlloc.
Los pasos para ubicar y usar la memoria son los siguientes:

- ✓ Ubicar el bloque de memoria llamando a GlobalAlloc; Esta función devuelve un manejador al bloque de memoria pedido.
- ✓ Bloquear el bloque de memoria llamando a GlobalLock; Esta función acepta un manejador al bloque de memoria y devuelve un puntero al bloque de memoria.
- ✓ Puedes usar el puntero para leer o escribir en la memoria.
- ✓ Desbloquear el bloque de memoria llamando a GlobalUnlock; Esta función invalida el puntero al bloque de memoria.
- ✓ Liberar el bloque de memoria llamando a GlobalFree; Esta función acepta un manejador al bloque de memoria.

Puedes sustituir Global por Local en LocalAlloc, LocalLock, etc.
El método anterior puede simplificarse radicalmente usando el flag GMEM_FIXED en la llamada a GlobalAlloc. Si usas esta bandera, El valor de retorno de Global/LocalAlloc será el puntero al bloque de memoria reservado, no el manejador. No tienes que llamar a Global/LocalLock y puedes pasar el puntero a Global/LocalFree sin llamar primero a Global/LocalUnlock.

La E/S de archivos bajo Win32 tiene una semblanza más notable que la de bajo DOS. Los pasos a seguir son los mismos. Sólo tienes que cambiar las interrupciones por llamadas a la API y ya está. Los pasos requeridos son los siguientes:

- ✓ Abrir o Crear el archivo llamando a la función CreateFile; esta función es muy versátil, añadiendo a los archivos, puede abrir puertos de comunicación, tuberías, dispositivos de discos. Cuando es correcto, devuelve un manejador al archivo o dispositivo. Entonces puedes usar este manejador para llevar a cabo operaciones en el archivo o dispositivo. Mueve el puntero a la posición deseada llamando a SetFilePointer.

- ✓ Realiza la operación de lectura o escritura llamando a ReadFile o WriteFile; estas funciones transfieren datos desde un bloque de memoria hacia o desde un archivo. Así que tendrás que reservar un bloque de memoria suficientemente grande para alojar los datos.

- ✓ Cierra el archivo llamando a CloseHandle; esta función acepta el manejador de archivo.

Ejemplo:

El siguiente programa muestra una caja de diálogo de abrir archivo. Deja al usuario seleccionar un archivo de texto y muestra el contenido de este archivo en un control de edición en su área cliente.
El usuario puede modificar el texto en el control de edición como desee y puede elegir guardar el contenido en un archivo.

```
.386
.model flat,stdcall
option casemap:none
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 equ 1
IDM_SAVE equ 2
IDM_EXIT equ 3
MAXSIZE equ 260
MEMSIZE equ 65535
EditID equ 1                    ; ID del control de edición
.data
ClassName db "Win32ASMEditClass",0
AppName  db "Win32 ASM Edit",0
EditClass db "edit",0
MenuName db "FirstMenu",0
ofn   OPENFILENAME <>
FilterString db "All Files",0,"*.*",0
        db "Text Files",0,"*.txt",0,0
buffer db MAXSIZE dup(0)
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwndEdit HWND ?                 ; manejador del control de edición
hFile HANDLE ?                  ; manejador de archivo
hMemory HANDLE ?                ; manejador del bloque de memoria reservada
pMemory DWORD ?                 ; puntero al bloque de memoria reservada
SizeReadWrite DWORD ?           ; numero de bytes actualmente para leer o escribir
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke GetCommandLine
  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:SDWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_WINDOW+1
  mov  wc.lpszMenuName,OFFSET MenuName
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke RegisterClassEx, addr wc
  invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
      WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
      CW_USEDEFAULT,300,200,NULL,NULL,\
      hInst,NULL
  mov  hwnd,eax
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
  .WHILE TRUE
```

```
    invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage, ADDR msg
  .ENDW
  mov    eax,msg.wParam
  ret
WinMain endp
WndProc proc uses ebx hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
  .IF uMsg==WM_CREATE
    invoke CreateWindowEx,NULL,ADDR EditClass,NULL,\
           WS_VISIBLE or WS_CHILD or ES_LEFT or ES_MULTILINE or\
           ES_AUTOHSCROLL or ES_AUTOVSCROLL,0,\
           0,0,0,hWnd,EditID,\
           hInstance,NULL
    mov hwndEdit,eax
    invoke SetFocus,hwndEdit
;=========================================================
;    Inicialización de los miembros de la estructura OPENFILENAME
;=========================================================
    mov ofn.lStructSize,SIZEOF ofn
    push hWnd
    pop  ofn.hWndOwner
    push hInstance
    pop  ofn.hInstance
    mov  ofn.lpstrFilter, OFFSET FilterString
    mov  ofn.lpstrFile, OFFSET buffer
    mov  ofn.nMaxFile,MAXSIZE
  .ELSEIF uMsg==WM_SIZE
    mov eax,lParam
    mov edx,eax
    shr edx,16
    and eax,0ffffh
    invoke MoveWindow,hwndEdit,0,0,eax,edx,TRUE
  .ELSEIF uMsg==WM_DESTROY
    invoke PostQuitMessage,NULL
  .ELSEIF uMsg==WM_COMMAND
    mov eax,wParam
    .if lParam==0
      .if ax==IDM_OPEN
        mov  ofn.Flags, OFN_FILEMUSTEXIST or \
             OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
             OFN_EXPLORER or OFN_HIDEREADONLY
        invoke GetOpenFileName, ADDR ofn
        .if eax==TRUE
          invoke CreateFile,ADDR buffer,\
             GENERIC_READ or GENERIC_WRITE ,\
             FILE_SHARE_READ or FILE_SHARE_WRITE,\
             NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
             NULL
          mov hFile,eax
          invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
          mov  hMemory,eax
          invoke GlobalLock,hMemory
          mov  pMemory,eax
          invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
          invoke SendMessage,hwndEdit,WM_SETTEXT,NULL,pMemory
          invoke CloseHandle,hFile
          invoke GlobalUnlock,pMemory
          invoke GlobalFree,hMemory
        .endif
        invoke SetFocus,hwndEdit
      .elseif ax==IDM_SAVE
        mov ofn.Flags,OFN_LONGNAMES or\
             OFN_EXPLORER or OFN_HIDEREADONLY
        invoke GetSaveFileName, ADDR ofn
          .if eax==TRUE
            invoke CreateFile,ADDR buffer,\
                   GENERIC_READ or GENERIC_WRITE ,\
                   FILE_SHARE_READ or FILE_SHARE_WRITE,\
```

```
                    NULL,CREATE_NEW,FILE_ATTRIBUTE_ARCHIVE,\
                    NULL
            mov hFile,eax
            invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
            mov  hMemory,eax
            invoke GlobalLock,hMemory
            mov  pMemory,eax
            invoke SendMessage,hwndEdit,WM_GETTEXT,MEMSIZE-1,pMemory
            invoke WriteFile,hFile,pMemory,eax,ADDR SizeReadWrite,NULL
            invoke CloseHandle,hFile
            invoke GlobalUnlock,pMemory
            invoke GlobalFree,hMemory
          .endif
          invoke SetFocus,hwndEdit
        .else
          invoke DestroyWindow, hWnd
        .endif
      .endif
    .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
.ENDIF
xor   eax,eax
ret
WndProc endp
end start
```

Análisis:

```
    invoke CreateWindowEx,NULL,ADDR EditClass,NULL,\
        WS_VISIBLE or WS_CHILD or ES_LEFT or ES_MULTILINE or\
        ES_AUTOHSCROLL or ES_AUTOVSCROLL,0,\
        0,0,hWnd,EditID,\
        hInstance,NULL
  mov hwndEdit,eax
```

En la sección WM_CREATE, creamos el control de edición. Date cuenta que los parámetros que especifican x, y, anchura y altura del control son todos ceros ya que reajustaremos el tamaño del control después para cubrir toda el área cliente de la ventana padre.
En este caso, no tenemos que llamar a ShowWindow para hacer que el control de edición aparezca en la pantalla porque hemos incluido el estilo WS_VISIBLE. Puedes usar este truco también en la ventana padre.

```
;=====================================================
;    Inicializa los miembros de la estructura OPENFILENAME
;=====================================================
    mov ofn.lStructSize,SIZEOF ofn
    push hWnd
    pop  ofn.hWndOwner
    push hInstance
    pop  ofn.hInstance
    mov  ofn.lpstrFilter, OFFSET FilterString
    mov  ofn.lpstrFile, OFFSET buffer
    mov  ofn.nMaxFile,MAXSIZE
```

Después de crear el control de edición tendremos que inicializar los miembros de ofn. Como queremos reciclar ofn en la caja de diálogo para guardar, pondremos solo los miembros comunes que son usados por ambos GetOpenFileName y GetSaveFileName.

La sección WM_CREATE es un lugar amplio para poner las inicializaciones únicas que se inicializan una sola vez.

```
.ELSEIF uMsg==WM_SIZE
 mov eax,lParam
 mov edx,eax
 shr edx,16
 and eax,0ffffh
 invoke MoveWindow,hwndEdit,0,0,eax,edx,TRUE
```

Recibimos el mensaje WM_SIZE cuando el tamaño de nuestra área cliente en la ventana principal cambia. También lo recibimos cuando la ventana es creada por primera vez. Para poder recibir el mensaje, el estilo de ventana debe incluir los estilos CS_VREDRAW y CS_HREDRAW. Usamos esta oportunidad para reajustar el tamaño de nuestro control de edición al mismo tamaño de nuestra área cliente de la ventana padre. Primero tenemos que saber la anchura y altura del área cliente de la ventana padre. Obtenemos esta información de lParam. La palabra alta de lParam contiene la altura y la palabra baja de lParam la anchura del área cliente. Entonces usamos la información para reajustar el tamaño del control de edición llamando a la función MoveWindow, que además de cambiar la posición de la ventana, permite cambiar su tamaño.

```
.if ax==IDM_OPEN
        mov  ofn.Flags, OFN_FILEMUSTEXIST or \
                OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
                OFN_EXPLORER or OFN_HIDEREADONLY
        invoke GetOpenFileName, ADDR ofn
```

Cuando el usuario selecciona el elemento de menú File/Open, rellenamos el miembro Flags de la estructura ofn y llamamos a la función GetOpenFileName para mostrar la caja de diálogo de abrir archivo.

```
          .if eax==TRUE
        invoke CreateFile,ADDR buffer,\
                GENERIC_READ or GENERIC_WRITE ,\
                FILE_SHARE_READ or FILE_SHARE_WRITE,\
                NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
                NULL
        mov hFile,eax
```

Después que el usuario ha seleccionado el archivo que desea abrir, llamamos a CreateFile para abrir el archivo. Hemos especificado que la función intentará abrir el archivo para lectura y escritura. Después de abrir el archivo, la función devuelve el manejador al archivo abierto que almacenamos en una variable global para futuros usos. Esta función es como sigue:

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

Donde:
dwDesiredAccess: especifica qué operación quieres que se haga en el archivo.
0 Abrir el archivo para pedir sus atributos. Tendrás derecho a escribir o leer los datos.
GENERIC_READ: Abrir el archivo para lectura.
GENERIC_WRITE: Abrir el archivo para escribir.
dwShareMode: especifica qué operaciones quieres reservar para que otros procesos puedan llevarlas a cabo en el archivo que va a ser abierto.
0 No compartir el archivo con otros procesos.
FILE_SHARE_READ: permitir a otros procesos leer los datos del archivo abierto
FILE_SHARE_WRITE: permitir a otros procesos escribir datos en el archivo abierto.
lpSecurityAttributes: no tiene significado bajo Windows 95.
dwCreationDistribution: especifica la acción a ejecutar por CeateFile cuando el archivo especificado en lpFileName existe o cuando no existe.
CREATE_NEW: Crear un nuevo archivo. La función falla si ya existe el archivo especificado.
CREATE_ALWAYS: Crea un nuevo archivo. La función sobrescribe el archivo si éste existe.
OPEN_EXISTING: Abre el archivo. La función falla si el archivo no existe.
OPEN_ALWAYS: Abre el archivo si existe. Si el archivo no existe, la función crea el archivo como si dwCreationDistribution fuera CREATE_NEW.
TRUNCATE_EXISTING: Abre el archivo. Una vez abierto, el archivo es truncado si su tamaño es de cero bytes. El proceso llamado debe abrir el archivo como mínimo con el acceso GENERIC_WRITE. La función falla si el archivo no existe.
dwFlagsAndAttributes especifica los atributos de archivo
FILE_ATTRIBUTE_ARCHIVE: El archivo es del tipo archivo. Las aplicaciones usan este atributo para marcar las copias de seguridad o los removibles.
FILE_ATTRIBUTE_COMPRESSED: El archivo o directorio está comprimido. Para el archivo esto significa que todos los datos del archivo están comprimidos. Para un directorio esto significa que la compresión es por defecto aplicada a los archivos y subdirectorios nuevos creados.
FILE_ATTRIBUTE_NORMAL: El archivo no tiene otros atributos activos. Este atributo es válido sólo si esta solo, no hay ningún otro atributo.
FILE_ATTRIBUTE_HIDDEN: El archivo es oculto. El archivo no es incluido en la lista ordinaria de directorios.
FILE_ATTRIBUTE_READONLY: El archivo es de solo lectura. Las aplicaciones pueden leer el archivo pero no pueden ni escribirlo ni borrarlo.
FILE_ATTRIBUTE_SYSTEM: El archivo es parte del sistema operativo o es usado por este exclusivamente.

```
Invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
mov  hMemory,eax
invoke GlobalLock,hMemory
mov  pMemory,eax
```

Cuando se abre el archivo, reservamos un bloque de memoria para usar con las funciones ReadFile y WriteFile. Especificamos el flag GMEM_MOVEABLE para

dejar a Windows mover el bloque de memoria para consolidarla. La bandera GMEM_ZEROINIT le dice a GlobalAlloc que rellene el nuevo bloque de memoria reservado con ceros.
Cuando GlobalAlloc vuelve satisfactoriamente, eax contiene el manejador al bloque de memoria reservado. Le pasamos este manejador a la función GlobalLock que nos devuelve un puntero al bloque de memoria.

```
Invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
Invoke SendMessage,hwndEdit,WM_SETTEXT,NULL,pMemory
```

Cuando el bloque de memoria esta listo para ser usado, llamamos a la función ReadFile para leer los datos del archivo. Cuando el archivo es abierto o creado por primera vez, el puntero del archivo esta en el desplazamiento 0. Así que en este caso, empezamos a leer el primer byte del archivo. El primer parámetro de ReadFile es el manejador del archivo a leer, el segundo es el puntero al bloque de memoria para contener los datos, el siguiente es el número de bytes a leer del archivo, el cuarto parámetro es la dirección de la variable de tamaño DWORD que rellenaremos con el número de bytes realmente leídos del archivo.
Después de rellenar el bloque de memoria con los datos, ponemos los datos en el control de edición mandando el mensaje WM_SETTEXT al control de edición con lParam conteniendo el puntero al bloque de memoria. Después de esta llamada, el control de edición muestra los datos en el área cliente.

```
Invoke CloseHandle,hFile
Invoke GlobalUnlock,pMemory
Invoke GlobalFree,hMemory
.endif
```

En este punto, no necesitamos tener el archivo abierto por más tiempo ya que nuestra intención es grabar los datos modificados en el control de edición en otro archivo, no el archivo original. Así que cerramos el archivo llamando a CloseHandle con el manejador como su parámetro. Seguido desbloquearemos el bloque de memoria y lo liberamos. Actualmente no tienes que liberar la memoria en este punto, puedes usar el bloque de memoria durante el proceso de grabación después. Pero como demostración, yo he elegido liberarla aquí.

```
Invoke SetFocus,hwndEdit
```

Cuando la caja de diálogo abrir archivo es mostrada en la pantalla, el foco de entrada se centra en ella. Así que después de cerrar el diálogo de abrir archivo, tendremos que mover el foco de entrada otra vez al control de edición.
Esto termina la operación de lectura de archivo. En este punto, el usuario puede editar el contenido del control de edición. Y cuando quiera salvar los datos a otro archivo, deberá seleccionar File o Save en el menú que mostrará la caja de diálogo de salvar archivo. La creación de la caja de diálogo de salvar archivo no es muy diferente de la de abrir archivo. Efectivamente, se diferencian sólo en el nombre de la función, GetOpenFileName y GetSaveFileName. Puedes reciclar la mayoría de los miembros de la estructura ofn excepto el miembro Flags.

```
mov ofn.Flags,OFN_LONGNAMES or\
    OFN_EXPLORER or OFN_HIDEREADONLY
```

En nuestro caso, queremos crear un nuevo archivo.
Así que OFN_FILEMUSTEXIST y OFN_PATHMUSTEXIST deben ser dejados fuera, si no la caja de diálogo no nos dejara crear un archivo que no exista ya.
El parámetro dwCreationDistribution de la función CreateFile deberá ser cambiada a **CREATE_NEW** ya que queremos crear un nuevo archivo.
El resto del código es idéntico a todas las secciones de abrir archivo excepto las siguientes líneas:

```
invoke SendMessage,hwndEdit,WM_GETTEXT,MEMSIZE-1,pMemory
invoke WriteFile,hFile,pMemory,eax,ADDR SizeReadWrite,NULL
```

Mandamos el mensaje WM_GETTEXT al control de edición para copiar los datos del bloque de memoria, el valor devuelto en eax es la longitud de los datos dentro del buffer. Después de que los datos estén en el bloque de memoria, los escribimos en un nuevo archivo.

2.11 Librerías De Enlace Dinámico (DLL)

Si tu programa se agranda demasiado, encontrarás que los programas que escribes usualmente tienen algunas rutinas en común. Es una pérdida de tiempo rescribirlas cada vez que empiezas un nuevo programa. Volviendo a los viejos tiempos del DOS, los programadores almacenaban estas rutinas que usaban comúnmente en una o varias librerías. Cuando querían usar las funciones, enlazaban la librería al archivo objeto y el enlazador extraía las funciones de la librería y las insertaba en el ejecutable final. Este proceso se llama enlace estático. Las librerías de rutinas del lenguaje C son un buen ejemplo. La parte negativa de este método está en que tienes funciones idénticas en muchos programas que las usan. Tu espacio en disco se llena almacenando copias idénticas de las funciones. Pero para programas de DOS este método es bastante aceptable ya que suele haber un único programa activo en memoria. Así que no hay un desperdicio notable de memoria.
Bajo Windows, la situación es más crítica porque puedes tener varios programas funcionando al mismo tiempo. La memoria es consumida rápidamente si tu programa es bastante grande. Windows tiene la solución a este tipo de problemas: librerías de enlace dinámico. Las librerías de enlace dinámico son una especie de recopilación de funciones comunes. Windows no cargará varias copias de la DLL en la memoria de manera que si hay muchos programas que la usen solo corriendo al mismo tiempo, habrá una copia de la DLL en la memoria para todos estos programas. Voy a aclarar este punto un poco. En realidad, los programas que usan la misma DLL tendrán su propia copia de esta DLL. Esto hará parecer que hay varias copias de la DLL en memoria. Pero en realidad, Windows hace que esto sea mágico a través de la paginación de manera que todos los procesos compartan el mismo código de la DLL. Así que en la memoria física sólo hay una copia del código de la DLL. Como siempre, cada proceso tendrá su sección única de datos de la DLL.

El programa enlaza la DLL en tiempo de ejecución, no como en las viejas librerías estáticas. ¿Por qué se la llama librería de enlace dinámico? Sólo puedes descargar la DLL en el proceso cuando ya no la necesitas. Si el programa es el único que usa la DLL, será descargada de la memoria inmediatamente. Pero si la DLL todavía es usada por algún otro programa, la DLL continúa en memoria hasta que el último programa que la use la descargue.
Como siempre, el enlazador tiene el trabajo más difícil cuando fija las direcciones del archivo ejecutable final. Como no puede extraer las funciones e insertarlas en el ejecutable final, de alguna manera tendrá que almacenar suficiente información sobre la DLL y las funciones en el ejecutable final para poder localizar y cargar la DLL correcta en tiempo de ejecución.
Ahí es donde interviene la librería de importación. Una librería de importación contiene la información sobre la DLL que representa. El enlazador puede extraer la información que necesita de la librería de importación y mete esos datos en el ejecutable. Cuando el cargador de Windows carga el programa en memoria, ve que el programa enlace a una DLL, así que busca esta DLL, la proyecta en el espacio de direcciones del proceso y fija las direcciones para las llamadas a las funciones en la DLL.
Puedes elegir tú mismo cargar la librería sin dejárselo al cargador de Windows. Este método tiene sus pros y sus contras:
No se necesita una librería de importación, así que puedes cargar y usar cualquier librería siempre aunque no venga con librería de importación. Pero todavía tienes que saber algo sobre las funciones de su interior, cuantos parámetros toman y sus contenidos.
Cuando dejas al cargador que cargue la DLL para tu programa, si el cargador no puede encontrar la DLL desplegará el mensaje "El archivo DLL requerido, xxxxx.dll no ha sido encontrado" tu programa no tiene la oportunidad de correr aunque esta DLL no sea esencial para esa operación. Si cargas la DLL tú mismo, cuando la DLL no ha sido encontrada y no es esencial para la operación tu programa podrá advertir al usuario sobre el suceso y seguir.
Puedes llamar a funciones no documentadas que no están incluidas en la librería de importación. Suponiendo que conozcas suficiente información acerca de la función.
Si usas LoadLibrary tienes que llamar a GetProcAddress para todas las funciones que quieres llamar. GetProcAddress devuelve la dirección del punto de entrada de la función de una DLL en particular. Así que tu código será un poco mas largo o mas corto, pero nada mas.
Viendo las ventajas o desventajas de la llamada a LoadLibrary, ahora iremos detallando como crear una DLL.
El siguiente código es el esqueleto de una DLL.

```
;-------------------------------------------------------------
;                    DLLSkeleton.asm
;-------------------------------------------------------------
.386
.model flat,stdcall
option casemap:none
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
.data
.code
DllEntry proc hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
    mov eax,TRUE
    ret
DllEntry Endp
;-------------------------------------------------------------------------------
;                    Esta es una función ficticia.
; No hace nada. La he puesto esto aquí para mostrar donde puedes insertar las funciones
; dentro de una DLL.
;-------------------------------------------------------------------------------
TestFunction proc
  ret
TestFunction endp
End DllEntry
;-------------------------------------------------------------
;                  DLLSkeleton.def
;-------------------------------------------------------------
LIBRARY  DLLSkeleton
EXPORTS  TestFunction
```

El programa anterior es el esqueleto de una DLL. Todas las DLL deben tener una función de punto de entrada. Windows llamará a la función del punto de entrada en caso que:
La DLL es cargada por primera vez, la DLL es descargada, un hilo es creado en el mismo proceso, el hilo es destruido en el mismo proceso

```
DllEntry proc hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
    mov eax,TRUE
    ret
DllEntry Endp
```

Puedes nombrar la función del punto de entrada como quieras pero tendrás que terminarla END <Nombre de la función de entrada>. Esta función tiene tres parámetros, sólo los dos primeros de estos son importantes.
hInstDLL: Es el manejador del módulo de la DLL. Este no es el mismo que el manejador de la instancia del proceso. Tendrás que guardar este valor si necesitas usarlo mas tarde.
reason puede ser uno de los cuatro valores:
DLL_PROCESS_ATTACH: La DLL recibe este valor cuando es insertada por primera vez dentro del espacio de direcciones del proceso. Puedes usar esta oportunidad para hacer la inicialización.
DLL_PROCESS_DETACH: La DLL recibe este valor cuando va a ser descargada del espacio de direcciones del proceso. Puedes aprovechar esta oportunidad para hacer algo de limpieza y liberar memoria.
DLL_THREAD_ATTACH: La DLL recibe este valor cuando el proceso crea un nuevo hilo.
DLL_THREAD_DETACH: La DLL recibe este valor cuando un hilo en el proceso es destruido.
Devuelves TRUE en eax si quieres que la DLL siga funcionando. Si devuelves FALSE, la DLL no será cargada. Por ejemplo, si tu código de inicialización debe reservar algo de memoria y no puede hacerlo satisfactoriamente, la función del punto de entrada devolverá FALSE para indicar que la DLL no puede funcionar.

Puedes poner tus funciones en la DLL detrás o delante de la función de punto de entrada. Pero si quieres que puedan ser llamadas por otros programas debes poner sus nombres en la lista de exportaciones en el archivo de módulo de definición ".def".
DLL necesita un archivo de módulo de definición en su entorno de desarrollo. Vamos a echarle un vistazo a esto.

```
LIBRARY  DLLSkeleton
EXPORTS  TestFunction
```

Normalmente deberás tener la primera línea. La declaración LIBRARY define el nombre interno del módulo de la DLL. Tendrás que proporcionarlo con el nombre de archivo de la DLL. La definición EXPORTS le dice al enlazador que funciones de la DLL son exportadas, es decir, pueden ser llamadas desde otros programas. En el ejemplo, queremos que otros módulos sean capaces de llamar a TestFunction, así que ponemos el nombre en la definición EXPORTS.
Cualquier otro cambio es en las opciones del enlazador. Deberás poner /DLL opciones y /DEF: <el nombre de archivo de tu def> en las opciones de tu enlazador, algo así:

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

Las opciones del ensamblador son las mismas, esto es /c /coff /Cp. Así que después de enlazar el archivo objeto, obtendrás un .dll y un .lib. El .lib es la librería importada que puedes usar para enlazar a otros programas que usen las funciones de la DLL.
A continuación mostraré como usar LoadLibrary para cargar la DLL.

```
;------------------------------------------------------------------
;                        UseDLL.asm
;------------------------------------------------------------------
.386
.model flat,stdcall
option casemap:none
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
.data
LibName db "DLLSkeleton.dll",0
FunctionName db "TestHello",0
DllNotFound db "Cannot load library",0
AppName db "Load Library",0
FunctionNotFound db "TestHello function not found",0
.data?
hLib dd ?                       ; el manejador de la librería (DLL)
TestHelloAddr dd ?              ; la dirección de la función TestHello
.code
start:
    invoke LoadLibrary,addr LibName
;-------------------------------------------------------------------------
; Llama a LoadLibrary con el nombre de la DLL deseada. Si la llamada es correcta.
; devolverá el manejador de la librería (DLL). Si no devolverá NULL.
; Puedes pasar el manejador a GetProcAddress u otra función que requiera
; el manejador de la librería como parámetro.
;-------------------------------------------------------------------------
    .if eax==NULL
        invoke MessageBox,NULL,addr DllNotFound,addr AppName,MB_OK
```

```
    .else
        mov hLib,eax
        invoke GetProcAddress,hLib,addr FunctionName
;-------------------------------------------------------------------------------
; Cuando obtienes el manejador de la librería, lo pasas a GetProcAddress con la
; dirección del nombre de la función en la DLL que quieres llamar. Esto devuelve la dirección
; de la función si es correcto. De otra manera devuelve NULL.
; Las direcciones de las funciones no cambian a menos que descargues y recargues la librería .
; Así que puedes ponerlas en variables globales para usos futuros.
;-------------------------------------------------------------------------------
        .if eax==NULL
            invoke MessageBox,NULL,addr FunctionNotFound,addr AppName,MB_OK
        .else
            mov TestHelloAddr,eax
            call [TestHelloAddr]
;-------------------------------------------------------------------------------
; Lo siguiente, puedes llamar la función con un simple call con la variable conteniendo
; la dirección de la función como operando.
;-------------------------------------------------------------------------------
        .endif
        invoke FreeLibrary,hLib
;-------------------------------------------------------------------------------
; Cuando no necesitas mas la librería descargarla con FreeLibrary.
;-------------------------------------------------------------------------------
    .endif
    invoke ExitProcess,NULL
end start
```

Puedes ver que usando LoadLibrary es un poco mas problemático pero mas flexible.

2.12 CONTROLES COMUNES

Windows 95, 98, NT viene con varias ampliaciones de la interface de usuario, esas ampliaciones enriquecen la GUI. Algunas de ellas eran ampliamente usadas antes de que Windows 95, 98, llegara a los almacenes, tales como la barra de estado, barras de herramientas etc. Los programadores tenían que escribir el código para estas ampliaciones. Ahora Microsoft las ha incluido con Windows 9x y NT.

Estos son los nuevos controles:
Toolbar, Tooltip, Status bar, Property sheet, Property page, Tree view, List view, Animación, Drag list, Header, Hot-key, Image list, Progress bar, Right edit, Tab_Trackbar, Up-down.

Ya que hay muchos, cargarlos todos en memoria y registrarlos sería un despilfarro de recursos. Todos ellos, con excepción del control rich edit, están almacenados en comctl32.dll, desde donde las aplicaciones pueden cargarlas cuando se quiera usarlos. El control rich edit reside en su propia dll, richedXX.dll, porque es muy complicado y debido a que es más grande.
Puedes cargar comctl32.dll incluyendo una llamada a InitCommonControls en tu programa.
InitCommonControls es una función en comctl32.dll, así que refiriéndola en cualquier parte del código de tu programa hará que el cargador de archivos PE[31] cargue comctl32.dll cuando corra tu programa. No tienes que ejecutarla, sólo

[31] Ejecutables Portables o Portátiles

inclúyela en algún lugar de tu código. Esta función no hace NADA, su única instrucción es "ret", su único propósito es incluir la referencia a comctl32.dll en la sección de importación de manera que el cargador de archivos PE la cargue cada vez que el programa sea cargado, el verdadero canal de batalla es el punto de entrada de la función en la DLL que registra todas las clases de controles comunes cuando es cargada la dll. Los controles comunes son creados sobre la base de esas clases así como los controles de ventana hija tales como edit, listbox, etc.
El control rich edit es otra cosa. Si quieres usarlo, tienes que llamar a LoadLibrary para cargarlo explícitamente y luego llamar a FreeLibrary para descargarlo.
Ahora aprenderemos cómo crearlos, puedes usar un editor de recursos para incorporarlos en las cajas de diálogo o puedes crearlos tú mismo. Casi todos los controles comunes se crean llamando a CreateWindowEx o a CreateWindow, pasándole el nombre de la close del control. Algunos controles comunes tienen funciones de creación específicas, CreateWindowEx para facilitar la creación de esos controles. Las funciones de creación específicas existentes son:

CreateToolbarEx, CreateStatusWindow, CreatePropertySheetPage, PropertySheet, ImageList_Create.

Con el fin de crear controles comunes, tienes que saber sus nombres de clase, los cuales se muestran a continuación.

Aparecen en la siguiente lista:

Nombre de la Clase	Control Común
ToolbarWindow32	Toolbar
tooltips_class32	Tooltip
msctls_statusbar32	Status bar
SysTreeView32	Tree view
SysListView32	List view
SysAnimate32	Animación
SysHeader32	Header
msctls_hotkey32	Hot-key
msctls_progress32	Progress bar
RICHEDIT	Rich edit
msctls_updown32	Up-down
SysTabControl32	Tab

Los controles property sheets, property pages e image list, tienen sus propias funciones de creación específicas. Los controles drag list son cajas de listas potenciadas, así que no tienen su propia clase. Los nombres de las clases de arriba pueden ser verificados chequeando el guión de recursos generado por el editor de recursos de Visual C++. Difieren de los nombres de clase que aparecen

en la lista de la referencia de la API de Windows de Borland[32]. La tabla anterior es la precisa.
Esos controles comunes pueden usar estilos de ventana generales tales como WS_CHILD, etc. También tienen sus estilos específicos tales como TVS_XXXXX para el control tree view, LVS_xxxx para el control list view, etc. La referencia de la API de Win32 es tu mejor amigo en este punto.
Ahora que sabemos cómo crear controles comunes, podemos ver el método de comunicación entre los controles comunes y sus padres. A diferencia de los controles de ventanas hija, los controles comunes no se comunican con el padre a través de WM_COMMAND. En vez de eso, ellos envían mensajes WM_NOTIFY a la ventana padre cuando algunos eventos interesantes ocurren con ellos. El padre puede controlar al hijo enviándoles mensajes. También hay muchos mensajes para los nuevos controles. Deberías consultar tu referencia de la API de win32 para más detalles.
Vamos ver los controles de barra de progreso y barra de estado en el siguiente ejemplo.

```
.386
.model flat,stdcall
option casemap:none
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
.const
IDC_PROGRESS equ 1          ; IDs de los controles
IDC_STATUS equ 2
IDC_TIMER equ 3
.data
ClassName db "CommonControlWinClass",0
AppName   db "Common Control Demo",0
ProgressClass db "msctls_progress32",0      ; el nombre de la clase de la barra de progreso
Message db "Finished!",0
TimerID dd 0
.data?
hInstance HINSTANCE ?
hwndProgress dd ?
hwndStatus dd ?
CurrentStep dd ?
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke WinMain, hInstance,NULL,NULL, SW_SHOWDEFAULT
  invoke ExitProcess,eax
  invoke InitCommonControls
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
```

[32] Empresa dedicada a la creación de lenguajes de programación. Lenguaje de programación basado en lenguaje C

```
    mov   wc.cbWndExtra,NULL
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_APPWORKSPACE
    mov   wc.lpszMenuName,NULL
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
WS_OVERLAPPED+WS_CAPTION+WS_SYSMENU+WS_MINIMIZEBOX+WS_MAXIMIZEBOX+WS_VISIBLE,CW_USED
EFAULT,\
        CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
        hInst,NULL
    mov   hwnd,eax
    .while TRUE
       invoke GetMessage, ADDR msg,NULL,0,0
      .BREAK .IF (!eax)
       invoke TranslateMessage, ADDR msg
       invoke DispatchMessage, ADDR msg
    .endw
    mov eax,msg.wParam
    ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   .if uMsg==WM_CREATE
       invoke CreateWindowEx,NULL,ADDR ProgressClass,NULL,\
         WS_CHILD+WS_VISIBLE,100,\
         200,300,20,hWnd,IDC_PROGRESS,\
         hInstance,NULL
      mov hwndProgress,eax
      mov eax,1000
      mov CurrentStep,eax
      shl eax,16
      invoke SendMessage,hwndProgress,PBM_SETRANGE,0,eax
      invoke SendMessage,hwndProgress,PBM_SETSTEP,10,0
      invoke CreateStatusWindow,WS_CHILD+WS_VISIBLE,NULL,hWnd,IDC_STATUS
      mov hwndStatus,eax
      invoke SetTimer,hWnd,IDC_TIMER,100,NULL      ; crear un temporizador
      mov TimerID,eax
   .elseif uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
      .if TimerID!=0
         invoke KillTimer,hWnd,TimerID
      .endif
   .elseif uMsg==WM_TIMER                                    ; cuando ocurre un evento de temporizador
      invoke SendMessage,hwndProgress,PBM_STEPIT,0,0 ; incrementar el progreso en la barra de progreso
      sub CurrentStep,10
      .if CurrentStep==0
         invoke KillTimer,hWnd,TimerID
         mov TimerID,0
         invoke SendMessage,hwndStatus,SB_SETTEXT,0,addr Message
         invoke MessageBox,hWnd,addr Message,addr AppName,MB_OK+MB_ICONINFORMATION
         invoke SendMessage,hwndStatus,SB_SETTEXT,0,0
         invoke SendMessage,hwndProgress,PBM_SETPOS,0,0
      .endif
   .else
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .endif
   xor eax,eax
   ret
WndProc endp
end start
```

Análisis:

```
invoke WinMain, hInstance,NULL,NULL, SW_SHOWDEFAULT
invoke ExitProcess,eax
invoke InitCommonControls
```

Colocando InitCommonControls después de ExitProcess para demostrar que InitCommonControls está justo ahí para poner una referencia a comctl32.dll en la sección de importación. Como puedes ver, los controles comunes trabajan incluso si InitCommonControls no se ejecuta.

```
.if uMsg==WM_CREATE
   invoke CreateWindowEx,NULL,ADDR ProgressClass,NULL,\
     WS_CHILD+WS_VISIBLE,100,\
     200,300,20,hWnd,IDC_PROGRESS,\
     hInstance,NULL
   mov hwndProgress,eax
```

Aquí es donde creamos el control común. Esta llamada a CreateWindowEx contiene hWnd como manejador de la ventana padre. También especifica un ID de control para identificar este control. Sin embargo, como tenemos un manejador del control de ventana, este ID no es usado. Todos los controles de ventana hija debe tener el estilo WS_CHILD.

```
mov eax,1000
mov CurrentStep,eax
shl eax,16
invoke SendMessage,hwndProgress,PBM_SETRANGE,0,eax
invoke SendMessage,hwndProgress,PBM_SETSTEP,10,0
```

Después de que es creada la barra de progreso, podemos establecer su rango. El rango por defecto es desde 0 a 100. Si no estás satisfecho con esto, puedes especificar tu propio rango con el mensaje PBM_SETRANGE. lParam de este parámetro contiene el rango, el máximo rango está en la palabra alta y el mínimo está en la palabra baja. Puedes especificar cuánto toma un paso usando el mensaje PBM_SETSTEP. El ejemplo lo establece a 10 lo cual significa que cuando tú envías un mensaje PBM_STEPIT a la barra de progreso, el indicador de progreso se incrementará por 10. También puedes poner tu indicador de posición enviando mensajes PBM_SETPOS. Este mensaje te da un control más estricto sobre la barra de progreso.

```
invoke CreateStatusWindow,WS_CHILD+WS_VISIBLE,NULL,hWnd,IDC_STATUS
mov hwndStatus,eax
invoke SetTimer,hWnd,IDC_TIMER,100,NULL      ; crea un reloj
mov TimerID,eax
```

Luego, creamos la barra de estado llamando a CreateStatusWindow. Esta llamada es fácil de entender. Después de que es creada la ventana de estado, creamos un temporizador. En este ejemplo, actualizaremos la barra de progreso en un intervalo regular de 100 ms así que debemos crear un control de temporización. A continuación se muestra el prototipo de la función SetTimer.

```
SetTimer PROTO hWnd:DWORD, TimerID:DWORD, TimeInterval:DWORD,
lpTimerProc:DWORD
```

Donde:
hWnd: Manejador de la ventana padre
TimerID: Un identificador del temporizador, nunca igual a cero. Puedes crear tu propio identificador.
TimerInterval: El intervalo del temporizador en milisegundos que deben pasar antes de que el temporizador llame al proceso del temporizador o envía un mensaje WM_TIMER
lpTimerProc: La dirección de la función del temporizador que será llamada cuando el intervalo de tiempo expire. Si este parámetro es NULL, el temporizador enviará más bien el mensaje WM_TIMER a la ventana padre.
Si esta llamada tiene éxito, regresará el TimerID. Si falla, regresa 0. Esto se debe a que el valor del ID del temporizador debe ser diferente a cero.

```
.elseif uMsg==WM_TIMER
  invoke SendMessage,hwndProgress,PBM_STEPIT,0,0
  sub CurrentStep,10
  .if CurrentStep==0
    invoke KillTimer,hWnd,TimerID
    mov TimerID,0
    invoke SendMessage,hwndStatus,SB_SETTEXT,0,addr Message
    invoke MessageBox,hWnd,addr Message,addr AppName,MB_OK+MB_ICONINFORMATION
    invoke SendMessage,hwndStatus,SB_SETTEXT,0,0
    invoke SendMessage,hwndProgress,PBM_SETPOS,0,0
  .endif
```

Cuando expira el intervalo de tiempo especificado, el temporizador envía un mensaje WM_TIMER. Aquí pondrás el código que será ejecutado. En este ejemplo, actualizamos la barra de progreso y luego chequeamos si ha sido alcanzado el límite máximo. Si ha sido alcanzado, mataremos el temporizador y luego pondremos el texto en la ventana de estado con el mensaje SB_SETTEXT. Se despliega una caja de mensaje y cuando el usuario hace click sobre OK, limpiamos el texto en la barra de estado y en la barra de progreso.

2.13 Icono de Bandeja [Tray Icon]

La bandeja del sistema es la región rectangular en la barra de tareas donde residen la mayoría de los iconos. Normalmente, verás como mínimo un reloj digital aquí. También puedes poner iconos en la barra de sistema. Debajo están los pasos que tienes que seguir para poner un icono en la barra de sistema:
rellena la estructura NOTIFYICONDATA que tiene los siguientes miembros:
cbSize: El tamaño de esta estructura.
hwnd: Manejador de la ventana que recibirá la notificación cuando ocurran eventos de ratón sobre el icono de la bandeja.
uID: La constante que es usada como identificador del icono. Tú eres el único que decide este valor. En caso de que tengas más de un icono de bandeja, podrás chequear aquí desde cuál icono de bandeja proviene la notificación del ratón.
uFlags: Especifica qué miembros son válidos
NIF_ICON: El miembro hIcon es válido.
NIF_MESSAGE: El miembro uCallbackMessage es válido.
NIF_TIP: El miembro szTip es válido.

uCallbackMessage: El mensaje común que Windows mandará a la ventana especificado por el miembro hwnd cuando ocurren eventos sobre el icono de bandeja. Creas este mensaje tú mismo.
hIcon: El manejador del icono que quieres poner en la barra de sistema
szTip: Un arreglo de 64-bytes que contiene la cadena que será usada como texto tooltip cuando el ratón esté sobre el icono de bandeja.
Llama a Shell_NotifyIcon que está definida en shell32.inc. Esta función tiene el siguiente prototipo:

```
Shell_NotifyIcon PROTO dwMessage:DWORD ,pnid:DWORD
```

dwMessage: Es el tipo de mensaje a enviar al shell de Windows.
NIM_ADD: Añade un icono al área de estado.
NIM_DELETE: Borra un icono del área de estado.
NIM_MODIFY: Modifica un icono en el área de estado.
pnid: es el puntero a la estructura NOTIFYICONDATA rellenada con valores apropiados. Si quieres añadir un icono a la bandeja, usa el mensaje NIM_ADD, si quieres borrar un icono, usa NIM_DELETE.
Eso es todo lo que hay en ella. Pero la mayoría de las veces no estás contento con sólo poner un icono aquí. Necesitas ser capaz de responder a eventos de ratón sobre el icono de bandeja. Puedes hacer esto procesando el mensaje que especificas en el miembro uCallbackMessage de la estructura NOTIFYICONDATA.
Este mensaje tiene los siguientes valores en wParam y lParam:
wParam: contiene el ID del icono. Este es el mismo valor que pones en el miembro uID de la estructura NOTIFYICONDATA.
lParam: La palabra baja contiene el mensaje de ratón. Por ejemplo, si el usuario pulsa el botón derecho en el icono, lParam contendrá WM_RBUTTONDOWN.
La mayoría de los iconos de bandeja, como siempre, muestran un menú emergente cuando el usuario pulsa el botón derecho sobre éste. Podemos implementar esta función creando un menú emergente y entonces llamando a TrackPopupMenu para mostrarlo. Los pasos están descritos abajo: a continuación crear un menú emergente llamando a CreatePopupMenu. Esta función crea un menú vacío. Esto devuelve un manejador al menú en eax si el resultado es satisfactorio.
Añade elementos de menú con AppendMenu, InsertMenu o InsertMenuItem.
Cuando quieres mostrar el menú emergente donde está el cursor del ratón, llama a GetCursorPos para obtener las coordenadas de pantalla del cursor y entonces llama a TrackPopupMenu para mostrar el menú. Cuando el usuario selecciona un elemento del menú emergente, Windows manda el mensaje WM_COMMAND a tu procedimiento de ventana como si fuera una selección de menú normal.
Nota: Precaución con dos molestos comportamientos cuando usas menús emergentes con un icono de bandeja:

- ✓ Cuando el menú emergente es mostrado, si pulsas fuera del menú, el menú emergente no desaparecerá inmediatamente, como debiera. Este comportamiento ocurre porque la ventana que recibirá la notificación desde el menú emergente deberá ser la ventana de primer plano.

Así que la llamada a SetForegroundWindow corregirá esto.

- ✓ Después de llamar a SetForegroundWindow, encontrarás que la primera vez que se muestra el menú emergente, este trabaja bien pero en las siguientes ocasiones, el menú emergente se mostrará y cerrará inmediatamente. Este comportamiento es intencionado. El cambio de tarea al programa que es dueño del icono de bandeja en un futuro cercano es necesario. Puedes forzar este cambio de tarea enviando cualquier mensaje a la ventana del programa. Usa PostMessage, no SendMessage!

Ejemplo:

```
.386
.model flat,stdcall
option casemap:none
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
.data
ClassName  db "TrayIconWinClass",0
AppName    db "TrayIcon Demo",0
RestoreString db "&Restore",0
ExitString  db "E&xit Program",0
.data?
hInstance dd ?
note NOTIFYICONDATA <>
hPopupMenu dd ?
.code
start:
  invoke GetModuleHandle, NULL
  mov   hInstance,eax
  invoke WinMain, hInstance,NULL,NULL, SW_SHOWDEFAULT
  invoke ExitProcess,eax
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov  wc.cbSize,SIZEOF WNDCLASSEX
  mov  wc.style, CS_HREDRAW or CS_VREDRAW or CS_DBLCLKS
  mov  wc.lpfnWndProc, OFFSET WndProc
  mov  wc.cbClsExtra,NULL
  mov  wc.cbWndExtra,NULL
  push hInst
  pop  wc.hInstance
  mov  wc.hbrBackground,COLOR_APPWORKSPACE
  mov  wc.lpszMenuName,NULL
  mov  wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov  wc.hIcon,eax
  mov  wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov  wc.hCursor,eax
  invoke RegisterClassEx, addr wc
  invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
```

```
WS_OVERLAPPED+WS_CAPTION+WS_SYSMENU+WS_MINIMIZEBOX+WS_MAXIMIZEBOX+WS_VISIBLE,CW_USED
EFAULT,\
        CW_USEDEFAULT,350,200,NULL,NULL,\
        hInst,NULL
    mov   hwnd,eax
    .while TRUE
      invoke GetMessage, ADDR msg,NULL,0,0
      .BREAK .IF (!eax)
      invoke TranslateMessage, ADDR msg
      invoke DispatchMessage, ADDR msg
    .endw
    mov eax,msg.wParam
    ret
WinMain endp
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    LOCAL pt:POINT
    .if uMsg==WM_CREATE
      invoke CreatePopupMenu
      mov hPopupMenu,eax
      invoke AppendMenu,hPopupMenu,MF_STRING,IDM_RESTORE,addr RestoreString
      invoke AppendMenu,hPopupMenu,MF_STRING,IDM_EXIT,addr ExitString
    .elseif uMsg==WM_DESTROY
      invoke DestroyMenu,hPopupMenu
      invoke PostQuitMessage,NULL
    .elseif uMsg==WM_SIZE
      .if wParam==SIZE_MINIMIZED
        mov note.cbSize,sizeof NOTIFYICONDATA
        push hWnd
        pop note.hwnd
        mov note.uID,IDI_TRAY
        mov note.uFlags,NIF_ICON+NIF_MESSAGE+NIF_TIP
        mov note.uCallbackMessage,WM_SHELLNOTIFY
        invoke LoadIcon,NULL,IDI_WINLOGO
        mov note.hIcon,eax
        invoke lstrcpy,addr note.szTip,addr AppName
        invoke ShowWindow,hWnd,SW_HIDE
        invoke Shell_NotifyIcon,NIM_ADD,addr note
      .endif
    .elseif uMsg==WM_COMMAND
      .if lParam==0
        invoke Shell_NotifyIcon,NIM_DELETE,addr note
        mov eax,wParam
        .if ax==IDM_RESTORE
          invoke ShowWindow,hWnd,SW_RESTORE
        .else
          invoke DestroyWindow,hWnd
        .endif
      .endif
    .elseif uMsg==WM_SHELLNOTIFY
      .if wParam==IDI_TRAY
        .if lParam==WM_RBUTTONDOWN
          invoke GetCursorPos,addr pt
          invoke SetForegroundWindow,hWnd
          invoke TrackPopupMenu,hPopupMenu,TPM_RIGHTALIGN,pt.x,pt.y,NULL,hWnd,NULL
          invoke PostMessage,hWnd,WM_NULL,0,0
        .elseif lParam==WM_LBUTTONDBLCLK
          invoke SendMessage,hWnd,WM_COMMAND,IDM_RESTORE,0
        .endif
      .endif
    .else
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
    .endif
    xor eax,eax
    ret
WndProc endp
end start
```

Análisis:

El programa mostrará una simple ventana, cuando presiones el botón minimizar, se ocultará y pondrá un icono en la bandeja del sistema, cuando hagas una doble pulsación sobre el icono, el programa se restablecerá y borrara el icono de la bandeja de sistema. Cuando pulses el botón derecho sobre este, se mostrará un menú emergente. Puedes elegir si restaurar el programa o salir de él.

```
.If uMsg==WM_CREATE
  Invoke CreatePopupMenu
  mov hPopupMenu,eax
  Invoke AppendMenu,hPopupMenu,MF_STRING,IDM_RESTORE,addr RestoreString
  Invoke AppendMenu,hPopupMenu,MF_STRING,IDM_EXIT,addr ExitString
```

Cuando se crea la ventana principal, crea un menú emergente y le añade dos elementos. AppendMenu tiene la siguiente sintaxis:

```
AppendMenu PROTO hMenu:DWORD, uFlags:DWORD, uIDNewItem:DWORD, lpNewItem:DWORD
```

hMenu: es el manejador del menú al que quieres añadir el elemento
uFlags: le dice a Windows sobre el elemento de menú a añadir al menú ,tanto si es un bitmap o una cadena o un elemento que se auto dibuja, activado, borrado o desactivado etc. En nuestro ejemplo, usamos MF_STRING, lo que significa que el elemento del menú es una cadena.
uIDNewItem: es el ID del menú ítem. Este es un valor definido por el usuario que es usado para representar el elemento del menú.
lpNewItem: especifica el contenido del elemento, dependiendo de lo que especifiques en el miembro uFlags. Como nosotros hemos especificado MF_STRING en uFlags, lpNewItem deberá contener el puntero a la cadena a mostrar en el menú emergente.
Después de que es creado el menú emergente, la ventana principal espera pacientemente a que el usuario pulse el botón de minimizar.
Cuando la ventana es minimizada, recibe el mensaje WM_SIZE con el valor SIZE_MINIMIZED en wParam.

```
.elseif uMsg==WM_SIZE
  .If wParam==SIZE_MINIMIZED
    mov note.cbSize,sizeof NOTIFYICONDATA
    push hWnd
    pop note.hwnd
    mov note.uID,IDI_TRAY
    mov note.uFlags,NIF_ICON+NIF_MESSAGE+NIF_TIP
    mov note.uCallbackMessage,WM_SHELLNOTIFY
    Invoke LoadIcon,NULL,IDI_WINLOGO
    mov note.hIcon,eax
    Invoke lstrcpy,addr note.szTip,addr AppName
    Invoke ShowWindow,hWnd,SW_HIDE
    Invoke Shell_NotifyIcon,NIM_ADD,addr note
  .endif
```

Aprovechamos esta oportunidad para rellenar la estructura NOTIFYICONDATA. IDI_TRAY es una constante definida al principio del código fuente. Puedes ponerlo al valor que quieras, esto no importa porque tienes un único icono de bandeja. Pero si vas a poner varios iconos en la bandeja del sistema necesitarás un ID

único para cada icono de bandeja. Especificamos todas las banderas en el miembro uFlags porque especificamos un icono (NIF_ICON), especificamos un mensaje común (NIF_MESSAGE) y especificamos el texto tooltip (NIF_TIP). WM_SHELLNOTIFY es un mensaje común definido como WM_USER+5. El valor actual no es importante siempre que sea único. Yo uso aquí el icono winlogo como icono de bandeja pero puedes usar cualquier icono en tu programa. Cárgalo desde el recurso con LoadIcon y pon el manejador devuelto en el miembro hIcon. Finalmente, rellenamos el szTip con el texto que queremos que muestre el shell cuando el ratón está sobre el icono. Ocultamos la ventana padre para dar la ilusión de parecer minimizar al icono de bandeja. Después podemos llamar a Shell_NotifyIcon con el mensaje NIM_ADD para añadir el icono a la barra del sistema. Ahora nuestra ventana principal está oculta y el icono está en la bandeja del sistema. Si mueves el ratón sobre él verás el tooltip que muestra el texto que ponemos dentro del miembro szTip. Después, si haces una pulsación doble en el icono, la ventana principal reaparecerá y el icono de bandeja se irá.

```
.elseif uMsg==WM_SHELLNOTIFY
  .if wParam==IDI_TRAY
    .if lParam==WM_RBUTTONDOWN
      invoke GetCursorPos,addr pt
      invoke SetForegroundWindow,hWnd
      invoke TrackPopupMenu,hPopupMenu,TPM_RIGHTALIGN,pt.x,pt.y,NULL,hWnd,NULL
      invoke PostMessage,hWnd,WM_NULL,0,0
    .elseif lParam==WM_LBUTTONDBLCLK
      invoke SendMessage,hWnd,WM_COMMAND,IDM_RESTORE,0
    .endif
  .endif
```

Cuando ocurre algún evento de ratón sobre el icono de bandeja, tu ventana recibe el mensaje WM_SHELLNOTIFY que es el mensaje común que especificas en el miembro uCallbackMessage. Vuelve a llamar a esta recibiendo el mensaje, wParam contiene la ID del icono de bandeja y lParam contiene el mensaje de ratón actual. En el código anterior, primero chequeamos si este mensaje viene del icono de bandeja que nos interesa. Si esto ocurre, chequeamos el mensaje del ratón. Como estamos interesados únicamente en la pulsación derecha o en la doble pulsación izquierda, procesamos sólo los mensajes WM_RBUTTONDOWN y WM_LBUTTONDBLCLK.
Si el mensaje de ratón es WM_RBUTTONDOWN llamamos a GetCursorPos para obtener las coordenadas actuales en la pantalla del ratón. Cuando la función vuelve, la estructura POINT es rellenada con las coordenadas en pantalla del ratón. Por coordenada de pantalla, quiero decir la coordenada de toda la pantalla sin considerar a ninguno de los bordes de ninguna ventana. Por ejemplo, si la resolución de la pantalla es 640 x 480, la esquina inferior derecha de la pantalla es x==639 e y==479. Si quieres convertir la coordenada de pantalla a coordenada de ventana usa la función ScreenToClient.
Como siempre, para nuestro propósito, queremos mostrar el menú emergente en la posición actual del ratón con la llamada a TrackPopupMenu y esto requiere coordenadas de pantalla, podemos usar las coordenadas rellenadas directamente en GetCursorPos.
TrackPopupMenu tiene la siguiente sintaxis:

```
TrackPopupMenu PROTO hMenu:DWORD, uFlags:DWORD, x:DWORD, y:DWORD, nReserved:DWORD,
hWnd:DWORD, prcRect:DWORD
```

hMenu: Es el manejador del menú emergente a mostrar
uFlags: Especifica las opciones de la función. Dice donde posicionar el menú relativo a las coordenadas especificadas después y qué botón del ratón será usado para sacar el menú. En nuestro ejemplo usamos TPM_RIGHTALIGN para posicionar el menú emergente a la izquierda de las coordenadas.
"x e y" especifican la localización del menú en las coordenadas de pantalla.
nReserved: Deberá ser NULL
hWnd: Es el manejador de la ventana que recibirá los mensajes desde el menú.
prcRect: Es el rectángulo en la pantalla donde es posible pulsar sin desconsiderar el menú. Normalmente ponemos NULL aquí, de manera que cuando el usuario pulse en cualquier sitio fuera del menú emergente, el menú no es desconsiderado.
Cuando el usuario hace una doble pulsación sobre el icono de la bandeja mandamos el mensaje WM_COMMAND a nuestra ventana especificando IDM_RESTORE para emular la selección del usuario del elemento Restore del menú emergente, restaurando la ventana principal y borrando el icono de la bandeja del sistema. Para poder estar preparados para recibir el mensaje de doble pulsación, la ventana principal deberá tener el estilo CS_DBLCLKS.

Como se muestra a continuación:

```
    Invoke Shell_NotifyIcon,NIM_DELETE,addr note
mov eax,wParam
.If ax==IDM_RESTORE
    Invoke ShowWindow,hWnd,SW_RESTORE
.else
    Invoke DestroyWindow,hWnd
.endif
```

Cuando el usuario selecciona el elemento del menú Restore, borramos el icono de la bandeja llamando otra vez a Shell_NotifyIcon, ahora especificamos como mensaje NIM_DELETE. Lo siguiente es restaurar la ventana principal a su estado original. Si el usuario selecciona Exit en el menú, también borramos el icono de la barra y destruimos la ventana principal llamando a DestroyWindow.

El lenguaje ensamblador nos sirve para analizar los diferentes programas, ya que la mayoría de ellos, al ser descompilados nos generan un código de alto nivel o sea el ensamblador.

De hecho yo creo que la mayoría de los usuarios que programan, utilizaran el lenguaje de alto nivel tardarían mucho en hacer una sencilla aplicación, ya que no es muy amigable que digamos, y requiere de mucho mas instrucciones para generar unas pequeñas aplicaciones o la mas simple de la instrucción en cualquier lenguaje.

PROTECCIÓN DE SOFTWARE

Considero que este es un tema muy interesante a tratar y que mucha gente desconoce,ya que, quien haya escuchado hablar de las protecciones de programas debe saber que no hay ninguna protección imposible de desactivar o por lo menos no la ha habido hasta ahora. Siempre hay una persona más habil e inteligente que otra que con unos conocimientos de programación consigue desactivar hasta las protecciones más potentes.

Primero vamos aclarar qué es una protección de software y los tipos de protecciones que podemos encontrar.

Tenemos las protecciones anti-copy que no son más, como su nombre indica, que sistemas para evitar copiar software, ya sean juegos o programas de cualquier tipo. Dentro de éstas podemos encontrar las que al copiar un determinado CD hacen que en un lugar concreto de la grabación ésta se nos detenga misteriosamente. Pero también hay protecciones más curiosas que nos permiten copiar un CD perfectamente sin problemas, pero a la hora de ejecutar el software, éste no se ejecuta correctamente dándonos errores.

Otros tipos de protecciones muy famosas son las de los programas de evaluación que son programas que normalmente encontramos en CD´s de revistas de informática o también en Internet.

La peculiaridad de éstos es que podemos disfrutar de todas sus funciones al completo hasta que pasa un determinado tiempo, aproximadamente un mes, y tras pasar dicho tiempo pueden pasar tres cosas: La primera es que nos aparezca un mensaje al arrancar el programa o cada cierto tiempo mientras lo utilizamos. Este mensaje nos dice que lo registremos rellenando un formulario y pagando una cantidad de dinero.

A pesar de aparecer este mensaje podemos disfrutar de él. Lo segundo que puede pasar es que nos quedemos con el programa a medias, es decir, al pasar el plazo el programa desactiva algunas opciones, pero otras las dejas accesibles al usuario, con lo que nos quedamos con un programa a medias con el cual podremos hacer tareas pero nunca igual que si lo tuviéramos completo.

Y por último lo que puede pasar es que el programa quede inutilizable al pasar el plazo de evaluación que le corresponda y nos quedemos sin él.

Ahora bien, tras estos múltiples problemas tenemos que hablar de sus soluciones, que también las hay aunque creamos que es imposible. Para eso tenemos a los crackers, personas con conocimientos de programación que se dedican a desactivar estas protecciones sin ánimo de lucro, sólo por afición.

Esencialmente lo que los crackers hacen es utilizar descompiladores con los que pueden ver los códigos de programación del archivo que contenga la protección.

Una vez que han encontrado las líneas de la protección sólo tienen que modificarlas para hacer que deje de tener efecto.

Cuando el cracker ha terminado de modificar el archivo que tenía la protección, lo guarda y lo difunde, normalmente por Internet, para que la gente pueda disfrutar de ellos. Así, cuando el usuario encuentra el archivo sin protección no tiene más que sustituirlo por el original que sí tenía protección. Estos archivos ya sin protección son conocidos como cracks o parches. Todo esto nos puede llevar a pensar que los cracks son ilegales. En realidad no lo son, siempre que no tengan fines lucrativos.

Tenemos que pensar que la ley sí permite hacer copias de seguridad de software, para esto se debe tener la licencia del programa original con lo que se afirma que quieres tener un back up de tu programa y no conseguir un programa copiado que no tenías antes, y que lo quieres conseguir de una forma más barata.

Hoy en día los programadores protegen la mayoría de su software para evitar la piratería, pero a pesar de esto, siempre hay alguna forma de copiar el software que deseemos para hacer un backup. Así que si nos compramos cualquier tipo de juego y queremos tener una copia de seguridad de él, por si el original se nos dañara, sólo tenemos que disponer de nuestra licencia de registro del juego y encontrar el crack para ese juego.

Esto es la esencia de lo que debemos saber sobre el mundo de las protecciones pero hay mucho más.

Cuando estamos trazando un determinado software, podemos utilizar varias técnicas para llegar antes a la parte de código en la cual sospechamos que puede haber parte del sistema de protección. Aquí describiré algunos de los métodos más usados.

A lo retro: Este método se basa en trazar el programa hacia atrás, es decir, dejar que el sistema de protección se active y parar la ejecución justo después, cuando el software nos avise con un mensaje de error, a partir de ese instante trazaremos hacia atrás, esto lo haremos examinando el código, buscando un salto que nos aleje o nos aproxime de la función que muestra el mensaje: típicamente esta función suele ser una de las siguientes:

MessageBoxA, MessageBox, DialogBoxParam,DialogBoxParamA

Por Predicción: Este método se utiliza cuando se sospecha que una determinada función del API de Windows esta siendo usada para el funcionamiento del sistema de protección.

Implementación por método de predicción: Se pone un breakpoint "BPX[33]" en el SoftIce[34] a la función que se sospecha esta siendo usada y se carga, continua con la ejecución del software. A partir de ahí se continua trazando normalmente hasta llegar al punto clave. Muy usada cuando se quiere buscar la función que pinta un banner molesto o una pantalla nag[35] de inicio, o cuando se conoce el sistema de protección que esta usando.

Por referencia a una cadena conocida: Este método se usa cuando el sistema de protección muestra un mensaje de error o un mensaje dentro de un cuadro de diálogo.

Implementación por referencia: Se copia el mensaje de error, se desensambla el archivo con el W32DAsm, se busca dicha cadena en la lista de referencias a cadenas y se hace doble click en esta, se apuntan las direcciones donde hay una posible referencia y se examina el código que hay alrededor buscando un salto que nos aleje de la referencia a dicha cadena. Similar al trazado a lo retro. Se comprueba que sucede si se invierte el salto o preferiblemente se examinan las llamadas previas al salto si las hay, ya que estas pueden ser las llamadas que comprueban si todo está en orden, fecha, mochila[36], numero de serié, etc.

Por búsqueda de cadenas: Se utiliza cuando se sospecha que una determinada cadena de caracteres está siendo utilizada y no se encuentra donde debería estar por el método de referencias. Esto es debido a que el software puede almacenar ciertas cadenas en partes reservadas al código del programa, para evitar que estas sean directamente visibles en el desensamblador.

Implementación por cadenas: Se busca con un editor hexadecimal en el archivo que se sospecha contiene la cadena, o se busca la cadena en memoria con el SoftIce una vez ejecutado el programa y antes de que se ejecute el sistema de protección. Si se encuentra en memoria se pone un BPM[37] sobre esta para interrumpir al programa antes de realizar cualquier cálculo con ella.

Por búsqueda de una secuencia de códigos de operación: Se utiliza cuando se sospecha que una determinada secuencia de órdenes en ensamblador está siendo usada por el sistema de protección o defensa.

Implementación por códigos de operación: Conocida la cadena secuencia de códigos / bytes a buscar se realiza su búsqueda con editor hexadecimal y se opera según sea el caso. Muy usada para la detección de trucos anti-debugging.

---

[33] BreakPoint, método por el cual hace un salto sobre una interrupción pasando hacia otra
[34] Programa desemsamblador.
[35] Son pantallas o diálogos que aparecen al inicio o final de una aplicación
[36] Envoltura o encriptación que protege un archivo ejecutable
[37] Aplicación de un BreakPoint cargado en memoria

3.1 SISTEMAS DE PROTECCIÓN DE TIEMPO

Los sistemas de protección por tiempo pueden operar de distintas formas:

El software desde la instalación de si mismo, el software procede a su salida inmediata o en el peor de los casos a su desinstalación automática. Durante la salida o desinstalación del software este puede mostrar algún mensaje informando al usuario del hecho en cuestión. Puede que el software vuelva a funcionar durante X días si se vuelve a instalar

El software, si es así procede de la misma manera que en el caso anterior. La diferencia está en que el software dejará de funcionar a partir de una fecha determinada y no funcionará si se vuelve a instalar.

Para detectar estos sistemas de protección deberemos tener en cuenta los siguientes datos:

El software sabe, por lo que deberá guardar esta información en algún lugar, probablemente en algún archivo propio, para localizarlos se puede utilizar Filemon o en el registro de Windows podría ser RegMon.

El software debe, es decir la fecha en el momento en el que el programa está siendo ejecutado, correspondientes cálculos para comprobar los días que han pasado o si es la fecha límite. Para esto dispone de las funciones de fecha o tiempo del API de Windows siguientes:

```
VOID GetSystemTime(LPSYSTEMTIME lpSystemTime );

VOID GetLocalTime(LPSYSTEMTIME lpSystemTime );
```

Estas dos funciones pasan un puntero a una estructura de tipo SYSTEMTIME con la siguiente definición:

```
typedef struct SYSTEMTIME
{
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
}
SYSTEMTIME;
```

La función GetTickCount retorna el número de ticks transcurridos desde el arranque del ordenador, puede ser usada para controlar el tiempo de ejecución de una rutina, con esto podrían saber si hay alguien trazando el código.

```
DWORD GetTickCount(VOID)
```

Algunos programas que usan las MFC[38], utilizan la siguiente función para calcular fechas:

```
double difftime( time_t timer1, time_t timer0 ); // Devuelve la diferencia entre dos tiempos en segundos.
```

Con esto en mente ya podemos deducir cual será el método a seguir para detectar el sistema: poner breakpoints en cada una de estas funciones y trazar paso a paso a partir de donde se ejecutó la función que provocó el BPX. También podemos usar el método de las referencias a cadenas. Existen casos en los que estos métodos no nos servirán de mucha ayuda o simplemente no nos facilitarán la tarea, estos casos se dan cuando el software está protegido con algún sistema de protección comercial por tiempo como el TimeLock o VBox.

Este tipos de sistemas se basan en DLL's externas, por lo que será más práctico el desensamblado y trazado de estas. En algunos casos simplemente bastará parchear alguna función de la DLL para haber terminado con el sistema de protección, con la ventaja de que todos los programas basados en esa protección estarán automáticamente desprotegidos habremos crackeado cientos de aplicaciones que ni siquiera conocemos.

Otra de las formas en que se pueden ver las protecciones por tiempo es de la siguiente manera.

Lo que debes hacer en este caso, es encontrar la línea que especifica el número de días usado se compara a 30, y modificar el código.
Aquí es lo que el código puede parecerse:

```
mov ecx,1E              ; mueve 1E hex (30 dec) dentro de ecx
mov eax,[esp+10]        ; mueve número de días usado dentro de eax
cmp eax,ecx             ; compara eax (número de días usado) con ecx (30)
jl ...              ; si eax es menor de 30 entonces todavía podemos evaluar el programa.
```

Una manera simple de crackear esto sería cambiar mov eax,[esp+10], a mov eax,1 o algo así. Entonces el programa siempre pensará que estamos en nuestro primer día de evaluación. Otra manera de crackear esto sería cambiando el salto condicional, en un salto incondicional.

Si el programa que estás intentando crackear ha expirado, debes mirar en el Registro bajo CURRENT_USER y LOCAL_MACHINE y debes comprobar si hay cualquier entrada para el programa que estás usando y los borras, para reinstalar, o asegúrate de que borras todos los archivos que se instalaron la primera vez, también usa un monitor del registro como cleansweep para supervisar los archivos que tu programa coloca en la unidad.

[38] Microsoft Fundation Classes

Tu programa te permite probarlo durante 30 días por ejemplo. Simplemente piensa cómo un programador perezoso podría verificar esto o con qué podría parecerse en ensamblador, quizás con algo así:

```
CMP DWORD PTR register, 1E  <--1E son 30 días por supuesto
JLE/JGE address             <--probablemente estos saltos condicionales
```

Así que piensa en términos de código de operación, nosotros tenemos 83 para CMP, sólo varias opciones de registro probables, cuando has visto muchas pruebas tiempo empezarás a sentir qué registros normalmente se usan más, entonces 1E para los 30 días y/o 7E (JLE) o 7D (JGE) para el salto condicional.

Así que tomando a nuestro editor Hexadecimal y realiza un poco de investigación. Puedes encontrar esto en una o más situaciones.
Inmediatamente en esta situación siente si esto debe ser un control de prueba tiempo.

3.2 SISTEMAS DE PROTECCIÓN DE BANNERS O "NAGS"

Estos sistemas no son propiamente un sistema de protección, más bien son sistemas para molestar al usuario del software y recordarle que adquiera el programa original, los banners[39] se utilizan mucho en programas de visualización o de retoque de fotografías, se trata de textos o imágenes que tapan parcialmente el trabajo que estamos viendo o haciendo, impidiéndonos su correcta visualización.

Los nags son pantallas o cuadros de diálogo que aparecen al inicio o al final de la aplicación y están activos hasta que el usuario pulsa un determinado botón o hasta que se completa una cuenta atrás. Por lo tanto para que exista un banner o un nag, este debe ser mostrado de alguna manera, ahora sólo nos queda saber como se muestran los nags y como lo hacen los banners.

- **Técnicas para hacer frente a los nags:**

Lo primero sería identificar el tipo de nag, ¿es un cuadro de dialogo?, ¿es un cuadro de mensaje?, ¿O es otro tipo?. Esto lo podremos averiguar con facilidad si echamos una mirada al estilo del nag. Si hay 2 o menos botones, es probable que sea un cuadro de mensaje, si no hay botones o hay más de dos, probablemente será un cuadro de diálogo. Otra pista es observar las opciones que hay escritas en los botones, si dicen Continuar, Aceptar, Cancelar, OK, es probable que sea un cuadro de mensaje, otra posibilidad es fijarnos en los iconos del cuadro, si vemos un signo de admiración, de información o una cruz en un circulo rojo, seguramente se trate de un cuadro de mensaje.

[39] Concepto utilizado para ventanas comerciales, o como ventanas de dialogo informativas

Estas funciones son las que Windows puede utilizar para crear un cuadro de diálogo o mensaje, las cinco primeras son las más utilizadas, las cuatro últimas son sólo para cuadros de mensaje.

Anticipándonos a la creación del cuadro de diálogo/mensaje:
Implementación: Poniendo BPX en las siguientes funciones
CreateDialogIndirectParamA / CreateDialogIndirectParam
CreateDialogParamA / CreateDialogParamW
DialogBox
DialogBoxIndirect
DialogBoxParam / DialogBoxParamA / DialogBoxParamW
EndDialog
MessageBeep
MessageBoxA / MessageBoxW
MessageBoxExA / MessageBoxExW
MessageBoxIndirect / MessageBoxIndirectA / MessageBoxIndirectW

- **Por referencia a una cadena conocida.**

Una vez localizada la llamada al cuadro de diálogo o mensaje, procederemos a su eliminación, para ello lo más habitual será eliminar la llamada al cuadro de diálogo o mensaje mediante NOP o instrucciones similares, INC EAX, DEC EAX, etc. Es posible que después de la llamada al cuadro de diálogo, este retorne un valor el cual la aplicación deberá procesar para así continuar su ejecución, por eso deberemos tener en cuenta este valor y hacer que el registro o memoria que contenía dicho valor, siga valiendo igual al eliminar la llamada al cuadro de diálogo.

Esto lo podremos lograr aprovechando el espacio que nos quede al eliminar la llamada, o simplemente cambiando el salto que compruebe este valor.

- **Técnicas avanzadas de eliminación de nags.**

En ciertas ocasiones, no podremos eliminar la llamada al cuadro de diálogo o mensaje debido a que ésta llamada está codificada como una llamada indirecta, o cuando la aplicación usa la misma llamada para que se procese más de un cuadro de diálogo.

Es de imaginar que si eliminamos esta llamada, también estaremos eliminando otras posibles llamadas a otros cuadros de diálogo que no sean el molesto nag, como un cuadro de dialogo de configuración etc. La solución no es muy compleja todos los cuadros de diálogo que la aplicación pueda generar deben estar identificados por un número ID o cadena que los identifica indiscutiblemente.

Sabemos que todas las funciones de generación de cuadros de diálogos necesitan que este ID o cadena sea pasada como un argumento a la función que no podemos eliminar, por lo que podremos averiguar el ID del cuadro de diálogo que nos molesta mirando los push previos a la llamada. Una vez que tengamos dicho ID podremos desensamblar.

- **Técnicas para hacer frente a los banners:**

Por el tipo de banner: Si es un simple banner de texto podremos poner un BPX en las siguientes funciones.

DrawText
DrawTextEx
ExtTextOut
TextOut

Si el banner es una imagen:

BitBlt
MaskBlt
PatBlt

Tenemos que ir con mucho cuidado con estas funciones ya que son usadas por el GDI de Windows para pintar los textos de los controles, botones, listas, etc.

3.3 SISTEMAS DE PROTECCIÓN DE CD'S

Estos sistemas de protección comprueban que el CD del software se encuentra en la unidad de CD-ROM cada vez que el software se ejecuta.

Se utilizan para evitar lo que se llaman CD-Rips, versiones recortadas de la versión original del CD, sin intros, sin audio, etc. Usados por los piratas para introducir más de un juego en un sólo CD. Estos sistemas no evitan que el CDROM pueda ser duplicado, lo que significa que si introducimos una copia, el software funcionará correctamente. Yo casi ni los considero un sistema de protección, más bien son un sistema de fastidiar al usuario obligándole a introducir el CD, aunque el software no necesite ningún dato del CD para funcionar.

Personalmente me he encontrado con dos sistemas de este tipo:

- ✓ En el primer sistema el software identifica el CD-ROM mediante la etiqueta que este posee, si el CD tiene la etiqueta esperada el software continúa su ejecución.

- ✓ El segundo método se basa en comprobar si existe un determinado archivo dentro del CD-ROM, si el archivo existe, se continúa con la ejecución. La dificultad de estos sistemas de protección se ve aumentada cuando el software en realidad necesita los datos que hay en el CD para poder continuar, intros, pistas de audio, etc. El más útil es el método de predicción.

Técnicas utilizadas para hacer frente a los CD checks:

- **Anticipándonos a la detección de la unidad de CD**

Poniendo BPX en las siguientes funciones

```
AUINT GetDriveType(LPCTSTR lpRootPathName); // dirección del camino raíz
```

Esta función es usada por Windows para detectar el tipo de unidad pasada como cadena en lpRootPathName, si retorna 5, la unidad es un CD-ROM, a partir de aquí iremos trazando poco a poco hasta llegar a la comprobación del CD Recuerda que el BPX será BPX **GetDriveTypeA**, ya que es una función de 32 bits.

- **Anticipándonos a la detección de la etiqueta de la unidad de CD**

Poniendo un BPX en las siguientes funciones

```
BOOL GetVolumeInformationA(
LPCTSTR lpRootPathName,  // dirección del directorio raíz del sistema de archivos system
LPTSTR lpVolumeNameBuffer,  // dirección del nombre del volumen
DWORD nVolumeNameSize,  // longitud del buffer lpVolumeNameBuffer
LPDWORD lpVolumeSerialNumber,  // dirección del numero de serie del volumen
LPDWORD lpMaximumComponentLength,  // dirección de la máxima longitud del nombre del archivo del sistema
LPDWORD lpFileSystemFlags,  // dirección de los flags del sistema de archivos
LPTSTR lpFileSystemNameBuffer,  // dirección del nombre del archivo del sistema
DWORD nFileSystemNameSize  // longitud del buffer
);
GetVolumeInformationW
```

Estas funciones obtiene un puntero a una cadena conteniendo la etiqueta del volumen de la unidad especificada, a partir de aquí es fácil buscar donde se compara con la etiqueta válida. La función GetVolumeInformationW obtiene la cadena en formato Wide-char.

3.4 SISTEMAS DE PROTECCIÓN ANTI-COPIA

Los sistemas de protección anti-copia para discos son bastante antiguos, utilizados desde comienzos de la década de los 80 hasta hoy en día.

Aunque ya casi no se utilizan siempre es útil conocerlos, ya que algunas compañías de software siguen utilizando el sistema del disco llave sin el cual la aplicación no puede ejecutarse.

Esto resulta peligroso ya que los discos suelen estropearse físicamente o infectarse por algún virus con lo que el usuario debe solicitar otra copia a la compañía, pero esta le llega no podrá trabajar, este es uno de los motivos por los que este sistema está en desuso.

Los sistemas más antiguos protegían los discos de 5 1/4 y 3 1/2 mediante un formateo anómalo de los sectores, sectores formateados a un tamaño diferente de los que debería tener el disco.

Asignando un número de sector no válido a determinados sectores o incluso no formateando determinadas pistas.

Para poder determinar alguna de estas situaciones, deberemos conocer la tabla de parámetros de la unidad FDPB[40], esta tabla tiene la siguiente estructura:

OFFSET	FUNCIÓN	EJEMPLO
0	**Frecuencia de paso y descarga del cabezal:** los cuatro bytes de la izquierda (left nibble) de este valor es la frecuencia de paso del cabezal de la unidad de disco. Los 4 bytes derechos es el tiempo de descarga del cabezal de la unidad.	DF
1	**Tiempo de carga del cabezal:** el nibble izquierdo es el tiempo de carga del cabezal y el derecho es el modo dma seleccionado.	02
2	**Retardo de puesta en marcha del motor:** tiempo de espera hasta que el motor se apaga.	25
3	**Número de bytes x sector** 0 = 128 bytes 1 = 256 bytes 2 = 512 bytes 3 = 1024 bytes Modificando este valor podemos cambiar el tamaño del sector.	02
4	**Último número de sector** se usa para formatear e indica el número de sectores que hay en una pista.	12
5	**Longitud de gap:** este valor es el que debemos modificar si obtenemos errores de CRC al intentar leer un sector con tamaño no estándar	1B
6	**Longitud de datos de pista:** contiene el número de bytes en un sector cuando el valor del byte 4 no contiene 0, 1, 2, o 3.	FF
7	**Longitud del formato del gap:** número de bytes en el gap entre sectores: sólo se usa cuando se formatean pistas especiales.	54
8	**Byte de relleno de formato:** cuando se formatee una pista este será el valor de inicialización de los sectores	F6
9	tiempo de parada del cabezal	0F
A	tiempo de inicio del motor	02

Las líneas marcadas son la que nos serán útiles para detectar si el disco está protegido de alguna de las maneras anteriores.

Para comprobar los sectores del disco, la protección podemos utilizar los siguientes métodos:

Usar la interrupción de BIOS número 13 para el manejo de discos, pudiendo utilizar las siguientes subfunciones de esta:

INT 13,18 - Set Media Type for Format (Establecer tipo de medio para formatear)

AH = 18h
CH = 8 bits bajos del número de pistas (0-1023 dec)

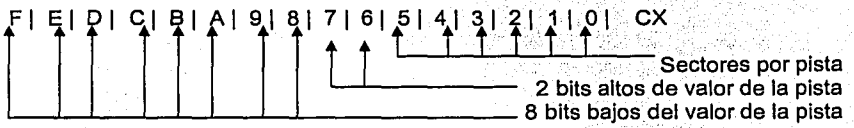
[40] Floppy Drive Parameter Block

CL = sectores por pista (1-17 dec)
DL = número de unidad (0=A:, 1=2nd floppy, 80h=unidad 0, 81h=unidad 1)

Retorna:

ES:DI = puntero a una tabla de 11 bytes Disk Base Table (DBT)
AH = 00h si la combinación solicitada es soportada
= 01h si la función no está disponible
= 0Ch si no está soportada o la unidad es desconocida
= 80h si no hay un disco en la unidad
CF = 0 si hay éxito 1 si hubo algún error

Válida sólo para BIOS XT/AT fechadas después del 11/15/1986, XT 286 y toda la línea PS/2.
Solo se comprueba la validez del número de disco.
El número de pista es un valor de 10 bits cogido de los 2 bits superiores de CL y los 8 bits de CH (bits bajos de la pista).



INT 13,2 - Read Disk Sectors (Leer sectores del disco)

AH = 02
AL = número de sectores a leer (1-128 dec)
CH = número de pista o cilindro (0-1023 dec)
CL = número de sector (1-17 dec.)
DH = número de cabeza (0-15 dec.)
DL = número de unidad (0= A, 1= 2nd floppy, 80h unidad 0, 81h= unidad 1)
ES: BX = puntero a un buffer donde copiar los datos leídos

Retorna:

AH = código de error

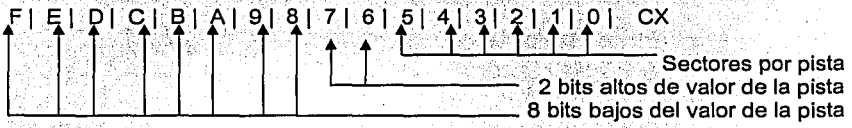
Uno de los posibles errores que podemos obtener es el error 9 DMA[41] Boundary error, indica que una rango ilegal fue cruzado cuando la información se escribió en la memoria. Si un offset de memoria acabado en tres ceros, ES:1000,ES:2000, cae en el medio de un área cubierta por un sector, este error ocurrirá. Esto puede indicar algún tipo de protección anti-copia.

AL = número de sectores a leídos
CF = 0 si hubo éxito = 1 si hubo algún error

Las lecturas de disco por BIOS deben ser ejecutadas por lo menos tres veces y la controladora debe ser reseteada si hubo algún error.
Asegurarse que ES:BX no cruza el límite de un segmento de 64K u ocurrirá un error de DMA.
Solo se comprueba la validez del número de unidad.

[41] Acceso Dinámico a Memoria

El número de pista es un valor de 10 bits cogido de los 2 bits superiores de CL y los 8 bits de CH (bits bajos de la pista).



Int 13,4 - Verify Disk Sectors (Verificar sectores del disco)

AH = 04
AL = número de sectores a verificar (1-128 dec.)
CH = número de pista o cilindro (0-1023 dec., see below)
CL = número de sector (1-17 dec.)
DH = número de cabeza (0-15 dec.)
DL = número de unidad (0=A:, 1=2nd floppy, 80h= unidad 0, 81h= unidad 1)
ES:BX = puntero a un buffer (si la BIOS es inferior al 11/15/1985)
Retorna:
AH = código de error
AL = número de sectores verificados
CF = 0 si hubo éxito = 1 si hubo error

Las verificaciones de disco por BIOS deben ser ejecutadas por lo menos tres veces y la controladora debe ser reseteada si hubo algún error.
Esta función hace que la controladora calcule el CRC de los datos del disco y los compare con los que hay almacenados en la cabecera del sector, retornando un error si estos valores no coinciden, esto es usado también por algunos sistemas de protección que alteran el CRC original.
BIOS anteriores al 11/15/1985 requieren que ES:BX apunte a un buffer válido solo se comprueba la validez del número de unidad.
El número de pista es un valor de 10 bits cogido de los 2 bits superiores de CL y los 8 bits de CH, bits bajos de la pista.

Ya sabemos que se usa la interrupción 13 para el sistema de protección, pero ¿Cómo procederemos para burlar el sistema?, lo normal será buscar todas las ocurrencias de la cadena CD13, código de operación de esta instrucción en el ejecutable y ver que tipo de acción realizan para así modificar lo que haga falta.

Pero hay casos en los que se llama a la interrupción sin usar la instrucción INT 13, esto se hace obteniendo un puntero a la INT 13 y luego saltando a esta dirección.

El siguiente esquema es equivalente a realizar una INT 13 para leer un sector de la unidad A, cara 0, pista 29h, sector FFh, y luego comprobar si el código de error es 10H.

cs:1000	mov	Ah,2	operación de lectura
cs:1002	mov	al,1	sector a leer

cs:1004	mov	Ch,29	pista 29h
cs:1006	mov	cl,ff	sector ffh
cs:1008	xor	Dx,dx	cara 0
cs:100B	xor	Bx,bx	unidad A
cs:100D	mov	Ds,bx	copia a DS
cs:100F	pushf		empuja flags a la pila
cs:1010	push	Cs	y CS
cs:1011	call	1100	empuja la dirección de la siguiente instrucción y salta a cs:1100
cs:1014	cmp	Ah,10	mira si hubo algún error de CRC
cs:1017	.....	.....	resto del código de verificación
cs:1100	pushf		empuja flags
cs:1101	mov	Bx,004c	puntero al vector de la INT 13 en la BIOS
cs:1104	push	[bx+2]	empuja el punto de entrada de la INT 13 a la pila
cs:1107	push	[bx]	
cs:1109	iret		saca cs,ip y flags, provocando un salto a la INT 13 y el posterior retorno a cs:1014

Este método de vector se basa en leer el vector de la int 13 desde la tabla que tiene almacenada la BIOS en la dirección 0000:0000, 4 bytes x 13h = 4Ch, esta tabla es un array de 256 entradas que apuntan a cada una de las 256 interrupciones posibles que posee un PC.

Una vez obtenido el vector, la rutina lo empuja a la pila y ejecuta una instrucción IRET que provoca la ejecución de la INT 13, esta al volver retorna a cs:1014 ya que previamente se hizo una llamada desde cs:1011. Si la protección usa este método no podremos detectarlo de la forma anterior.

Otra técnica se basa en sustituir la INT 13 por otra como la INT 10, y una vez que estemos ejecutando el programa, cambiar el código de la operación al vuelo, es decir, tendríamos un código auto modificable, pero prevendría de que se pudiera encontrar la INT 13 con un simple desensamblado.

También se puede obtener un vector a una interrupción usando un servicio del DOS, concretamente la:

INT 21,35 - Get Interrupt Vector
AH = 35h
AL = número de interrupción a obtener
Retorna:

ES:BX = puntero al controlador de la interrupción.

Este es el método estándar para obtener vectores de interrupción.
Pues ya sabes, al buscar las posibles INT 21,35 que aparezcan en el código y observar que vector piden. Un breakpoint para detectar si se intenta obtener el vector de interrupción 13, sería:

```
> bpx if (ax==3513)
```

Por último, se podría crear una rutina que fuera una copia del código original de la INT 13 e incluirla en nuestro programa, pero eso ya es más difícil de encontrar en un sistema de protección, aunque este seria el sistema más difícil de crackear.
Accediendo directamente a la controladora de discos vía puertos:

```
PORT 01F0-01F7 - HDC 1 (1st Fixed Disk Controller) (ISA, EISA)
PORT 0170-0177 - HDC 2 (2nd Fixed Disk Controller) (ISA, EISA)
01F0 RW registro de datos

01F1 R- registro de errores
Descripción de los Bits:
--- errores de diagnostico---
7 unidad que falló (0 = master, 1 = slave)
6-3 reservado
2-0 código de error
001 no hubo errores
010 error formato de dispositivo
011 error del sector del buffer
100 error de la circuiteria ECC
101 error del microprocesor de la controladora
--- errores de operación---
7 bloque erroneo detectado
6 error incorregible de ECC
5 reservado
4 ID encontrado
3 reservado
2 comando abortado prematuramente
1 pista 0 no encontrada
0 DAM no encontrado (siempre 0 para CP-3022)
01F1 -W WPC/4 (Cilindro de precompensación de escritura dividido entre 4)
01F2 RW número de sectores
01F3 RW número de sector (modo CHS mode) dirección del bloque lógico, bits 0-7 (modo LBA)
01F4 RW cilindro bajo (modo CHS) dirección del bloque lógico, bits 15-8 (modo LBA)
01F5 RW cilindro alto (modo CHS) dirección del bloque lógico, bits 23-16 (modo LBA)
01F6 RW unidad/cabeza

Descripción de los Bits:
7 =1
6 modo LBA activado, en vez del modo CHS
5 =1
4 selector de unidad (0 = drive 0, 1 = drive 1)
3-0 bits de selección de la cabeza (modo CHS mode) dirección del bloque lógico, bits 27-24 (modo LBA)
01F7 R- registro de estado

Descripción de los Bits:
7 la controladora está ejecutando un comando
6 unidad lista
5 fallo de escritura
4 posicionado completo
3 sector buffer requieres servicio
2 lectura de datos del disco se corrigió con éxito
1 índice - puesto a 1 cada revolución del disco
0 el comando previo termino en un error
01F7 -W registro de comando
00h opt nondata NOP
1xh opt nondata recalibrate 1
20h req PIOin read sectors with retry 1
21h req PIOin read sectors without retry 1
```

22h req PIOin read long with retry 1
23h req PIOin read long without retry 1
30h req PIOout write sectors with retry 2
31h req PIOout write sectors without retry 2
32h req PIOout write long with retry 2
33h req PIOout write long without retry 2
3Ch IDE opt PIOout write verify 3
40h req nondata read verify sectors with retry 1
41h req nondata read verify sectors without retry 1
50h req vend format track 2
7xh req nondata seek 1
8xh IDE vendor vend vendor unique 3
90h req nondata execute drive diagnostics 1
91h req nondata initialize drive parameters 1
92h opt PIOout download microcode
94h E0h IDE opt nondata standby immediate 1
95h E1h IDE opt nondata idle immediate 1
96h E2h IDE opt nondata standby 1
97h E3h IDE opt nondata idle 1
98h E5h IDE opt nondata check power mode 1
99h E6h IDE opt nondata set sleep mode 1
9Ah IDE vendor vend vendor unique 1
A1h ATAPI opt PIOin ATAPI Identify (see #P089)
B0h SMART opt Self Mon., Analysis, Rept. Tech. (see #P280)
C0h-C3h IDE vendor vend vendor unique 2
C4h IDE opt PIOin read múltiple 1
C5h IDE opt PIOout write múltiple 3
C6h IDE opt nondata set múltiple mode 1
C8h IDE opt DMA read DMA with retry 1
C9h IDE opt DMA read DMA without retry 1
CAh IDE opt DMA write DMA with retry 3
CBh IDE opt DMA write DMA w/out retry 3
DBh ATA-2 opt vend acknowledge media chng [Removable]
DCh ATA-2 opt vend Boot / Post-Boot [Removable]
DDh ATA-2 opt vend Boot / Pre-Boot (ATA-2) [Removable]
DEh ATA-2 opt vend door lock [Removable]
DFh ATA-2 opt vend door unlock [Removable]
E0h-E3h (second half of commands 94h-96h)
E4h IDE opt PIOin read buffer 1
E5h-E6h (second half of commands 98h-99h)
E8h IDE opt PIOout write buffer 2
E9h IDE opt PIOout write same 3
EAh ATA-3 opt Secure Disable [Security Mode]
EAh ATA-3 opt Secure Lock [Security Mode]
EAh ATA-3 opt Secure State [Security Mode]
EAh ATA-3 opt Secure Enable WriteProt [Security Mode]
EBh ATA-3 opt Secure Enable [Security Mode]
EBh ATA-3 opt Secure Unlock [Security Mode]
ECh IDE req PIOin Identify drive 1 (see #P087)
EDh ATA-2 opt nondata media eject [Removable]
EEh ATA-3 opt identify device DMA (see #P087)
EFh IDE opt nondata set features 1 (see #P281)
F0h-F4h IDE vend EATA standard
F1h Security Set Password
F2h Security Unlock
F3h Security Erase Prepare
F4h Security Erase Unit
F5h-FFh IDE vendor vend vendor unique 4
F5h Security Freeze Lock
F6h Security Disable Password

PORT 3510-3513 - ESDI primary harddisk controller

3510w R- status word
3510w -W command word
3512 R- basic status
3512 -W basic control
3513 R- interrupt status
3513 -W attention

PORT 3518-351B - ESDI secondary harddisk controller

3518w R- status word
3518w -W command word
351A R- basis status
351A -W basic control
351B R- interrupt status
351B -W attention

De todo lo que hay aquí, lo más interesante son los posibles accesos a los puertos 01F1 (registro de errores) y al 01F7 registro de estado, ambos registros son de sólo lectura, por lo que ya sabemos que las dos únicas maneras de comprobar el disco serán o por la INT 13 o por puertos. Todos estos accesos a los puertos y a la INT 13 podremos controlarlos con breakIpoints en el Softice[42], BPIO para los puertos y BPINT para las interrupciones, te aconsejo que uses breakpoints condicionales ya que si no podrías estar trazando toda la vida hasta llegar a lo que buscas. Aquí tienes unos ejemplos:

->BPINT 13 if (ah==4)
->BPINT 13 if (ah==2)
->BPINT 13 if (ah==18)

Los sistemas de protección anti-copia para CD, son relativamente nuevos. Son posteriores a la puesta en el mercado de las unidades de CD, ya que un principio estas no provocaron un boom debido a su escasa velocidad de lectura, con la aparición de las unidades de 4x, los freakers[43], comenzaron a copiar masivamente el software a este tipo de soporte, sabiendo donde buscar, una persona podía adquirir uno de estos CDs con más de 100 programas por un precio infinitamente inferior al precio de todo el software que contenía.

Posteriormente con la aparición de las consolas; PSX[44] y Saturn básicamente, la piratería se multiplicó. Las grandes casas de software decidieron entrar en combate y como no, una vez más idearon diversos sistemas de protección para su preciado software.

3.5 PROTECCIONES USANDO HUELLAS DIGITALES O MARCAS LÁSER

SecuROM: es un sistema patentado por Sony que identifica al verdadero CD-ROM con un sistema de autentificación especial. Durante el proceso de masterización del DADC una huella digital electrónica es aplicada en la capa de plástico que asigna un número a cada CD-ROM.
Identificación: Uno de estos ficheros debe estar en el directorio de instalación o en el directorio raíz del original.

CMS16.DLL
CMS_95.DLL
CMS_NT.DLL

[42] Aplicación capas de desensamblar o ensamblar programas ejecutables
[43] Personas encargada a la reproducción o copias de CD's
[44] Formato para consolas Playstation

Nota: En las últimas versiones de esta protección, estos ficheros no están en ningún directorio, si no que se encuentran almacenados en una de las secciones del ejecutable. El CD deberá contener este logotipo en el aro interior **DADC.**

Eliminación: Existen varios parches genéricos que funcionan con algunas versiones de esta protección

SafeDisc C-Dilla: Es un sistema de protección basado en software que no requiere cambios en los sistemas hardware. Consiste en la combinación de una firma digital incrustada en el disco y una encriptación multicapa del ejecutable que asegura el contenido del CD. La firma digital que no es posible copiar con una CDR es introducida mediante el láser durante la masterización del CD. Esta protección ha sido creada por la compañía C-dilla, propiedad de Macrovision Corporation.

Identificación: Los siguientes ficheros existen en el CD original:

00000001.TMP
CLCD16.DLL
CLCD32.DLL
CLOKSPL.EXE

El icono del archivo clockspl.exe es un CD tachado con dos barras. Actualmente se conocen varias versiones de esta protección y en las últimas los ficheros mencionados anteriormente ya no existen.

Eliminación: Existe también un programa creado por BlackChecks y otro creado por Risc que usando fuerza bruta, consiguen obtener una copia funcional del ejecutable original, partiendo incluso de la copia.

LaserLock: Protección que usa una combinación de encriptación de software y marca láser en la superficie del CD creada durante un proceso de masterización especial. Cada aplicación tiene un parámetro de encriptación diferente.

Identificación: El CD original contiene un directorio oculto llamado LASERLOCK. En este directorio hay archivos con errores de lecturas. En las últimas versiones este directorio ya no existe.

Protect-CD : Protección creada por VOB.

CD-Cops: Protección creada por Link Data Security y Spinner Software, es una protección añadida al ejecutable principal del CD. Las diferencias de tiempo son medidas para establecer una huella digital y para asegurar que no se pueda copiar. La huella habitualmente es expresada como un código de 8 dígitos o como un número clave.

DiscGuard: Protección creada por TTR Technologies Inc. Este sistema realiza dos cambios básicos en el software:

Los ejecutables principales están encriptados.
Una marca digital especial está inscrita en el disco esta marca contiene una llave de desencriptación de software. La marca digital no es reproducible mediante copia.

Cuando un disco original es utilizado la firma está presente, la desencriptación se produce y la aplicación funciona. Cuando es una copia, la firma no está presente, no existe desencriptación y la aplicación no funciona, en su lugar un mensaje, una pequeña demo, un link a una Web comercial o una presentación aparecen, transformando cada copia ilegal en una herramienta de marketing.

3.6 PROTECCIONES MEDIANTE LA ALTERACIÓN DE LA ESTRUCTURA DEL CD O DE LOS DATOS DE ESTE

Oversize: Algunos programas ocupan mas de 660 Mb y estos no pueden ser copiados por la mayoría de CDR's que no están preparadas para copiar mas de 659 Mb en un CD de 74 minutos. Con la aparición de los CD's de 80 minutos esta protección ya no es efectiva.

Illegal TOC: Esta protección se reconoce observando las pistas del CD. Normalmente aparece una segunda pista de datos, después de algunas de audio. Las normas estándar CD ISO no permiten eso, de ahí el nombre Tabla de contenidos ilegal.

Actualmente, todos los programas de duplicación de CD incluyen una opción para ignorar esta alteración, por lo que ya no es efectiva.

Dummy Files: Esta protección crea archivos falsos que señalan aleatoriamenté a una parte del CD usada por otros archivos. Cuando se copia el contenido del CD al disco duro la imagen es mucho más grande que el tamaño de las pistas originales, normalmente las pistas parecen tener 2 Gb.

Esta protección suele aparecer en combinación con la de Oversize. Para saltarse esta protección, si el original tiene menos de 659 Mb, hacer una copia directa (DAO/TAO) y recreara los archivos en el CD copiado. Cuando el CD tenga más de 659 Mb hacer lo mismo mediante overburning o usar un CD de 80 minutos.

Sectores Ilegibles: Este sistema es fácilmente reconocible por aparecer un aro en el CD. Este aro es una sección del CD estropeada, dentro hay unos ficheros que no se usan en el juego, pero que el programa analiza que estén y que no son posibles copiar. Las posibles maneras de saltarse esta protección son dos, usar el LeeTodo o BlindRead[45] o volcar el contenido del CD y crear los ficheros con cualquier editor de texto con la extensión del fichero corrupto.

[45] Programa encargado de saltarse los sectores defectuosos y sacar una imagen completa.

3.7 SISTEMA DE PROTECCIÓN MEDIANTE HARDWARE

Básicamente un dispositivo por hardware no es más que una caja de plástico que contiene un circuito de algún tipo, que puede variar en complejidad según el tipo de dispositivo.

Algunos dispositivos constan de memoria en los cuales se almacenan datos usados por el propio dispositivo. Los dispositivos más utilizados suelen ser las Sentinel Pro de Rainbow Technologies o las HASP de Aladdin Systems. Estas últimas tienen fama de ser las mejores en cuanto a protección se refiere, pero esto resulta cierto sólo en la medida en la cual el programador haya sabido implementar la protección.

En la mayoría de caso resulta penoso comprobar que todo el sistema depende de tan sólo la modificación de un sólo byte del ejecutable y demuestra que el programador ni siquiera se molestó en leer los manuales.

- **Cómo actuar ante cualquier dispositivo**

Los pasos a dar antes de empezar a desproteger una aplicación protegida con hardware son los siguientes:

1.- Identificar el tipo de dispositivo a la cual nos enfrentamos. No suele ser difícil en la mayoría de los casos, el dispositivo si tienes acceso a esta, comprueba los ficheros instalados por la aplicación, busca archivos DLL o VXD que contengan un copyright diferente al del autor del programa protegido. Algunos de los nombres más comunes que puedes encontrar son:

```
HASP*.VXD
HASP*.DLL
SENTINEL.VXD
```

2.- Recoger toda la información posible en las páginas del fabricante del dispositivo, manuales, FAQ's, todo lo que encuentres puede ser de gran ayuda.

Encontraras el API del hardware con la documentación de las operaciones que realiza cada función. No te asustes por lo que diga el manual: encriptación RSA[46], anti-debugging, etc. Efectivamente todo esto puede estar presente, pero la mayoría de veces no es así, además se trata de crackear la aplicación y no el hardware.

3.- Ahora ya disponemos de todo lo necesario, solo nos queda pensar un poco. Imaginemos que los programadores son bastantes ingenuos y su esquema de protección es de los más fáciles que podamos encontrar, primero siempre intenta una aproximación fácil del tipo.

[46] Metodo de encriptación para evitar una descompilación de un ejecutable.

```
call CheckDonglePresence
test eax,eax
jz NoDongle
...
...
@NoDongle:
```

Aunque te parezca increíble que esto pueda ser así, en la mayoría de los casos. Muchos programadores tan solo utilizan una función que devuelve TRUE o FALSE para comprobar si el hardware esta conectado. A veces realizan más de una comprobación por lo que deberemos parchear la rutina de comprobación en vez del salto jz NoDongle.

En la mayoría de casos la función CheckDonglePresence está contenida dentro de una DLL o VXD, los que previamente hemos encontrado, estas DLL o VXD están programados por los fabricantes del dispositivo y suelen estar bastante bien protegidos: técnicas anti-debugging, código auto-modificable, encriptación de datos etc. Mi consejo es no modificarlos si no es absolutamente necesario, ya que suele ser más fácil modificar la aplicación protegida.

La primero es encontrar la llamada, a veces más de una, para ello desensamblaremos la DLL que contenga el API con el W32Dasm o el IDA. En la lista de funciones exportadas, buscaremos cualquier nombre que levante sospechas estos pueden ser algunos:

```
IsDonglePresent
CheckDongle
ReadDongle
WriteDongle
```

Con el manual del API ahora podremos investigar las funciones una a una e identificar donde se guardan y que operaciones se realizan sobre los parámetros pasados a cada una de las funciones.

➢ MÉTODO 1

Los programadores tienen la manía de avisarnos cuando no se detecta el hardware, normalmente utilizan un cuadro de mensaje, por lo que procederemos de la manera siguiente:

1.- Establecer un breakpoint sobre la función que crea el cuadro de mensaje, generalmente.

```
MessageBoxA.
```

2.- Cuando el breakpoint surta efecto, pulsaremos F12 para volver a la función que lo llamó y examinaremos el código previo a la llamada en busca de algún salto condicional que este cercano. Si no encontramos nada pulsaremos de nuevo F12 y procederemos de la misma forma. Si encontramos algún salto sospechoso, prueba invertirlo y ejecuta de nuevo el programa haber lo que sucede.

➢ MÉTODO 2

Establecer un punto de ruptura en los accesos a puerto. Voy a tratar exclusivamente los dispositivos que se colocan en uno de los puertos paralelo, es decir un LPT[47], por ser estas las más extendidas. Sabemos que una PC tiene 65535 puertos distintos y que los puertos paralelos corresponden a:

```
LPT1 -> 378h
LPT2 -> 278h
```

Uno de los métodos más utilizados para encontrar donde se envían datos del hardware, se basa en utilizar un punto de ruptura sobre el puerto donde se encuentra instalado el dispositivo, si utilizamos el debugger SoftIce, esto lo haremos con el comando:

```
> BPIO 378
> BPIO 278
```

Esto hace que el SoftIce detenga la ejecución del programa cuando este intenta enviar datos al candado en alguno de los dos puertos. A partir de aquí podremos trazar el código paso a paso comprobando que es lo que hace el programa. Este método tiene ciertas desventajas: nos dejará justo en el API del candado con lo cual si el API esta preparado para detectar un debugger, podremos acabar con un cuelgue del sistema. Se obtienen mejores resultados con este método si la aplicación es de MS-DOS.

➢ MÉTODO 3

Establecer un punto de ruptura forzado en el API. Podemos colocar el código de operación CCh, INT 3, al inicio de cualquier función del API, esto implica modificar el archivo con editor hexadecimal y establecer un punto de ruptura sobre la INT 3 en SoftIce con:

```
> BPINT 3
```

Ahora ejecutaremos SoftIce y este se detendrá si se llama a esa función, ahora reemplazaremos el byte CCh por el que había originalmente con el comando.

Traza paso a paso y comprueba que sucede cuando no hay candado. Con suerte encontrarás una comprobación seguida de un salto tipo: jz NoDongle.

Invierte el salto y comprueba lo que sucede, quizás tu aplicación comience a funcionar.

Puedes intentar parchear el código para que emule el candado retornando un valor que satisfaga el proceso que llamó a la función. En la mayoría de los casos esto resulta suficiente.

---

[47] Referencia que se hace a los puertos paralelos

Si la aplicación se resiste, quizás sé este usando más de una función del API y estas también deberán de ser emuladas o crackeadas.

➢ MÉTODO 4

Establecer un punto de ruptura antes de cargar el API del dispositivo. Este método de ataque se basa en interrumpir la ejecución del programa cuando se carga la DLL o el VXD que maneja el candado. Para realizar esto podemos utilizar los siguientes breakpoints:

```
BPX LoadLibraryA
> BPX LoadLibraryExA
> BPX CreateFileA
```

Nota: la **A** final significa que son las funciones en su versión de 32Bits.
Las funciones LoadLibrary se encargan de cargar una DLL en memoria y la función CreateFile se utiliza para establecer una vía de comunicaciones con un dispositivo VXD, se utiliza para muchas más cosas, ver la guía de referencia del API de Windows, la descripción de estas funciones es como sigue:

```
HINSTANCE LoadLibrary(
LPCTSTR lpLibFileName // dirección que contiene un nombre de un fichero DLL
);
Parámetros:
lpLibFileName
```

El código anterior apunta a una cadena terminada en nulo que nombra al módulo ejecutable puede ser una .DLL o .EXE. El nombre especificado es el nombre del fichero y no tiene nada que ver con el nombre del módulo almacenado en el fichero como se especifica por la palabra clave LIBRARY en el fichero de definición del módulo (.DEF)

Si la cadena especifica un camino y el fichero no existe en el directorio especificado la función falla. Si no se especifica un camino y se omite la extensión, la extensión por defecto es .DLL. Aún así el nombre del fichero puede incluir el carácter punto "." Para indicar que el módulo no tiene extensión. Cuando no se especifica un camino la función busca el fichero mediante el siguiente criterio:

1.- El directorio desde donde se cargó la aplicación.

2.- Windows 9x: El directorio system the Windows. Windows NT: El directorio system de 32 bits de Windows SYSTEM32.

3.- Windows NT: El directorio system de 16-bit de Windows SYSTEM.

4.- Los directorios que aparecen en la variable de entorno PATH.

Retorna: Si hay éxito retorna el manejador del módulo. Si no hay éxito retorna NULL.

```
HANDLE CreateFile(
LPCTSTR lpFileName, // puntero al fichero
DWORD dwDesiredAccess, // tipo de acceso
DWORD dwShareMode, // modo de compartición
LPSECURITY_ATTRIBUTES lpSecurityAttributes, // puntero a los atributos de seguridad
DWORD dwCreationDistribution, // método de creación
DWORD dwFlagsAndAttributes, // atributos del archivo
HANDLE hTemplateFile // handle al fichero con los atributos a copiar
);
```

Parámetros que nos interesan:

lpFileName

Apunta a una cadena terminada en nulo que contiene el nombre del objeto a crear o abrir. Archivo, directorio, dispositivo de disco, recurso de comunicaciones, consola etc.

Retorna:

Si hay éxito, un handle abierto al objeto especificado.
Si falla, el valor es igual a INVALID_HANDLE_VALUE = -1

Este método presentado de esta manera suele ser bastante inútil, ya que las dos funciones anteriores son muy utilizadas y establecer un simple breakpoint sería bastante engorroso, ya que deberíamos de ir comprobando si realmente se está pasando como argumento el módulo que contiene el API del dispositivo para cada interrupción que haga el breakpoint, puede haber bastantes interrupciones.

La solución es bastante sencilla, utilizaremos un breakpoint condicional, el cual surtirá efecto cuando se pase como argumento las cuatro primeras letras del módulo, para ello deberemos saber que el primer argumento de la función se encuentra en ESP->4 y como es un puntero que apunta a una cadena quedará *(esp->4), por lo tanto para la función LoadLibrary haremos:

```
BPX LoadLibraryA IF *(ESP->4)=='SENT'
BPX LoadLibraryExA IF *(ESP->4)=='SENT'
```

Esto detendrá la ejecución cada vez que se intente carga una DLL la cual se llame sent*.*, con toda probabilidad si la DLL se llama sentinel.dll, este breakpoint surtirá efecto. Para realizar la misma operación con un VXD el breakpoint variará un poco debido a que cuando se desea abrir un VXD, el nombre de este debe de estar precedido de los 4 caracteres "\\.\", por lo tanto haremos:

```
BPX CreateFileA IF *(ESP->4+4)=='\\.\HASP'
```

El +4 se suma a la dirección en ESP->4 para saltarse los 4 caracteres que preceden a "HASP[48]". Esto detendrá la ejecución cada vez que se intente abrir un VXD cuyo nombre comience por "HASP". **Nota**: Este breakpoint es de mucha utilidad cuando tratamos con hardware de tipo HASP.

[48] Nombre del Hardware creado para ser candado de una aplicación

A partir de aquí si se ha abierto el VXD, la comunicación con el dispositivo se realizará a través del VXD mediante el handle devuelto por CreateFile, usando la función DeviceIOControl.

```
BOOL DeviceIoControl(
HANDLE hDevice, // handle al dispositivo de interés
DWORD dwIoControlCode, // código de control de la operación a realizar
LPVOID lpInBuffer, // puntero a un buffer que contiene información de entrada
DWORD nInBufferSize, // tamaño del buffer de entrada
LPVOID lpOutBuffer, // puntero a un buffer que recibe información de salida
DWORD nOutBufferSize, // tamaño del buffer de salida
LPDWORD lpBytesReturned, // puntero a una variable que recibirá el conteo de los bytes de salida
LPOVERLAPPED lpOverlapped // puntero a una estructura superpuesta para operaciones asíncronas
);
```

Con lo cual podremos establecer otro breakpoint cada vez que intente enviar datos al VXD de la mochila con:

```
BPX DeviceIOControl if (esp->4)==hDevice
```

Donde hDevice será el valor devuelto por CreateFile. Ahora podremos ir examinando los buffers que la función DeviceIOControl pasa al VXD y lo que esta retorna después de ser ejecutada.

En definitiva la idea general es encontrar algún punto donde se comparen los datos devueltos por el API con los que la aplicación espera e invertir el salto. A veces no es posible y la única solución es emular el funcionamiento del candado mediante el parcheado del código.

3.8 ANTI-DEBUGGING

- **Que es un debugger**

Se podría definir como un sistema de software integrado en un sistema informático con el fin de identificar los errores lógicos de los programas y proporcionar medios para enmendarlos. Tal software es utilizable al propio tiempo que se hacen funcionar los programas al objeto de que facilite información relativa a los procesos mientras estos están teniendo lugar. Un buen depurador puede disponer de mandatos para mostrar el contenido de la memoria y de los registros, e incluso para modificarlos y para provocar la ejecución de rutinas durante la presentación de datos importantes que faciliten el diagnóstico de fallos.

- **Cómo podemos detectarlos**

La detección de un debugger puede resultar complicada, aún así existen métodos muy sencillos que ofrecen buenos resultados.

Algunos de estos métodos utilizan instrucciones privilegiadas y resulta complejo utilizarlos cuando nuestra aplicación se ejecuta bajo Windows en ring3[49], no hay

[49] Nivel mínimo de privilegios sobre ejecución.

problema bajo DOS, esto se debe a que Windows no nos permite utilizar las instrucciones privilegiadas necesarias para completar la detección, este tipo de detecciones solo las podremos realizar si estamos en ring0[50], lo cual implica la creación de un VXD o la utilización de algún truco que nos permita entrar en ring0. Cabe destacar también la proliferación de métodos que detectan exclusivamente a nuestro debugger preferido: SoftIce, uno de los más utilizados, se denomina MeltIce, también es el más conocido y es el utilizado por los creadores del SoftIce para comprobar si este está en memoria, podemos encontrarlo en la DLL nmtrans.dll dentro del directorio donde se encuentra el SoftIce.

Frente a lo que contrariamente puedas creer, la detección de un debugger no ofrece en si ninguna protección, en realidad la protección se deriva después. Que hacer una vez que se ha detectado la presencia de un debugger. Muchas aplicaciones simplemente informan al usuario mediante algún cuadro de diálogo y después finalizan la aplicación. Otras aplicaciones son menos consideradas y se limitan a finalizar la aplicación sin mostrar mensaje alguno, este es un sistema aceptable y las hay que son más agresivas y ponen en peligro la integridad de nuestra máquina realizando un cuelgue inmediato, esto es muy peligroso y puede llegar a destruir información de otras aplicaciones.
En mi opinión ninguno de los métodos descritos anteriormente resulta efectivo, frente a estos casos la mayoría de los crackers sospecharían lo que sucede y tomarían medidas inmediatas, lo único que ganaríamos es dificultar por momentos el trabajo del cracker. Sería más efectivo actuar de la siguiente manera:

Continuar la ejecución del programa, el cracker no sospechará que hemos detectado y deberá investigar si desea saber con certeza que hemos detectado. Intentar volver loco al cracker, podemos alterar el funcionamiento de la aplicación para que el cracker no encuentre lo que busca o incluso podemos enviar comandos al debugger que desactiven los breakpoints, esto es posible si somos capaces de entrar en ring0 y al finalizar la aplicación los volvemos a activar, esto confundirá a muchos crackers y se pasarán un buen rato pensando porqué no funcionan sus breakpoints.

```
/*Función: IsSICELoaded, Descripción: Este método de detección es utilizado por la mayoría de compresores o
encriptadores que puedes encontrar en Internet, se basa en la búsqueda de la firma 'BCHK' usando la INT 3 que el
SoftIce intercepta para sus servicios. Funciona tanto con MS-DOS como con Windows. Retorna: TRUE si se detecta
SoftIce*/

__inline bool IsSICELoaded()
{
  _asm {
    push ebp
    mov ebp,'BCHK'    // 'BCHK' -> 4243484Bh
    mov eax,4         // Función 4h
    int 3             // Llama a la interrupción 3
    cmp al,3          // compara AL con 3
    setnz al          // si no es igual, SoftIce está presente
    pop ebp
  }
}
```

[50] Nivel máximo de privilegios sobre ejecución

El código anterior es uno de los más sencillos de implementar, se basa en el uso de la INT 3 servicio 4h, cuando se le basa el valor BCHK[51] en EBP esta retornará un valor diferente a 3 en AL. Los crackers suelen defenderse de este método con un breakpoint de interrupción como el siguiente:

```
BPINT 3 if al==4
```

Este breakpoint nos permitirá modificar el valor oportuno y así evitar la detección del SoftIce.

```
/*
Función: IsSICELoaded2
Descripción: Método de detección muy usado por los compresores o encriptadores, se basa en la INT 41, esta
interrupción es utilizada por Windows para comprobar si hay un debugger instalado. Solo funciona bajo Windows.
Retorna: TRUE si se detecta SoftIce
*/
__inline bool IsSICELoaded2()
{
  _asm {
     mov eax,0x4f            // AX = 004Fh
      int 0x41               // INT 41 CPU - MS Windows debugging kernel - DEBUGGER INSTALLATION CHECK
     cmp ax,0xF386           // AX = F386h si hay un debugger presente
     jz SoftICE_detected
      xor eax,eax
SoftICE_detected:
  }
}
```

El código anterior es también uno de los más sencillos de implementar, se basa en el uso de la INT 41 servicio 4fh esta retorna el valor F386h si hay un debugger presente. Los crackers suelen defenderse de este método con un breakpoint de interrupción como el siguiente:

```
BPINT 41 if al==4f
```

Este breakpoint nos permitirá modificar el valor oportuno y así evitar la detección de cualquier debugger que se este ejecutando.

```
/*
Función: IsSICELoaded3
Descripción: Método de detección muy usado por los compresores o encriptadores, similar al anterior poro usando
la INT 68. Solo funciona bajo Windows.
Retorna: TRUE si se detecta SoftIce
*/

__inline bool IsSICELoaded3()
{
  _asm {
     mov ah,0x43
     int 0x68
     cmp ax,0xF386
     jz SoftICE_Detected
     xor eax,eax
SoftICE_detected:
  }
}
```

[51] BoundsChecker = valor asignado a paso de parámetros

Parecido al anterior pero usando la INT 68. Los crackers suelen defenderse de este método con un breakpoint de interrupción como el siguiente:

```
BPINT 68 if ah==43

/* Función IsSICELoaded4
Descripción: El siguiente método sólo funciona si la aplicación se está ejecutando en modo real (MS-DOS), se basa
en el uso de la interrupción 2Fh, utilizada por Windows para obtener un punto de entrada al dispositivo VXD
identificado en el registro BX, el  identificador del Softice es 0202h
*/

__inline bool IsSICELoaded4()
{
   _asm {
      xor di,di
      mov es,di
      mov ax, 1684h
      mov bx, 0202h // VxD ID of winice
      int 2Fh
      mov ax, es     // ES:DI -> VxD API entry point
      add ax, di
      test ax,ax
      jnz SoftICE_Detected
   }
}
```

Los crackers suelen defenderse de este método con un breakpoint de interrupción como el siguiente:

```
BPINT 2f if (ax==1684) && (bx=0202)

// Función: IsSoftice9xLoaded
// Descripción: Comprueba si el controlador VXD del Softice para Win9x está
//cargado en memoria, para ello utiliza la función
// CreateFile del API de Windows, esta función se encarga (entre otras cosas)
//de establecer comunicación con los dispositivos VXD.
//Retorna: TRUE si Softice está en memoria.

__inline BOOL IsSoftice9xLoaded()
{
   HANDLE hFile;

  // "\\.\SICE" sin las secuencias de escape
   hFile = CreateFile( "\\\\.\\SICE",
                  GENERIC_READ | GENERIC_WRITE,
                  FILE_SHARE_READ | FILE_SHARE_WRITE,
                  NULL,
                  OPEN_EXISTING,
                  FILE_ATTRIBUTE_NORMAL,
                  NULL);

  // Si se retorna un handle válido, Softice está en memoria
   if( hFile != INVALID_HANDLE_VALUE )
   {
     CloseHandle(hFile);   // cierra la comunicación con el VXD
     return TRUE;          // y devuelve TRUE
   }
   return FALSE;           // Softice no detectado
}

// Función: IsSofticeNTLoaded
//Descripción: Comprueba si el controlador VXD del Softice para NT está
//cargado en memoria, para ello utiliza la función
// CreateFile del API de Windows, esta función se encarga (entre otras cosas)
//establecer comunicación con los dispositivos VXD.
// Retorna: TRUE si Softice está en memoria.
```

```
__inline BOOL IsSoftIceNTLoaded()
{
  HANDLE hFile;

  // "\\.\NTICE" sin las secuencias de escape
  hFile = CreateFile( "\\\\.\\NTICE",
            GENERIC_READ | GENERIC_WRITE,
            FILE_SHARE_READ | FILE_SHARE_WRITE,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

  if( hFile != INVALID_HANDLE_VALUE )
  {
    CloseHandle(hFile);
    return TRUE;
  }

  return FALSE;
}
```

Este código se denomina MeltIce, es de los más utilizados para detectar la presencia del SoftIce pero también es uno de los más fáciles de engañar. Si te decides por usar este método nunca pases la cadena que hace referencia al VXD como en el ejemplo, ya que si lo haces así, está aparecerá dentro del ejecutable como una cadena legible, por lo que solo tendremos que buscar dentro del ejecutable la cadena 'SICE' o 'NTICE' con un editor hexadecimal y reemplazarla por cualquier otro valor como 'XICE', anulando por completo nuestro sistema de protección, esto sucede la mayoría de los casos. Incluso si creamos la cadena dinámicamente carácter a carácter, siempre podemos detectar este método con un breakpoint condicional sobre la función CreateFile:

```
BPX CreateFileA if *(esp->4+4)=='SICE' || *(esp->4+4)=='SIWV' || *(esp->4+4)=='NTIC'
```

El funcionamiento de este breakpoint es sencillo, se comprueba que el primer parámetro pasado a la función CreateFileA apunte a la cadena SICE o NTIC, se suma 4 para omitir la cadena '\\.\' de caracteres de longitud, si se cumple esta condición, el breakpoint se ejecuta, Nótese el uso del condicional == y la operación lógica ||, similar a la sintaxis del lenguaje C. Cuando el breakpoint surta efecto, sólo deberemos modificar la cadena pasada para que contenga cualquier otro nombre que no sea un VXD de Windows.

Puedes utilizar los métodos para detectar si hay un breakpoint sobre la función CreateFile, pero estos métodos también pueden ser detectados. Lo cual deja a este sistema como uno de los más inocentes ante un cracker. Esto es debido a que depende de la DLL kernel32 que contiene la función CreateFileA y los sistemas de protección que dependen de DLL externas, que como ya sabemos, son fáciles de detectar.

A continuación se muestran los métodos más básicos que existen para la detección del SoftIce o de cualquier otro debugger que se ejecute bajo Windows.

➢ Método 1:

Descripción: Este método se suele utilizar muy poco, aunque es bastante efectivo ya que detectará el SoftIce aunque no este en memoria, su funcionamiento es muy simple: comprueba la existencia en el registro de una de las siguientes claves, esto delatará la presencia de nuestro estimado amigo SoftIce.

```
- #1 : HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\SoftICE
- #2 : HKEY_LOCAL_MACHINE\Software\NuMega\SoftICE
```

Para comprobar la existencia de claves en el registro podemos usar la función del API RegOpenKey, RegOopenKeyEx.
Para detectar este método podemos utilizar un breakpoint en el SoftIce tal como:

```
-> BPX _regopenkey if *(esp->8+0x13)=='tICE' || *(esp->8+0x37)=='tICE'
```

➢ Método 2:

Descripción: Este método es muy efectivo ya que detecta cualquier debugger, aunque solo funciona bajo Win9x, se basa en la comparación de las direcciones donde están los manejadores de la INT 1 y la INT 3. icy debe ser la dirección de un buffer de un mínimo de 6 bytes. El único método de detección de este método que conozco es por búsqueda de los opcodes de la instrucción SIDT (0F 01) o del CMP, pero este puede tener combinaciones más diversas.

Personalmente desconocía este método hasta que me encontré con el en el BlindWrite, un programa de grabación de CD-R que usa este sistema. La verdad que este sistema se puede explotar mucho más ya que de su funcionamiento se deriva un fallo en el Windows que nos permitiría ejecutar código en Ring0 desde nuestra aplicación en Ring3. El fallo reside en que el área de memoria donde reside a IDT[52], no está protegida contra lectura y escritura, pero esto solo sucede en Win9x y no en NT. La IDT viene a ser algo así como una tabla de vectores, donde se almacenan las direcciones de memoria que apuntan al código que ejecuta al llamar a una interrupción.

```
#include <windows.h>
// alinea las estructuras a WORD
#pragma pack(2)
typedef struct _IDTGATE
{
   unsigned short gateOffsetLow;
   unsigned short gateSelector;
   unsigned short gateFlags;
   unsigned short gateOffsetHigh;
} IDTGATE;

typedef struct _IDT
{
   unsigned short idtLimit;
   IDTGATE*   idtGate;
} IDT;
```

[52] Interrupt Descriptor Table = Tabla Descriptora de Interrupciones o Vectores.

```
bool IsDebuggerRunning()
{
  IDT icy;
  _asm {
    SIDT [icy];
    mov eax,dword ptr [icy.idtGate]
    add eax,0x8
    mov ebx, dword ptr [eax]
    add eax,0x10
    mov eax, dword ptr [eax]
    and eax,0x0000FFFF
    and ebx,0x0000FFFF
    sub eax,ebx
    cmp eax,0x1E
    je Sice
    xor eax,eax
Sice:
  }
}
```

➢ Método 3:

Descripción: Este método se basa en la búsqueda de la cadena WINICE.BR en la memoria, su funcionamiento es muy sencillo ya que es una simple búsqueda en memoria. Resulta muy efectivo y bien implementado puede ser difícil de detectar, aunque sabiendo la dirección de memoria que se analiza se puede utilizar un BPM[53] para detectarlo.

```
bool FindSoftIceString()
{
  _asm {
    mov   al, 'W'
    mov   edi, 0x10000
    mov   ecx, 0x400000-0x10000
More:
    repnz   SCASB           // "W"
    jecxz   NotFound
    cmp   dword ptr [edi], 'CINI'    // INIC
    jz   Ok1

    jmp   More
Ok1:
    add   edi, 4
    cmp   dword ptr [edi], 'RB.E'    // E.BR
    jnz   More
    mov   eax,1
NotFound:
  }
}
```

➢ Método 4:

Descripción: Consiste en ver si el registro de trazado DR7 contiene un valor distinto de 0, en caso afirmativo el sistema esta bajo un debugger. Esto podemos comprobarlo si estando en SoftIce tecleamos el comando CPU. Este método se basa en los registros de trazado "DRx[54]" de los Pentium para su correcto funcionamiento, la aplicación que lo utilice debe tener privilegios de Ring0, esto

[53] BreakPoint Cargado en Memoria.
[54] Debug Registers

implica programarnos nuestro VXD o hacer uso de algún tipo de truco que nos permita pasar de Ring3 a Ring0.

Existe un método que modificando la IDT nos permite redireccionar una interrupción a una dirección de código que este en nuestra aplicación, el código al ser ejecutado como una interrupción tendrá privilegios Ring0, el siguiente código demuestra como podemos implementar esto desde C. Nótese que este método no funciona bajo NT, también destacar que puede utilizarse otra interrupción que no sea la del ejemplo. Si por ejemplo elegimos la INT 1, el SoftIce se irá de bajada ya que él usa esta interrupción.

```
void Ring0Proc()
{
  _asm {
     mov   eax, dr7      // Instrucciones privilegiadas en Ring0 :)
     iretd               // Esto se ejecutará como una interrupción por lo que deberemos retornar tal y como lo
hacen las interrupciones
  }
}
DWORD GetDebugReg7()
{
  _IDT addr;                          // mirate el método 2 para ver la descripción de la estructura
  unsigned long lpOldGate;            // esto es para guardar la dirección original
  _asm {
     sidt fword ptr [addr]
     mov ebx,dword ptr [addr+2]
     add ebx,8*5
     mov   dx, word ptr [ebx+6]       // Salva el word alto de la puerta IDT
     shl   edx, 16d
     mov   dx, word ptr [ebx]         // word bajo
     mov   [lpOldGate], edx

     mov   eax, offset Ring0Proc      // Instala nuestra rutina Que se ejecutará en Ring0
     mov   word ptr [ebx], ax         // word bajo
     shr   eax, 16d
     mov   word ptr [ebx+6], ax       // word alto

     int 5                  // llama a la INT 5 que es la que hemos sustituido, esto ejecutará nuestra rutina en Ring0
     cmp eax,0
     jne Sice
     xor eax,eax
Sice:
     mov   ebx, dword ptr [addr+2]    // restaura la dirección original
     add   ebx, 8*5
     mov   edx, [lpOldGate]
     mov   word ptr [ebx], dx
     shr   edx, 16d
     mov   word ptr [ebx+6], dx

  }
}
```

- **Método 5:**

Descripción: Consiste en utilizar una función del VWIN32 que se encarga de despachar los servicios de la INT 41, a esta función se le pasan unos determinados valores que servirán para determinar la presencia del debugger. Solo funciona bajo Win9x, para poder detectar este método podemos utilizar uno de los siguientes breakpoints:

```
BPINT 41 if ax==4f , BPINT 30 if ax==0xF386, BPX Exec_PM_Int if eax==41 && edx->1c==4f && edx->10==002A002A
o BPX Kernel32!ord_0001 if esp->4==002A002A && esp->8==4f.
push 0000004fh      // función 4fh
push 002a002ah      // el word alto especifica el VXD (VWIN32), el word bajo especifica el servicio
(VWIN32_Int41Dispatch)
call Kernel32!ORD_001 // VxdCall
cmp ax, 0f386h      // número mágico devuelto por los debuggers
jz SoftICE_detected
```

- **Método 6:**

Descripción: Consiste en utilizar una llamada VXD que nos permite averiguar si hay instalado el controlador de dispositivo virtual que le indiquemos, para ello podemos utilizar dos de los identificadores de los que el SoftIce dispone 202h o 7a5fh. Este método solo funciona en Ring0 por lo que es bastante restrictivo a no ser que utilicemos el truco del método 4.

```
mov eax, Device_ID     // 202h para SICE o 7a5Fh para SIWVID VxD ID
mov edi, Device_Name // solo se usa si no tenemos el ID del VxD, en nuestro caso será 0 (NULL)
VMMCall Get_DDB
mov [DDB], ecx         // ecx=DDB si es dispositivo está instalado o ecx= 0 si el VxD no esta instalado
```

3.9 DETECCIÓN DE BREAKPOINT

- **Que es un breakpoint**

Los breakpoints son una de las grandes ventajas que todo debugger brinda al cracker, mediante un breakpoint, el cracker puede establecer un punto en el cual el debugger parará la ejecución del programa, posteriormente el debugger pasará el control del programa al cracker, con lo cual este podrá examinar que es lo que sucede al llegar al punto del código donde se estableció el breakpoint.

La mayor utilidad de los breakpoints es utilizarlos con funciones del API de Windows por ejemplo con MessageBoxA, esto nos permite tomar el control del programa antes de que se llame a la función del API sobre la que hayamos puesto un breakpoint. Es por eso que no es aconsejable informar al usuario de ningún error del sistema de protección mediante una función del API o de cualquier DLL externa.

- **Cómo funciona un breakpoint**

Los breakpoints siempre han funcionado de la misma manera a lo largo de la larga vida de los debuggers, cuando el usuario decide establecer un BPX en alguna parte del código, este manda una instrucción al debugger y el debugger reemplaza el primer byte del código por el código hexadecimal 'CCh', este código equivale a la instrucción INT 3, la cual genera una interrupción. Cada vez que esta instrucción se ejecuta, se devuelve el control al debugger, este, reemplaza la INT 3 (CCh) por el byte de código que había previamente y muestra al usuario el código actual donde se encuentra el programa.

Ahora el usuario puede seguir trazando el programa si lo desea y examinar el código para cambiar el comportamiento del programa, pudiendo llegar anular

nuestro sistema de protección. Esta claro que si pudiéramos detectar si en nuestro código se han puesto BPX, podríamos abortar la ejecución del programa debido a una violación del sistema de seguridad.

➢ Cómo podemos detectarlos

Pues bien, esto lo podemos conseguir fácilmente en ensamblador pero en C resulta algo más complicado, pero no por eso imposible. La teoría sería la siguiente: obtener un puntero a la función o al inicio del código que queremos comprobar, leer el primer byte de código, comprobar si es igual a 'CCh' (INT 3) y actuar en consecuencia. A continuación tienes un pequeño programa realizado en C que detecta si existe algún BPX sobre la función MessageBox del API de Windows. El código ha sido comprobado y funciona perfectamente compilándolo con Visual C++. Este código puede detectar cualquier debugger, ya que todos actúan igual.

```
#include <stdio.h>
#include <windows.h>
#pragma warning( disable : 4035 ) /* desactiva la generación de warnings para las funciones que aparentemente no
retornan nada*/

/* Las funciones están declaradas como funciones __inline, esto provoca que sean compiladas como si se tratara
de macros y no funciones, es decir, cada vez que en nuestro código en C llamemos a una de estas funciones, el
compilador no generará ninguna llamada si no que el código se añadirá cada vez que las llamemos. Esto evita que
el cracker pueda parchear la función y con esto evitar que todas las llamadas para comprobar si hay un BPX fuesen
inutilizadas con un solo cambio. Recomiendo hacer un uso bastante intensivo de cualquiera de estas dos
funciones para complicar la vida al cracker */

__inline bool IsBPX(void * address)
{
   _asm {
mov esi,address   // carga la dirección de la función
mov al,[esi]      // comprueba si existe un breakpoint sobre la función
cmp al,0xCC       // esto se realiza comprobando que el primer byte de código
                  // de la función sea diferente que CCh, que es el código
                  // de operación de la INT 3 usada por todo debugger
je BPXed          // si encontramos CCh, la función tiene puesto un breakpoint
                  // saltamos para devolver TRUE
xor eax,eax       // FALSE,
jmp NOBPX         // no hay breakpoint
BPXed:
mov eax,1         // hay un breakpoint
NOBPX:
   }
}
// Esta segunda versión de la función, realiza lo mismo, pero lo hace de una //manera menos evidente, para que el
//cracker no pueda buscar posibles
//comparaciones
// de un registro con C, en el desensamblado del código

__inline bool IsBPX_v2(void * address)
{
   _asm {
mov esi,address   // carga la dirección de la función
mov al,[esi]      // comprueba si existe un breakpoint sobre la función
mov ah,0x66       // carga ah con 66h (mitad de CCh)
add ah,ah         // lo suma a si mismo dando 0xCC, y se compara
cmp ah,al         // esto se realiza comprobando que el primer byte de código
                  // de la función sea diferente que CCh, que es el código
                  // de operación de la INT 3 usada por TODO debugger
je BPXed
```

```
xor eax,eax       // no hay breakpoint
jmp NOBPX
BPXed:
mov eax,1         // hay un breakpoint
NOBPX:
   }
}

#pragma warning( default : 4035 ) // establece la generación de warning para las funciones que no retornan valor, al valor por defecto del compilador

void main()
{
   void *addr;
   addr=MessageBox;

   if (IsBPX_v2(addr))
   {
      MessageBox(NULL,"Estas intentando crackear este programa?","Mensaje",MB_OK|MB_ICONEXCLAMATION);
   }
   else MessageBox(NULL,"Bien  pareces legal...","Mensaje",MB_OK);
}
```

Bueno, puedes ver que tampoco es tan difícil de implementar y resulta bastante efectivo, no creo que el código necesite mucha explicación ya que las partes interesantes están perfectamente comentadas, quizá la variable addr no es necesaria pero creo que así se ve más claro. Analicemos ahora los pros y contras de este código.

Si el cracker llega a detectar el código que causa la detección del BPX, puede crear un pequeño programa o utilizar un editor hexadecimal que busque los códigos que pertenecen a las funciones de detección y que realice las modificaciones oportunas, evitándose así tener que ir de una en una, esto es debido a que el compilador siempre generará el mismo código para las macros. Podríamos solucionar esto creando múltiples versiones de las macros con instrucciones diferentes.

El método es difícil de detectar si se utiliza una macro bien oculta, en este caso, la primera macro es muy sencilla de encontrar debido a la instrucción cmp al, 0xcc y probablemente sería poco efectivo ante un cracker poco experto.
Existe un método infalible para detectar la comprobación del breakpoint que muchos crackers ignoran, si en el SoftIce colocamos un BPM sobre la función que sospechamos, el debugger se detendrá justo después de leer el byte de código. En el ejemplo haríamos: BPM MessageBoxA R, esto le indica al Softice que detenga la ejecución si se intentan leer bytes de la función MessageBoxA.

Como puedes comprobar, cualquier truco anti-debugging no está exento de su correspondiente truco anti-anti-debugging.

3.10 ENCRIPTACIÓN XOR

La encriptación de datos es una de las mejores técnicas que podemos utilizar para proteger nuestras aplicaciones de los eventuales crackers. La encriptación XOR es uno de los métodos más sencillos que existen, su rapidez y fácil

implementación la hacen ideal para los creadores de virus y para cualquiera que desee implementar un sencillo método de encriptación.

La encriptación XOR, se basa en la propiedad lógica de dicha operación.

Pongamos dos valores 25h y 63h en binario:

00100101-> 25h
01100011 -> 63h

Según el funcionamiento de la operación lógica XOR, los bits del primer operando que sean iguales a los del segundo, resultaran en 0 (false) y el resto de bits en 1 (true) por lo tanto si realizamos el XOR de 25h con 63h obtendremos:

01000110 -> 46h

Como podemos comprobar se cumple la lógica de esta instrucción. Si ahora tomamos este valor y volvemos a realizar un XOR con 63h.

01000110 -> 46h
01100011 -> 63h XOR

Obtendremos:

00100101 -> 25h

El dato original, esta es la base de la encriptación XOR, pongamos ahora un ejemplo genérico.

Si tenemos un dato A, que deseamos encriptar y una llave de encriptación B, efectuaremos una operación XOR tal que así:

A XOR B = C

Donde C será el valor encriptado. Si sustituimos las variables por datos:
25h XOR 63h = 46h

A = 25h
B = 63h
C = 46h

Posteriormente para recuperar el dato encriptado (A) haremos:

C XOR B = A

Que usando los anteriores valores queda.

46h XOR 63h = 25h

En este ejemplo se ha utilizado una llave (B) de 8 bits, el tamaño de la llave puede ser más grande 16,32 o más bits, si la llave es más grande, mayor seguridad, ya

que hay más llaves posibles. Aquí tienes un resumen de las posibles combinaciones según el tamaño de la llave.

Para una llave de 8bits tiene un total de 256 = 2^8 llaves diferentes.

Para una llave de 16bits tenemos un total de 65535 2^16 = llaves diferentes.

Para una llave de 32bits tenemos un total de 4294967295 = 2^32 llaves diferentes.

Para una llave de 64bits tiene un total de 18446744073709551616 = 2^64 llaves diferentes.

Para una llave de 128 bits tenemos:
3.402823669209384634633746074317 7e+38 = 2^128 llaves diferentes.

Hay que tener en cuenta esto si vamos a implementar una protección que utilice encriptación, ya que si se intenta un ataque por fuerza bruta sobre nuestra encriptación y nuestra llave es de 8 bits, el tiempo necesario para conseguir una llave válida será muy inferior que si por el contrario nuestra llave es de 32bits o superior. Todo esto no sirve de mucho si no se toman las medidas necesarias para ocultar la llave, para que me entiendas, no sirve de nada tener una súper puerta de seguridad anti-robo y a prueba de bombas y todo lo que tú quieras. Por lo tanto deberemos ocultar nuestra preciada llave de cualquier ladrón.

Un último comentario, en la encriptación XOR siempre podemos encontrar la llave si tenemos los datos encriptados y los datos sin encriptar con un simple XOR de ambos valores:

```
B = A xor C

46h = 25h XOR 63h
```

Para encriptar un bloque de datos en assembler de 32 bits haríamos algo así:

```
mov esi,Original_Data_Ptr // carga puntero al buffer a encriptar
mov edi,Crypted_Data_Ptr // carga puntero al buffer donde guarda los datos
mov ecx,Data_Length // carga longitud del buffer a encriptar
or ecx,ecx // comprueba si es 0
jz @NoEncrypt // si es 0 no encripta
mov al,XORKey // carga la llave XOR en AL
@NextByte:
mov ah,byte ptr [esi] // lee byte a encriptar
xor ah,al // encripta
mov byte ptr [edi],ah // y guarda buffer destino
inc esi // pasa al siguiente byte del buffer
inc edi // pasa al siguiente byte del buffer
dec ecx // decrementa contador de bytes encriptados
jnz @NextByte // continua si aún quedan más
@NoEncrypt:
....
....
en 16 bits seria:
....
....
les si,Original_Data_Ptr
```

```
lds di,Crypted_Data_Ptr
mov cx,Data_Length
or cx,cx
jz @NoEncrypt
mov al,XORKey
@NextByte:
mov ah, byte ptr es: [si]
xor ah,al
mov byte ptr ds:[di],ah
inc si
inc di
dec cx
jnz @NextByte
@NoEncrypt:
....
....
```

La desencriptación se realizaría de la misma manera pero pasando en Original_Data_Ptr el buffer encriptado y en Crypted_Data_Ptr el buffer donde desencriptar. Esta es una de las ventajas de la encriptación XOR, no hace falta escribir otra función que desencripte, ya que se puede utilizar la misma. Por eso muchos virus utilizan este tipo de encriptación: ocupa poco espacio, es rápida y sirve tanto para encriptar como para desencriptar.

3.11 Puertas de llamada y VXD Calls

Antes de seguir debo decir que la información que he encontrado en la red sobre estos temas ha sido bastante pobre en lo que a explicaciones se refiere, tras un trabajo de investigación y de atar unas cosas con otras presento el resultado de mi investigación que puede no ser correcta.

La información que aquí se brinda puede resultar extremamente peligrosa si se hace un uso indebido de ella. Las siguientes técnicas pueden aplicarse para la construcción de virus y programas que pueden destruir el sistema sin que el usuario ni tan siquiera reciba ningún mensaje de error.

Pero también pueden ser utilizadas para crear herramientas que nos sean de gran utilidad en el noble arte del cracking, como volcadores de memoria, API, Monitoreadores, y parcheadores de procesos.

Puertas de llamada, para conseguir el nivel de privilegio máximo sobre el S.O. Pues los niveles de privilegio de una aplicación son como una especie de autorización que el S.O concede a cada tarea para que trabaje.

En principio de una PC puede tener 4 niveles de privilegio y a mayor privilegio menos nivel, es decir, el nivel máximo de privilegio que una aplicación puede tener es el nivel 0, o Ring 0, y el mínimo el nivel 3 o Ring 3. Si no me equivoco Windows no utiliza los 4 niveles de privilegio si no que solo usa el 0 y el 3, siendo estos el de supervisor y el de usuario respectivamente.

En nivel 0 se ejecuta el código del S.O y normalmente el código que utilizan los controladores de dispositivos, conocidos estos también por tener la extensión VXD, por último el resto de aplicaciones se ejecutan en nivel 3.

Una aplicación en nivel 3 esta muy limitada en cuanto a acceso directo al hardware se refiere, las instrucciones de E/S son exclusivas del nivel 0 y una aplicación en nivel 3 no puede acceder directamente a la memoria usada por otra aplicación, o por poner un ejemplo no puede acceder a una posición de memoria que este fuera de la memoria asignada a esa tarea.

Para nosotros esto significa la imposibilidad teórica, de poder alterar el código del programa en caliente, interceptar las llamadas al API y monitorizarlas con una función propia, y más imposible es aún alterar el propio funcionamiento del S.O.

Todo esto antiguamente no existía ya que el DOS, no distinguía entre privilegios, ni de protección de memoria.

Algunas de las cosas que he explicado anteriormente si son posibles de realizar mediante el uso de algunas funciones del API de Windows, pero seguimos estando limitados en muchos aspectos.
La solución a nuestro problema se llama puertas de llamada.

La aplicación en cuestión utilizaba el método de comparación de vectores de la INT 1 y la INT 3. Y que puede ser fácilmente burlado. Tras una breve sesión de trazado de código fuente, me encontré con algo que me hizo sospechar bastante. Primero encontré una llamada a la función VMM Get_DDB (Get Device Description Block) seguida de Test_Debug_Intalled, estas dos funciones forman parte del VXD VMM[55], son privilegiadas y solo pueden ser ejecutadas por código que corra en un nivel de privilegio igual a 0, además se suelen usar para detectar el SoftIce.
Entonces si las aplicaciones se ejecutan en nivel 3, ¿Cómo es posible que se pudieran estar ejecutando estas dos funciones privilegiadas?. Pues de alguna manera la aplicación debía de poder entrar en Ring 0, y para ello debía de utilizar algún método. Primero pensé que era posible que la aplicación estuviera utilizando un método, este método se basa en reemplazar un vector de interrupción de la IDT, por la dirección del código que queremos que se ejecuta al llamar a esta interrupción, pero tras algunas comprobaciones deseche esa teoría.

pushad		guarda el contenido de los registros
push	ebx	y empuja EBX a la pila, esto lo hace así para no tener que guarda el resultado de la siguiente instrucción en un buffer a parte
sgdt	fword ptr [esp-2]	lee el registro de la Tabla Global de Descriptores (GDT) de la tarea actual en la pila
pop	ebx	carga base de la GDT desde la pila
xor	eax,eax	borra EAX
sldt	eax	y carga el registro de la Tabla Local de Descriptores en AX (Local Descriptor Table Register LDTR)

[55] Virtua Memory Manager = Manejador de Memoria Virtual

and	al,f8	borra nibble alto y ultimo bit del nibble bajo, esto elimina los bits de privilegio RPL y TI del selector y deja el Indice a la GDT en AX
add	eax,ebx	suma la base de la GDT al indice
mov	ch,byte ptr [eax+7]	usa resultado como indice para llenar CH
mov	cl,byte ptr [eax+4]	y CL
shl	ecx,10	desplaza ECX 16bits a la izquierda, quedando el valor anterior de CX en la parte alta de ECX y CX a 0
mov	cx,word ptr [eax+2]	lee resto de la base de la LDT en el word bajo de ECX
lea	edi,[ecx+8]	suma a la base de la LDT 8, esto lo hace para modificar el segundo descriptor de segmento de la LDT tal y como sigue
cld		borra flan de dirección (por seguridad, para que se copia hacia adelante)
mov	eax,esi	copia dirección del código a ejecutar en Ring 0
stosw		y guarda la parte baja de esa dirección, en el campo limite del descriptor
mov	eax,ec000028	Establece los atributos en 0xEC00 y el selector en 0x28
stosd		lo guarda
shld	eax,esi,10	desplaza parta baja de eax a la parte alta y llena la parte baja con los bits mas significativos de ESI
stosw		y guarda parta baja de EAX,esto pertenece a la parte alta de la dirección a la que saltar en Ring 0 + los atributos + los derechos de acceso
popad		restaura los registros
call	000f:00000000	y salta a la dirección de código a ejecutar en Ring 0, el selector es 0xF, ya que se modificó el segundo selector de la LDT, para volver de esta llamada se debe usar la instrucción RETF ya que es una llamada lejana la que se realiza, en la rutina también deben salvarse los selectores actuales de la CPU para ello usamos PUSHAD y POPAD

Se obtiene el registro GDTR mediante la instrucción SGDT para la tarea actual, se obtiene el registro LDTR mediante la instrucción sldt, se enmascara para obtener el índice a la GDT y se calcula la dirección del descriptor de segmento en EDI. Se modifican los datos del descriptor para que apunten a la dirección de código almacenada en ESI esta dirección es una dirección dentro de la aplicación en Ring 3, que nos servirá de puerta de entrada al código en Ring 0, y también se modifican sus atributos y derechos de acceso.

Por último se llama a la puerta mediante CALL 000F:00000000, si entramos dentro de la llamada el código siguiente se estará ejecutando en Ring 0, y tal y como yo esperaba es el que se encarga de llamar a Get_DDB y Test_Debug_Installed.

Veamos ahora lo que son las tablas de descriptores globales y locales.

Al entrar en modo protegido, deben estar residiendo en memoria principal las tablas de descriptores, que contienen las referencias precisas para los segmentos que va a usar el procesador. Un sistema multitarea se compone de un área global, en la que residen todos los objetos comunes a todas las tareas, y un área local para cada tarea, con los segmentos propios de cada una. Cada segmento del área global está definido por un descriptor, existiendo una tabla llamada, tabla de descriptores globales o tabla global de descriptores (GDT), que contiene todos los descriptores del área global.

Asimismo existe una tabla para cada tarea, que recoge todos los descriptores de los segmentos de cada una de ellas. Se trata de las tablas de descriptores locales (ldt) o tablas locales de descriptores. Existirán tantas LDT como tareas soporte el sistema. En un momento determinado el 386+ estará ejecutando una tarea concreta y tendrá activas la GDT y la LDT correspondiente a la tarea en curso.

Dos registros internos de la CPU, manejados por el programador de sistemas, apuntan a la base de la GDT y a la base de la LDT activa, denominandose GDTR y LDTR, respectivamente. La GDT y la LDT actúan como segmentos del sistema y sólo son accesibles por el sistema de explotación. La estructura interna de una tabla de descriptores se muestra en la siguiente tabla y puede contener un máximo de 8K descriptores de ocho bytes cada uno. La LDT es una tabla local propia de la tarea en curso y una conmutación de tarea provocará automáticamente el cambio de la LDT a través de la modificación del valor en el registro LDTR.

+N x 8	DESCRIPTOR N
...	...
+16	DESCRIPTOR 2
+8	DESCRIPTOR 1
0	DESCRIPTOR 0

Nótese que los registros GDTR y LDTR apuntan al descriptor 0 de sus respectivas tablas.

Y por último la estructura de los registros GDTR y LDTR.

```
// estructura para GDTR
typedef struct
{
word limit; // tamaño de la tabla
dword base; // dirección base del descriptor 0
}FPWORD;
// estructura para el registro LDTR
word selector; // 16 bits que actúan como un selector de un descriptor de la GDT.
```

Un selector se define de la siguiente manera:

(15-3)	(2)	(1-0)
INDICE	TI	RPL

Donde:
INDICE: Apunta a una de las entradas a la tabla de descriptores seleccionada con TI[56].
Este bit es el indicador de tabla, cuando TI = 1, se selecciona la LDTn, mientras que, si TI = 0, se hace referencia a la GDT.

[56] Table Indicador = Tabla Indicadora

RPL Este campo es el valor del nivel de privilegio del segmento, es decir, es el nivel de privilegio del peticionario (PL). Puede ser 0, 00 en binario o 3, 11 en binario.

Comprobemos ahora el valor del segmento del código anterior. Se establece el selector a 0x0028, que en binario es 101000, vemos pues que los bits 1 a 0 son 0b petición para un nivel de privilegio 0, y el bit TI es 0b también, por lo que estamos haciendo referencia a la GDT. Por último los bits 15 a 3 valen 101b que en decimal es 5, por lo tanto se hace referencia al 5 descriptor de la GDT, la dirección de este descriptor seria (GDTR BASE + (5 * 8). Según los libros que he podido consultar un descriptor de segmento es una estructura de datos formada por 8 bytes, estos bytes contienen los parámetros que definen completamente el segmento referenciado, es decir: la base, el límite, los derechos de acceso y sus atributos. Como se muestra en la tabla siguiente:

BITS	31 -24	23-20				19-16	15-7					7-0
DIRECCIÓN N+4	BASE bits (31-24)	G 1 bit	D/B 1 bit	1 bit	AVL 1 bit	LIMITE (bits 19-16)	P 1 bit	DPL 2 bits	S 1 bit	TIPO 3 bits	A 1 bit	BASE (23-16)
DIRECCIÓN N	BASE (15-0)						LIMITE (15-0)					

Según esto podríamos definir la estructura de un descriptor como lo siguiente:

```
// Estructura de un descriptor
typedef struct {
WORD limit_low;
WORD base_low;
BYTE base_m;
BYTE access;
BYTE limit_high;
BYTE base_high;
} Descriptor;
```

Pero por alguna razón que desconozco, a la hora de la verdad, en el código se usa la siguiente estructura

```
//compuertas del 386+
typedef struct {
WORD offs_low; // word alto con la dirección de la memoria a la que apunta el descriptor/compuerta
WORD selector; // selector
WORD attributes; // atributos
WORD offs_high; /// word bajo con la dirección de la memoria a la que apunta el descriptor/compuerta
}Compuerta;
```

El tamaño es el mismo pero no así el asignamiento de bytes de cada campo del descriptor o compuerta.
BASE: Campo de 32bits que contiene la dirección lineal donde comienza el segmento.
LÍMITE: Campo de 20 bits que expresa el tamaño del segmento. Ya que con 20 bits el tamaño máximo es de 1MB, hay otro bit complementario en el campo de atributos, llamado de granularidad, G, que indica si el límite está expresado en

bytes (G=0) o en páginas (G=1). Si fuera en páginas el tamaño máximo del segmento sería de 1M x 4Kb = 4GB.

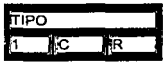
Es un campo de 4 bits de los cuales uno de ellos debe estar a 0 para mantener la compatibilidad con los procesadores superiores como los 486 y los Pentium.
G-> granularidad: Los 20 bits del campo límite del descriptor indican el tamaño del segmento, que estará expresado en bytes si G = 0, y en páginas si G = 1.
D/B -> defecto o grande: En los segmentos de código el bit D, y en los segmentos de datos este mismo bit llamado B, permite distinguir los segmentos nativos de 32bits para el 386+, de los que pertenecen al 286. Así se mantiene una compatibilidad total con el software creado para el 80286, sin penalizar las instrucciones que aporta el 386+.

AVL-> DISPONIBLE: Este bit esta a disposición del usuario para poder diferenciar ciertos segmentos que contengan un tipo determinado de información o que cubran alguna función específica.

En rojo los derechos de acceso:

A-> ACCEDIDO: Este bit se pone a 1 cada vez que el procesador accede al segmento.
P -> BIT DE PRESENCIA: Indica si el segmento al que referencia el descriptor está cargado, o sea, se halla presente en la memoria principal (P=1), o bien, está ausente (P=0).
DPL -> NIVEL DE PRIVILEGIO: Indica el nivel de privilegio del segmento al que se referencia el descriptor. Su valor puede variar entre el 0 y el 3 y consta de dos bits.
S -> TIPO DE SEGMENTO: Si S=1, el segmento correspondiente al selector es normal, o sea, un segmento de código, de datos o de pila. Si S=0, se refiere a un segmento del sistema, que referencia a un recurso especial del sistema, como puede ser una puerta de llamada, un segmento TSS, etc.
TIPO: Los tres bits de este campo distinguen en los segmentos normales si se trata de uno de código, da datos o de pila. Además determinan el acceso permitido: lectura, escritura, ejecución.

Si el bit E del campo tipo es 1, TIPO se define como:



Donde:
C -> AJUSTABLE: Si C = 0, al ser accedido el segmento no cambia su nivel de privilegio. Si C = 1, se llama segmento ajustable, porque, cuando se accede a él, su nivel de privilegio toma el valor del que tiene el segmento que lo ha pedido.

R -> LEÍBLE: Si R=1 el segmento de código se puede leer. En ningún caso se puede escribir un segmento de código.
Cuando E = 0 y se hace referencia a un segmento de datos, los otros dos bits de TIPO tiene el siguiente significado

Donde:
ED-> EXPANSIÓN DECRECIENTE: Si ED = 0, se trata de un segmento de datos normal, lo que supone que el crecimiento del mismo se realiza incrementando el valor de la dirección. Cuando ED = 1, se trata de un segmento de pila pues su crecimiento se efectúa decrementando el valor de la dirección que apunta a su cima.

W -> ESCRIBIBLE: Si W=1 el segmento de datos se puede leer y escribir, mientras que, si W = 0, sólo se puede leer.
Tal y como podemos ver en la estructura el parámetro más interesante es el de los atributos en rojo, en el programa en cuestión se establecen los privilegios a 0xEC00, veamos que significa esto.
0xEC00 en binario -> 1110110000000000, si comprobamos cada campo de los atributos.
P bit 15 -> Está a 1, por lo tanto el segmento al que hace referencia el descriptor está cargado, esto es lógico ya que el código que ejecutará la aplicación se encuentra en un segmento ya cargado, el de la aplicación en Ring 3.

DPL (bits 14-13) -> Indica el nivel de privilegio, vale 11, que es 3, por lo tanto Ring3.
S bit 12 -> Esta a 0, el segmento correspondiente al selector es un segmento de sistema que pertenece a un recurso especial como una puerta de llamada, correcto, es lo que buscábamos.

TIPO bits 11- 9 -> Bit 11 a 1, indica que se hace referencia a un segmento de código, los bits 10 y 9 indican que el segmento es ajustable adoptará el nivel de privilegio del segmento que lo solicitó, que tal y como vimos era 0x0028 y tenia un RPL de 0, y que no es legible respectivamente.

Visto el funcionamiento de los selectores, descriptores, GDT y LDT, podemos comprender el funcionamiento del código usado por el programa para entrar en Ring0. Otro punto a tener en cuenta es que deberíamos hacer una copia de el descriptor antiguo antes de modificarlo para luego dejarlo tal y como estaba todo, he realizado diversas pruebas y si no restauráramos el descriptor no sucede nada ya que solo se salta a la dirección que marca el descriptor cuando se accede directamente a ella, pero es de buena educación dejar las cosas como estaban. Por ultimo aquí debes tener el código fuente en C que muestra como saltar a

Ring0, y llamar a la función del VXD VMM Test_Debug_Installed, y si se encuentra debugger se cuelga la máquina intencionadamente llamando a la INT 19.

La INT 19, reinicia el sistema sin limpiar la memoria y sin restaurar los vectores de interrupción. Debido a que los vectores son preservados, esta interrupción causará un cuelgue del sistema si cualquier programa captura alguno de los vectores entre el 00 y el 1Ch, particularmente la INT 8. Este cuelgue sucederá siempre bajo Windows ya que estos vectores están modificados por Windows.

Desafortunadamente esta técnica no funciona bajo Windows NT y el sistema generará un error de aplicación, de todas maneras yo lo he probado bajo Windows 98 y Windows Me, y funciona de maravilla, con lo que deduzco que probablemente funcionará con Windows 95.

```
#include <stdio.h>
#include <windows.h>
char caption[]= {"Test Debug"};
// Alinea las estructuras a BYTE
#pragma pack(1)
// estructura para la base de la GDT
typedef struct
{
WORD limit;
DWORD base;
}FPWORD;
// estructura de un descriptor
typedef struct {
WORD limit_low;
WORD base_low;
BYTE base_m;
BYTE access;
BYTE limit_high;
BYTE base_high;
}Descriptor;
// Estructura de un salto lejano, para las CALLGATES
typedef struct {
DWORD offset32;
WORD seg;
}FARJMP;

//compuertas del 386+
typedef struct {

WORD offs_low;
WORD selector;
WORD attrib;
WORD offs_high;

}Compuerta;
__declspec( naked ) void MyProc()
{
_asm {
xor eax,eax
int 0x20 // Esto ejecuta la llamada VXD Test_Debug_Installed, las VxdCalls se utilizan a traves
_emit 0xc1 // de la INT 20, tras la interrupción siguen dos WORDS, el primer identifica el número
_emit 0x00 // de servicio y el siguiente el identificador VXD, en este caso se usa el servicio
_emit 0x01 // 0x00C1 del VXD VMM, para obtener una descripción de los servicio y sus identificadores
_emit 0x00 // mirate la lista de interrupciones de RalfBrown.
jz NoDebug // hay debugger ?
int 0x19 // si, colgamos la máquina
NoDebug:
retf // no, salimos
```

```
}
}
__declspec( naked ) void TestDebugVXD(void * addr)
{
DWORD GateAddr; // dirección de la puerta
Compuerta OldGate; // estrutura donde copiar valores actuales
FPWORD GDT;
FARJMP CallGate;
_asm
{
push ebp
mov ebp, esp
sub esp, __LOCAL_SIZE // con este definición el compilador calcula automáticamente el tamaño de pila necesario
para las variables locales que usa la función
}
_asm {
pushad // guarda el estado de los registros antes de ejecutar nada
mov ebx,[addr]
mov esi,ebx // copia dirección de la función a ejecutar en Ring 0
sgdt fword ptr GDT
mov ebx,GDT.base
xor eax,eax // borra EAX
sidt ax // y carga el registro de la Tabla Local de Descriptores en AX (LDT)
and al,0xf8 // borra nibble alto y ultimo bit del nibble bajo
add eax,ebx // suma la base de la GDT al valor calculado de la LDT
mov ch,[eax+7] // usa resultado como indice para llenar CH
mov cl,[eax+4] // y CL
shl ecx,0x10 // lo desplaza 16bits a la izquierda
mov cx,[eax+2] // carga parte baja de la base, con lo anterior se obtiene la base de la LDT en ECX
lea edi,[ecx+8] // y carga en EDI la base calculada más 8
mov dword ptr [GateAddr],edi // hace una copia de la dirección de la compuerta
cld // borra flag de dirección para establecer
mov cx,word ptr [edi] // copia antigua parte baja
mov [OldGate.offs_low],cx // del desplazamiento de la compuerta
mov eax,esi // guarda parte baja de la dirección de la función que se ejecutará en ring 0
stosw // como limite de segmento
mov ecx,[edi] // guarda antiguo
mov dword ptr [OldGate.offs_low+2],ecx // selector y atributos
mov eax,0xec000028 // y establece la base a 28 y los niveles de privilegio a ECh
stosd
shld eax,esi,0x10 // desplaza parte baja de eax a la parte alta y llena la parte baja con los bits mas significativos de
ESI
mov cx,word ptr [edi] // copia antigua parte alta
mov [OldGate.offs_high],cx // del desplazamiento de la compuerta
stosw // y guarda parta baja de EAX,esto pertenece a la parte alta de la dirección a la que saltar en Ring 0
mov CallGate.seg,0x0f
mov CallGate.offset32,0x0
call fword ptr CallGate
mov edi,dword ptr [GateAddr] // restaura vieja compuerta
mov ax,[OldGate.offs_low]
mov [edi],ax
stosw
mov eax,dword ptr [OldGate.offs_low+2]
stosd
mov ax,[OldGate.offs_high]
stosw
popad // restaura los registros
}
_asm {

mov esp, ebp
pop ebp
ret
}
}
void main()
{
int resp;
```

resp=MessageBox(NULL,"Esta aplicación salta a Ring 0 y comprueba la existencia de un debugger. Si existe debugger la máquina se colgará irremediablemente, caption, MB_YESNO);

```
If (resp==IDYES)
{
TestDebugVXD(&MyProc);
MessageBox(NULL,"No hay ningún debugger activo",caption,MB_OK);
}
}
```

3.12 NÚMERO LIMITADO DE EJECUCIONES (ADYNTS)

Si un programa tiene un número limitado de ejecuciones, sabremos que allí tiene un contador.

Supongamos que este contador se disminuye cada vez. Debe haber un DEC[57] cada vez que usas el programa, así que DEC deberá aparecer a menudo. Normalmente tienes un programa 32-Bit, así que investiguemos en el listado de ensamblador para dec dword ptr con grep(*). Usar grep es importante aquí porque te permite ver los sucesos múltiples en la misma situación.

Digamos que restringimos un programa a un uso limitado de 25 usos antes de que el programa se desactive. Así que lancemos el programa y observa la caja de diálogo que te informa de cuántas veces has ejecutado el programa, sólo relanza el programa unas cuantas veces más para tener una percepción de lo que está pasando.

Aproximación aquí siente cómo este código podría llevarse a cabo 19h = 25 dec. Quizá husmea con el editor HEX para algunos probables bytes, sin embargo un desensamblaje probablemente será tu mejor acercamiento.

Debes localizar fácilmente algo así dentro de W32DASM:

```
CMP BYTE PTR [004628D4],00     <--Comprueba la 1ª vez que el programa es ejecutado
JZ 0045B7D9                    <--Salta la 1ª vez ejecutada
CMP DWORD PTR [004628D8],1A    <--Comprueba las veces ejecutadas (1A = 26dec)
JGE 0045B72A                   <--Salta si eres un chico malo
```

En este esquema debes ver fácilmente nuestros 2 flags importantes, 004628D4 decide si esta es la primera ejecución del programa, donde el número de veces que el programa se ha ejecutado se marcará como 004628D8.

El programa compara el número de veces ejecutadas con 26 en decimal, un truco menor para engañar nuestra búsqueda HEX.

Debes poder ver muchas maneras de anular este esquema, podrías conformarte, por ejemplo, con aumentar 1A (26) digamos a FF (255), esto es un cambio

[57] Contador Decimal

bastante débil pero puede ayudarte totalmente a evaluar el programa, o puedes el JGE 0045B72A, eso anularía el control de 26 veces de ejecución completamente, o quizá podrías forzar el JZ 0045B7D9 a un JMP.

La mayoría de los esquemas de limite de 'runtime' son similares en funcionamiento a este y son contadores de situación normalmente bastante débiles.

Aunque a veces los programas pueden incrementar un contador escondido dentro del archivo del programa o en una DLL), debes prestar particular atención a direcciones que se usan como flags y a como éstas puede ser potencialmente malévolas y debes de estar muy seguro de verificar que eso no sea una dirección espejo que verifica el mismo flag.

3.13 CONCLUSIONES

Antes de realizar una protección estos son algunos de los consejos que debes seguir, por supuesto esta lista podría ser mucho más larga, pero son los puntos mas relevantes para tus programas.

1.- Nunca, bajo ningún concepto, utilices nombres que puedan ser sospechosos a la hora de bautizar las funciones que se encargan del sistema de protección.

Por ejemplo si tu protección se basa en un número de serié, no le pongas un nombre como IsRegNumValid, ya que esto habla por si sólo. Usa nombres que no levanten la menor sospecha.

Es increíble, que compañías de software bastante importantes, no sigan este consejo tan básico, esto es debido a que cientos de personas trabajan en un mismo proyecto y se haría un lió el sistema de protección si fuera muy complicado, por eso resulta un juego de niños desproteger sus aplicaciones más caras.

2.- No hagas que las cadenas que informan del error de protección sean cadenas constantes, ya que si lo haces así, el cracker puede encontrar las referencias a esta cadena y a partir de ahí ya es más fácil buscar el núcleo de la protección.

Para evitar esto, crea las cadenas dinámicamente, o mejor aún prueba a encriptarlas.

3.- Cuando detectes un error o violación en el sistema de protección, no informes con ningún tipo de mensaje al usuario, simplemente sal del programa o haz que el sistema se cuelgue, esto hará que el cracker tenga menos puertas de entrada.

4.- Si tu sistema utiliza la fecha del sistema para ver si el periodo de evaluación a pasado, realiza siempre varias comprobaciones al inicio, y al cabo de por ejemplo 3 días, vuelve a comprobar.

A ser posible nunca utilices las funciones del SO para obtener fechas, es mejor extraerla de los ficheros.

5.- Haz comprobaciones internas del ejecutable, esto es una suma de comprobación o verificación del código que puede ser objetivo de ataques externos, haciendo esto, podrás saber si tu aplicación ha sido modificada por algún cracker o incluso algún virus y actuar como indica el punto 3.

6.- Nunca uses una misma función para comprobar la validez de un numero de serie, mantén siempre varias copias de esta función, aunque esto signifique tener código repetido, te aconsejo que modifiques su funcionamiento aunque no su resultado, esto generará código diferente para cada función.

Elige la función de una manera aleatoria. Esto dará por obvio a muchos crackers, y aunque consigan parchear una función, deberán de parchear también el resto.

7.- Puedes añadir trucos anti-debugging, pero no te serán de mucha ayuda a no ser que sean métodos desconocidos, ya que cualquier cracker que se precie conoce los métodos más utilizados, esto sólo le retardará unos instantes.

8.- Bajo ningún concepto utilices un sistema de protección comercial, el 99% de estos sistemas ya se ha conseguido desproteger y es relativamente fácil encontrar parches genéricos.

9.- Encripta o comprime todo o parte del código del programa, esto evitará que el cracker pueda disponer de un desensamblado que corresponda con el código verdadero del programa.

No utilices compresores o encriptadores genéricos ya que en el 99% de los casos existen un descompresor o desencriptador genérico para estos compresores o desencriptadores.

Descomprime o Desencripta el código, sólo cuando hayas comprobado que no se ha violado el sistema de protección, esto forzará al cracker a tener que desproteger el sistema para poder obtener el código real.

Este sistema casi nunca se implementa por ser quizá muy complejo para la mayoría de programadores profesionales, pero resulta extremadamente efectivo contra los crackers menos avanzados.

10.- Nunca uses el registro para guardar información relevante al sistema de protección. Lo mejor es guardarlos en lugares poco comunes, por ejemplo: archivos de bases de datos de la misma aplicación.

11.- Nunca hagas que tu sistema de protección dependa exclusivamente de una DLL externa, ya que es muy fácil crackear una DLL y luego hacerla servir para todos los programas que utilicen ese tipo de protección.

Este es otro de los fallos que cometen muchos programadores.

12.- Procura que el sistema de protección no pueda burlarse con tan sólo cambiar el flujo del programa, es decir con tan sólo modificar un salto, realiza múltiples comprobaciones, haz saltos inútiles y añade código superfluo, esto creará un clima de confusión muy desagradable para el cracker.

13.- Nunca utilices una variable global para indicar si el programa esta registrado o no 0 o 1 es lo más típico ya que con tan solo cambiar este valor, el resto del sistema de protección quedara inestable.
14.- Si tu aplicación tiene alguna función recortada, no incluyas el código fuente dentro del programa demo, ni tampoco muestres esta función en ningún menú.

15.- Utiliza el propio código de tu protección como tabla para el cálculo de los números de registro, esto pueda resultar muy desagradable para los que suelen usar el SoftIce.

16.- Utiliza números de serié que contengan letras y números separados por guiones, suelen ser más difíciles de crackear que los numéricos.

17.- Cuando hagas la rutina de cálculo, hazla lo más complicada posible, esto no quiere decir que el cálculo sea bastante intenso.

Al contrario, utiliza las instrucciones XOR, AND, SHR y RCL RCR muy eficaces y difíciles de invertir.

18.- Sobre todo después de un cálculo complejo, nunca incluyas la típica comprobación final:

```
cmp reg/mem,reg/mem
jne @NoRegistrado
```

Esto es lo más sencillo de invertir, y puede sumir un buen sistema de protección en lo más ridículo.

19.- Utiliza llamadas indirectas a tus funciones de protección, punteros a funciones, guardados en tablas.

```
mov eax,dword ptr [EDI*4+32]
call [eax]
```

en C:

```
Funcion=(int *) TablaFunciones[i+32];
Funcion();
```

Esto evita que el cracker pueda encontrar las referencias a las funciones relativas al sistema de protección.

20.- Puedes ocultar los datos de la protección en archivos de sonido e imagen o de objetos 3D usando steganografía[58]. En caso de que el cracker sea capaz de deducir donde se encuentran los datos.

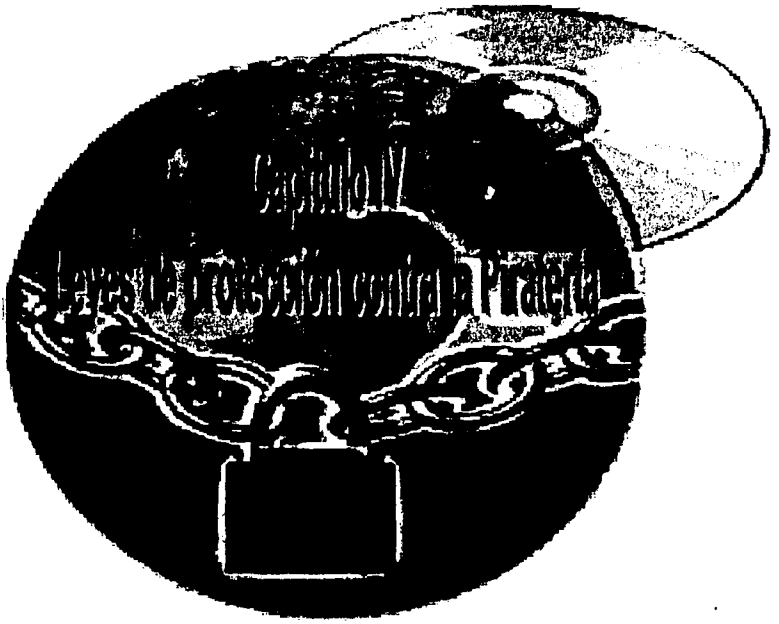
Se verá obligado a conocer la estructura interna de estos archivos y esto dificultará durante mucho tiempo su trabajo. Puedes utilizar este sistema para identificar cada copia de tu programa y así saber de donde provino la filtración. Aunque esto resulta peligroso si algún trabajador de tu empresa obtiene una copia que es para uno de tus clientes y la distribuye por ahí.

21.- Analiza el código fuente de varios virus e implementa sus técnicas de protección en tus sistemas, los virus son una fuente increíble de como poder proteger tu software.

Auto-modificación del código, generación de código aleatorio y un sin fin de trucos que solo puedes encontrar en estas maravillas de programas. Esto es para los que creen que los virus sólo sirven para fines malignos.

22.- Camufla los archivos relevantes a la protección, aunque no sirve de mucho puede despistar a más de uno.

[58] Método Sofistica de ocultación de datos.



4.1 HISTORIA

El debate sobre la protección legal que merece el software nació en el momento en el que se comenzó a distinguir del software y el hardware. Como resultado de la creciente demanda de los programas de aplicación, el software comenzó a adquirir un valor económico independiente que se fue incrementando en medida que trascurría el tiempo.

Un programa de computación puede ser considerado, por un lado como una obra científico-técnica, mientras que por el otro lado como un componente de una máquina como un programa que controla e indica que tareas realiza y en que forma.

Dada su particular naturaleza, el software siempre fue recio a dejarse encasillar en las tradicionales categorías jurídicas, siendo complicado encontrar la rama del derecho que lo proteja en su forma acabada. Es tanto que el régimen del derecho de autor como el de marcas y patentes son susceptibles de ser utilizados para proteger distintos aspectos de un mismo software, existiendo algunas zonas en las que la aplicación de los mencionados troncos se superpone, y otras que parecen estar desprotegidas.

El régimen de marcas proporciona un medio relativamente económico y eficaz de proteger al software en el ámbito internacional. Es difícil para los piratas vender ilegalmente copias de software y acceder a un publico masivo sin usar su palabra distintiva, frase, logo o símbolo gráfico utilizado para identificar la fuente del producto y distinguir al fabricante.

Las patentes, por su lado, tiene por objetivo la tutela de invenciones que arrojen como resultado un nuevo producto o procedimiento de aplicación industrial. Este régimen tradicionalmente encontró obstáculos para proteger el software dado su peculiar naturaleza.

Fue el derecho de autor el que supo brindar la protección más idónea en los últimos años en el ámbito internacional, básicamente a partir de la década de los 80's. La protección del software conforme a las leyes del copyright se debió al hecho que sus normas y principios subyacentes eran bien conocidos a escala mundial en virtud de la existencia de convenios internacionales.

Conforme al Convenio Universal o al Convenio de Berna, toda obra creada en un país es susceptible de recibir de forma automatica idéntica protección en casi todos los países del mundo. El hecho de que exista en principio un consenso internacional entre la mayoría de los países en clasificar el software como una obra literaria no hace si no justificar a prioridad la decisión de utilizar al copyright como medio principal de protección.

Al proteger sólo la expresión de un trabajo, y no la idea en si, el derecho de autor no prohíbe a terceros, a basarse en ésta última para desarrollar nuevos trabajos.

Asimismo, un producto independiente puede arribar a idéntico resultado y, en la medida en que no haya tenido acceso a la obra que resulto imitada, tendrá derecho a que la suya goce de los mis derechos de protección. Debe existir cierto grado de intención por parte del que copia para que configure la infracción.

Por otro lado técnicas como la ingeniería inversa, descompilación y desensamblaje no pueden ser evitadas satisfactoriamente con este régimen.

En definitiva la protección que otorga el derecho del autor impide la repetición pero no su uso.

La jurisprudencia no parece encontrar en el derecho de autor el marco adecuado para proteger determinados aspectos del software. En los Estados Unidos, fue paradigmático el caso de "Apple Computer Inc vs Microsoft Corp". La demandante exigía ser indemnizada por haberse sido violado sus derechos alegando que Microsoft había copiado la interfase en especial el uso de íconos de su programa.

La demanda fue rechazada. El golpe de gracia lo dio el caso de "Lotus vs Borland" en donde se decidió que el menú de instrucciones del programa de Lotus no era protegido por el régimen de copyright, dejando en una posición extremadamente vulnerable a los programas de software.

4.2 EL RÉGIMEN DE LAS PATENTES

La aplicación del régimen al software, concluye un informe del Office Technology Assesment del Congreso de los Estados Unidos de América, ha resultado en que la política es definida por los jueces, con un saldo de ambigüedad que no satisface a nadie.

Frente a semejante escenario, y dado el significativo crecimiento de la industria del software, los creadores de programas de computación o inventos que hacen uso de ellos, comenzaron a buscar un tipo de protección más fuerte:

El régimen de las patentes da respuesta a todos los desatinos presentados por el copyright. Al conferir un monopolio de explotación, habilita al inventor a oponerse a que otros no autorizados exploten su invención, aunque sean éstos productores independientes. Los problemas surgen en aquellas situaciones en las que la novedad del invento reside principalmente en el programa que lo controla.

Tradicionalmente el régimen de las patentes no fue visto con buenos ojos para proteger inventos relacionados con el software. La "European Patent Convention" prohíbe expresamente otorgar patentes a programas de computación como tales.

En Estados Unidos, si bien no se hace referencia explícita a los programas de computación, la United States Patent and Trademark Office, supo rechazar sistemáticamente las solicitudes con el fundamento de que el software consistía en algoritmos matemáticos, los cuales fueron siempre considerados como leyes

de la naturaleza o puros pensamientos, ambos excluidos del ámbito de protección del régimen de las patentes.

Sin embargo, tanto la doctrina como la jurisprudencia fueron perfilando argumentos para utilizar el régimen de las patentes. Las rígidas políticas instrumentadas tanto por la PTO como por la European Patent Office (EPO) comenzaron a distenderse, dando lugar a un criterio más flexible. La jurisprudencia no hizo más que confirmar el camino escogido.

En Europa, leyendo el caso de "Vicom/Computer-related invention" adoptó el concepto de contribución técnica del invento al arte. La Junta de Apelación sostuvo que Un invento que podría ser patentable de acuerdo con el criterio de patentabilidad convencional, no debería ser excluido de la protección por el sólo hecho de que, para su implementación, sean utilizados medios técnicamente modernos en forma de programas de computación.

Así, para que un invento cuya novedad resida en el software utilizado reciba protección debe cumplir dos requisitos. Producir alguna contribución técnica y pasar el tradicional test que se le aplican a todos los inventos. Que sea novedoso, que produzca una ventaja para la industria y que ésta sea inesperada para una persona buena en el tema.

El rumbo adoptado por Estados Unidos es similar. El emblemático caso de "Diamond vs Diehr" despejó el camino. Allí se estableció que al invento hay que verlo como un todo, sin apoyarse en el hecho de que éste funcionase en virtud de un programa para negarle la patente. En el caso de "Arrhythmia Research Technology, Inc. vs. Corazonix Corporation" se determinó que lo que prevalecía no era el proceso si no el resultado del mismo. Asimismo, en el caso de In re Alappat se concluyó que lo importante era que la máquina produzca un resultado útil, concreto y tangible.

En el caso de "Computer Associates vs Altai" directamente se sostuvo que el régimen de patentes podría ser la rama más apropiada para proteger la tecnología proveniente de las ciencias de computación. Siguiendo los lineamientos jurisprudenciales, en 1995 la PTO anunció que no rechazaría más reclamos por el sólo hecho de que este involucrado un programa de computación, si no que simplemente exigiría que cumpliesen con los requisitos tradicionales. Se estudiará únicamente si el invento es útil a las artes tecnológicas en el "mundo real", por oposición a algo que no representa más que una idea o concepto, lo cual no sería patentable.

4.3 Las Patentes e Internet

Prácticamente todo tipo de negocios que tiene lugar en Internet está siendo patentado en los Estados Unidos: desde páginas de apuestas hasta sitios de encuentros amorosos. La puerta al patentamiento de estos 'inventos' fue dejada

abierta por el famoso caso de State Street Bank & Trust Co. vs. Signature Financial.

En este caso, el tribunal determinó que un método financiero para gestionar fondos mediante un ordenador constituye un modelo de negocios patentable. Lo interesante del caso radica en el nuevo tratamiento que se le da a las excepciones que fueron utilizadas jurisprudencial mente para denegar el otorgamiento de patentes. La excepción del "algoritmo matemático" y la del "modelo de negocio". Con relación a la primer excepción la Corte sostuvo que el algoritmo matemático, entendiéndose a esto como una formula matemática usada para organizar datos y así procesarlos mediante el ordenador, no era patentable toda vez que no es más que una idea abstracta.

La cuestión no era tan clara desde el momento en que en caso en que el algoritmo tuviese una aplicación práctica, podríamos estar frente a un procedimiento patentable. En el caso State Street Bank, la computadora, a través de una serie de cálculos matemáticos, transforma datos que representan diversas cantidades de dólares en un precio final. Conforme a lo sostenido por el tribunal, ello constituye una aplicación práctica de un algoritmo matemático toda vez que produce un resultado útil, concreto y tangible: un precio final aceptado por las autoridades y utilizado en subsiguientes transacciones. Después de todo se sostuvo todo proceso, sea electrónico y químico, implica la utilización de un algoritmo en el sentido amplio del término.

Respecto a la excepción de los modelos de hacer negocios se sostuvo que la misma debía ser cuestionada ya que se basaba en un principio legal obsoleto. Tal es así que el PTO estableció removiendo uno de los ataques más poderosos que existía contra la patentabilidad del software en el texto "Examination Guidelines for Computer Related Inventions" que las solicitudes no debían ser categorizadas como modelos de negocios si no que debían ser tratadas como cualquier otro proceso. Esta fue la postura de la Corte quien estableció que bastaba con que cumpliese con los requisitos de novedad, no obviedad y aplicación industrial para ser acreedora de una patente.

Así las cosas, vemos que en Estados Unidos se han lanzado de lleno en el tanto en el patentamiento de software como en los modelos de hacer negocios en Internet, comenzando a transitar por un terreno inexplorado que todos pretenden conquistar. Las consecuencias son imprevisibles.

4.4 QUÉ ES LA PIRATERÍA

Se entiende por piratería cualquier acto por el cual se lleva a cabo una explotación de derechos de propiedad intelectual de manera ilícita, con la finalidad de eludir el cumplimiento de la ley.

Los aspectos que se ven afectados fundamentalmente por la piratería son ciertos derechos de explotación de contenido económico, que corresponden a los autores: reproducción, distribución y comunicación pública, así como aspectos del derecho moral.

Las actividades más usuales consisten en la reproducción y distribución mediante la duplicación y la comercialización de discos sin la autorización de la entidad que gestión de los derechos de autor, sin la autorización del productor discográfico propietario del master discográfico o de la entidad que gestiona sus derechos y por último, sin la autorización de la entidad que gestiona los derechos que la ley reconoce a los artistas.

4.5 EL MARCO INTERNACIONAL

Esta tendencia, que está teniendo lugar en los países desarrollados, ya ha encontrado un marco legal internacional donde recostarse. El acuerdo cubre diferentes áreas en el campo de la propiedad intelectual, tales como los derechos de autor, marcas, diseño industrial, y patentes, entre otros. Este tratado exige a los países signatarios que tengan un marco legal adecuado para el otorgamiento de patentes a todo tipo de invento, sea producto o proceso, en todos los campos de la tecnología, sin discriminación alguna, sujetos sólo al tradicional test de la novedad, inventiva y aplicabilidad industrial.

Asimismo establece que los derechos otorgados por una patente deben ser disfrutados sin importar el lugar de invención o el hecho de que el producto sea importado o producido localmente. El tratado comenzó a tener efectos en 1995 y tendrá pleno vigor en los países signatarios en el 2000 para los países más desarrollados y en el 2005 para los menos. Su falta de implementación acarreará sanciones comerciales.

4.6 QUE ES LA PROPIEDAD INTELECTUAL

Bajo ese término se agrupan distintos privilegios que se otorgan sobre bienes intangibles con valor económico. De ellos podemos destacar los de copyright y similares, que protegen de la copia no autorizada de trabajos literarios o artísticos, programas, recopilaciones de datos, diseños industriales, etc.

Las marcas, que protegen símbolos, las indicaciones geográficas, que protegen denominaciones de origen, el secreto industrial, que respalda la ocultación de información, y las patentes, que otorgan monopolio temporal sobre un invento, a cambio de desvelarlo.

Aunque la Declaración Universal de los Derechos Humanos (Art. 27) reconoce a las personas el derecho a que se protejan los intereses morales y materiales que le correspondan por razón de las producciones científicas, literarias o artísticas de que sean autores, en la práctica este derecho suele ser transferido a las empresas que emplean a los creadores o que comercializan sus creaciones.

Además muchos de los desarrollos y resultados prácticos de este derecho son discutibles por razones poderosas.

4.7 LOS DERECHOS DE COPIA (COPYRIGHT)

El copyright protege la expresión de un contenido, no el contenido en sí mismo.

Se ha desarrollado para recompensar a los autores de libros o de arte. Las obras protegidas pueden expresar ideas, conocimientos o métodos libremente utilizables, pero se prohíbe reproducirlas sin permiso, total o parcialmente, con o sin modificaciones. Esta protección es muy sencilla, ya que entra automáticamente en vigor en el momento de publicación de la obra con ámbito casi universal. Modernamente se ha extendido a los programas de computadora y a recopilaciones de datos.

Las tecnologías de la información, y en especial la red, han trastocado profundamente la economía de los productos de información, que se hoy pueden copiar y distribuir sin pérdida de calidad, a un costo muy bajo, por medio de la misma máquina donde funcionan y son útiles, sin que el poseedor del original pierda nada con ello.

Esta posibilidad de crear riqueza sin que cueste ha llevado a gran parte de la sociedad y en particular a los países pobres a duplicar programas sin pagar licencia, no existiendo una conciencia social que eso sea una mala acción como un robo.

Por otro lado los fabricantes de programas, solos o en coalición la BSA[59] presionan fuertemente para que las licencias se paguen y los gobiernos persigan lo que ellos llaman piratería.

Centrándonos en los programas de computadora, hoy día su costo puede representar una proporción muy grande del costo total de un equipo informático, llegándose en muchos casos a superarlo ampliamente, por lo que el impacto del pago de licencias en una economía débil consumidora de informática puede ser muy importante.

4.8 EL SECRETO COMERCIAL

Otro de los recursos que tienen las empresas para rentabilizar sus inversiones es el secreto comercial, protegido por las leyes de propiedad intelectual, siempre que las empresas tomen las medidas suficientes para ocultar la información que no quieren desvelar. En el caso de productos químicos o farmacéuticos que requieran aprobación gubernamental, el Estado se compromete a no desvelar los datos entregados que no sea obligatorio hacer públicos.

[59] Business Software Alliance

Una de las aplicaciones del secreto comercial más conocidas se encuentra en la industria del software, que generalmente comercializa programas compilados sin dar acceso al código fuente, para así impedir el desarrollo de programas derivados.

A primera vista parece que la protección del secreto comercial es perversa, ya que puede privar indefinidamente a la sociedad de conocimientos útiles.

Permitiendo la ingeniería inversa para desarrollar productos sustitutos, aunque la presión de las industrias ha conseguido que en muchos países ésta sea una actividad prohibida y en otros sólo esté permitido en aras de la compatibilidad.

Sea perverso o no el secreto comercial, en muchos casos es mejor que una patente, ya que da una ventaja competitiva al que pone un producto en el mercado, mientras la competencia trata de imitarlo con ingeniería inversa.

Cuanto más sofisticado sea el producto, más costará a la competencia reproducirlo, mientras que si es trivial, lo copiará rápidamente. La imitación con mejoras ha sido fundamental para el desarrollo de las que hoy son superpotencias (Estados Unidos y Japón) y es muy importante para la independencia económica de los países en desarrollo.

4.9 LAS PATENTES

La alternativa al secreto comercial es la patente. A cambio de un monopolio de 17 a 25 años y un determinado coste económico, un invento es revelado públicamente, de forma que sea fácilmente reproducible. Con ella se pretende promover la investigación privada, sin costo para el contribuyente y sin que el resultado se pierda.

El poseedor de una patente puede decidir si permite a otros utilizarla y el precio que debe pagar por ello.

La doctrina oficial de que el sistema de patentes promueve la innovación, pero cada vez más se hacen oír voces que afirman que la dificulta, ya sea porque opinan que el sistema está mal implementado, ya porque creen que es perverso en sí mismo.

Lo que es considerado un invento ha ido variando con el tiempo, existiendo grandes presiones para ampliar la cobertura del sistema, incluyendo algoritmos, modelos de negocio, sustancias naturales, genes y formas de vida, incluidas plantas y animales.

Las presiones de la Organización Mundial de la Propiedad Intelectual (OMPI o WIPO) pretenden eliminar la necesidad de que el invento tenga aplicación industrial y también rebajar los estándares de inventividad exigibles en una patente.

Estados Unidos está a la cabeza de los países con un mínimo de exigencias sobre lo que es patentable, siendo además el más competitivo para que otros países adopten sus estándares, sin acordarse que él mismo se negó a aceptar las patentes extranjeras cuando era un país subdesarrollado.

Una vez obtenida una patente, los derechos del poseedor son independientes de la calidad del invento y del esfuerzo invertido en obtenerlo. Dado el costo de mantenimiento de una patente y los costos de litigación, solamente las grandes empresas pueden mantener y mantienen una amplia cartera de patentes que la sitúan en una posición que les permite ahogar cualquier competencia.

Dada la facilidad para colocar patentes sobre soluciones triviales o de muy amplia aplicabilidad, pueden monopolizar para sí un espacio muy amplio de actividad económica.

Con patentes, muchas actividades, como la programación, se hacen extremadamente arriesgadas, ya que es muy fácil que en el desarrollo de un programa complicado se viole accidentalmente alguna patente.

Todo desarrollo técnico complejo puede convertirse en una pesadilla si para cada una de las soluciones de sus partes es necesario investigar si la solución encontrada está patentada, para intentar licenciarla o para buscar una solución alternativa.

La llamada innovación incremental, la más común, la que permitiría a una empresa de un país pobre adaptar una tecnología a las condiciones locales, puede ser bloqueada por una patente.

Cuando dos o más empresas están investigando para resolver un problema, es muy probable que lleguen a una solución casi al mismo tiempo, pero sólo una generalmente la de más recursos, logrará patentar su invento, perdiendo la otra toda posibilidad de rentabilizar su inversión.

No obstante han surgido con fuerza movimientos que promueven la información libre, entre los que destaca el de la programación libre (free software u open source).

Hoy día este movimiento ha logrado resultados notables el más conocido es Linux que permiten no sólo obtener todos los programas que se necesitan gratuitamente, si no lo que es más importante, conocimiento, que es condición necesaria para la independencia tecnológica de los pueblos, ya que los programas libres se distribuyen en fuente y cualquiera con la formación apropiada los puede leer y modificar de acuerdo con sus necesidades aunque sólo sea para adaptarlos al idioma local. A pesar de lo que dicen las corporaciones transnacionales, la libertad de información ha generado resultados inimaginables con información privatizada.

El ejemplo paradigmático es Internet, resultado de la difusión de protocolos abiertos, con una infraestructura soportada fundamentalmente por programas libres.

El software libre es hoy día la única alternativa para parar al monopolio de los programas, Microsoft, y así lo van entendiendo otras multinacionales, como IBM o Sillicon Graphics.

Y siguiendo el ejemplo van apareciendo otras iniciativas.
Sin embargo estos movimientos están siendo fuertemente amenazados por las patentes el comercio electrónico en Internet con las patentes sobre modelos de negocio, el software libre con las patentes de software, los cursos libres con las patentes sobre métodos educativos.

Pero la libertad de información no significa no regular la copia de información.

El software libre es posible apoyándose en las leyes de copyright, con las que es posible impedir la privatización de programas derivados. Los conocimientos tradicionales de los pueblos indígenas hay que protegerlos, porque si no, serán igualmente explotados en beneficio de los que los obtengan y sean capaces de sacarles dinero, sin ningún retorno.

Los creadores de información libre tienen que ser remunerados de alguna manera y es necesario buscar mecanismos para ello, en algunos casos con modelos de negocio apropiados, en otros con financiación pública o por medio de asociaciones de usuarios.

4.10 PROTECCIÓN DE LAS PÁGINAS WEB

Los elementos de capturas involuntarias o de las apropiaciones indebidas pueden ser objeto de la normativa de competencia desleal. Pero, en general, establece la posibilidad de acudir al ejercicio de acciones administrativas, civiles y penales.

La Ley de Propiedad Intelectual determina acciones y procedimientos que pueden entablarse en los supuestos de violación de los derechos de explotación y de los derechos morales así como aquellos actos de desconocimiento de los derechos de remuneración. Esta protección es ejercitable tanto si los derechos atacados corresponden al autor, a un tercero adquirente de los mismos, o a los titulares de los derechos conexos o afines.

Es necesario hacer una serie de precisiones sobre el Registro de la Propiedad Intelectual:

- ✓ Es un mecanismo administrativo para la protección de los derechos de propiedad intelectual de los autores y demás titulares sobre las creaciones originales de carácter literario, artístico o científico.

- ✓ Los derechos de propiedad intelectual no están sometidos a ninguna formalidad, por tanto no es obligatorio su inscripción en el Registro para su obtención.

- ✓ La protección ofrecida por el Registro consiste en proporcionar una prueba cualificada sobre la existencia y pertenencia de dichos derechos.

- ✓ El Registro es público, por tanto cualquier persona puede acceder al contenido de los datos que figuran en los asientos registrales.

- ✓ En el Registro pueden inscribirse los derechos sobre todas las obras, actuaciones y producciones protegidas por la ley de Propiedad Intelectual.

- ✓ Todas las creaciones literarias, artísticas y científicas, expresadas por cualquier medio o soporte, pueden acceder al Registro.

- ✓ La inscripción es eficaz desde la fecha de la presentación de la solicitud, salvo que haya defectos sustanciales en la misma, en cuyo caso, será la fecha en la que se aporte la documentación que subsane los mismos.

Por tanto, el Registro de la Propiedad Intelectual sólo da prueba de la fecha cierta y de la existencia de una obra. Esta deficiencia podría cubrirse con la utilización de otros numerosos medios de pruebas públicos, como son la elevación a público de las páginas, o la inscripción de las mismas en el Registro que se va a crear para los Prestadores de Servicios de la Sociedad de la Información, que se contempla en el Anteproyecto de ley de Servicios de la Sociedad de la Información y de Comercio Electrónico.

También existe la posibilidad de proteger los contenidos de la página web a través de las condiciones generales de las mismas y estableciendo condiciones para la obtención de copias.

Por otro lado, antes de llegar a los tribunales cabe la opción del arbitraje y la mediación en el Centro de mediación y Arbitraje de la OMPI, a nivel internacional aunque ya existen este tipo de instituciones a nivel nacional como es la Comisión Mediadora y Arbitral de la Propiedad Intelectual.

Por último, en el ámbito técnico existen numerosas y complejas soluciones para prevenir como las marcas de agua que realizan una copia única de cada documento embebiendo alguna etiqueta personalizada. De esta forma, si se encontrase una copia ilícita, se podría recuperar la etiqueta y comprobar la identidad del propietario original.

El objetivo consiste en lograr que estas etiquetas permanezcan invisibles a los ojos del pirata o que si resultan detectadas sean imposibles de eliminar sin deteriorar la calidad del objeto que las contiene.

4.11 PROTECCIÓN DE DATOS

Con motivo de la entrada en vigor de la Ley Orgánica de Protección de Datos de Carácter Personal 15/1999, de 13 de Diciembre, surgen una serie de obligaciones para aquellas empresas que posean ficheros con datos de carácter personal.

Así mismo, desde el 26 de Junio de 1999 está en vigor el Reglamento de Seguridad que desarrolla la mencionada Ley Orgánica y que establece la obligación de las empresas de poner en marcha diversas medidas destinadas a garantizar la protección de dichos datos, afectando a sistemas informáticos, archivos de soportes de almacenamiento, personal, procedimientos operativos, etc.

- ✓ **Obligaciones legales de la normativa de protección de datos**

Inscripción de los ficheros en el Registro General de la Protección de Datos. Artículo 26 LOPD. Artículos. 5 y 6 R.D 1332/1994.

Redacción del documento de seguridad. El responsable del fichero elaborará e implantará la normativa de seguridad mediante un documento de seguridad de obligado cumplimiento para el personal con acceso a los datos automatizados de carácter personal y a los sistemas de información R.D 994/1999.
Redacción de cláusulas de protección de datos. Artículo 5 LOPD.
Auditoria. Artículo 17 R.D. 994/1999.

Demás medidas de seguridad de índole técnica y organizativas necesarias para garantizar la seguridad de los datos objeto de tratamiento. Artículos 9 y 10 LOPD y R.D 994/1999. Redacción de los contratos, formularios y cláusulas necesarias para la recogida de datos, los tratamientos por terceros y las cesiones o comunicaciones de datos.

- ✓ **Niveles de seguridad**

La ley identifica tres niveles de medidas de seguridad, BÁSICO, MEDIO y ALTO, los cuales deberán ser adoptados en función de los distintos tipos de datos personales, datos de salud, ideología, religión, creencias, infracciones administrativas, de morosidad, etc.

Como se muestra en la siguiente tabla:

	NIVEL BÁSICO
TIPO DE DATOS	Nombre Apellidos Direcciones de contacto (tanto físicas como electrónicas) Teléfono (tanto fijo como móvil) Otros

MEDIDAS DE SEGURIDAD OBLIGATORIAS	Documento de seguridad Régimen de funciones y obligaciones del personal Registro de incidencias Identificación y autenticación de usuarios Control de acceso Gestión de soportes Copias de respaldo y recuperación

	NIVEL MEDIO
TIPO DE DATOS	Comisión infracciones penales Comisión infracciones administrativas Información de Hacienda Pública Información de servicios financieros
MEDIDAS DE SEGURIDAD OBLIGATORIAS	Medidas de seguridad de nivel básico Responsable de Seguridad Auditoria bianual Medidas adicionales de Identificación y autenticación de usuarios Control de acceso físico

	NIVEL ALTO
TIPO DE DATOS	Ideología Religión Creencias Origen racial Salud Vida
MEDIDAS DE SEGURIDAD OBLIGATORIAS	Medidas de seguridad de nivel básico y medio Seguridad en la distribución de soportes Registro de accesos Medidas adicionales de copias de respaldo Cifrado de telecomunicaciones

4.12 MEDIDAS DE PROTECCIÓN LEGAL CONTRA LA PIRATERÍA INFORMÁTICA

Motivados e impulsados por las empresas fabricantes de software, los distintos ordenamientos jurídicos han intentado frenar las consecuencias negativas de este extendido fenómeno así, de forma paulatina los diferentes países han ido incorporando a sus sistemas jurídicos las medidas legislativas de protección que equiparan los programas de ordenador a las creaciones literarias.

Asimismo, en su articulado se expresan medidas especiales de protección, señalando que los Estados miembros deberán adoptar medidas adecuadas contra las personas que pongan en circulación una copia de programa de computadora conociendo o pudiendo conocer su naturaleza ilegítima o que tengan con fines comerciales una copia de programa de computadora.

Ocho gobiernos europeos han llevado a cabo la transposición de esta norma comunitaria a sus correspondientes legislaciones. Los gobiernos de Alemania, Dinamarca, España, Francia, Grecia, Irlanda, Italia y Reino Unido han adaptado nuevas leyes nacionales sobre la Propiedad Intelectual que refuerzan la protección jurídica de los programas.

En España, la protección jurídica de los programas de ordenador está regulada en dos ámbitos, el civil y el penal. El primero de ellos queda regulado por el Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regulando, aclarando y armonizando las disposiciones legales vigentes sobre la materia.

Por otra parte, en los artículos 270 y siguientes del código penal de 1995 se recogen los delitos relativos a la propiedad intelectual, denotando el espíritu progresista de esta norma con la inclusión de un supuesto que sanciona, la circulación y tenencia de cualquier medio destinado a la supresión o neutralización de dispositivos utilizados para proteger los programas de computadora.

La normativa sobre esta materia se completa aludiendo a la posibilidad de enérgicas actuaciones judiciales, pudiendo autorizar registros sorpresa, acordar medidas cautelosa tales como el secuestro de medios técnicos y la clausura del centro de actividad del infractor, así como las sanciones e indemnizaciones que en cada caso correspondan de acuerdo con los daños patrimoniales y morales causados.

4.13 MEDIDAS JUDICIALES CONTRA LA PIRATERÍA INFORMÁTICA

Si finalmente existe evidencia del delito, las medidas judiciales que se pueden adoptar son:

- ✓ Solicitar al Juez un registro sorpresa de las instalaciones del presunto infractor, tanto como la vía civil, como la vía penal.
- ✓ Solicitar al Juez adoptar medidas cautelares de protección.
- ✓ Exigir indemnizaciones acordes con los daños materiales y morales causados.
- ✓ El cierre de actividad del infractor.

4.14 NIVEL DE PIRATERÍA

La piratería de software es una preocupación mayor para la industria de software mundial. Como media, de cada copia licenciada de software en uso, como poco se hace una copia no autorizada o pirateada. En algunos países se hacen hasta 99 copias no autorizadas de cada copia legal en uso.

La implicación de Microsoft en educar a los consumidores acerca de la piratería de software y trabajar para prevenir su crecimiento se ha incrementado considerablemente en los pasados dos años, de acuerdo con mantener la marcha con el creciente impacto de la piratería en la economía global.

La industria de software contribuye de varias maneras a la prosperidad económica en los Estados Unidos y en todo el mundo. Los beneficios de que disfrutamos y el potencial que esperamos alcanzar son amenazados sin embargo por la proliferación de la piratería de software.

La piratería mina el valor y la integridad del software, y la amenaza a la innovación tecnológica y la disminución de oportunidades económicas creadas por la industria de software están poniendo en peligro a los usuarios. Las economías locales y nacionales, así como los vendedores, usuarios finales, y desarrolladores de software, todos sufren de piratería.

La piratería, que es realmente una infracción sobre los derechos de propiedad intelectual de un usuario o compañía, no sólo afecta a la industria del software, si no también a las industrias de música, cine, textiles, y de accesorios.

4.15 PAÍSES DE AMÉRICA LATINA EN CAMPAÑA

Las acciones de la BSA incluyen a algunos países de América latina:

Argentina
Colombia
Chile
Ecuador
Panamá
Perú
Venezuela

Por mencionar algunos. Los acuerdos entre la Business y los gobiernos latinoamericanos evidencian la preocupación de estos últimos por la piratería, pues la protección de los productos de los fabricantes de software es un aliciente para la inversión extranjera y permite una constante transferencia de tecnología a la región.

Con el transcurso de los años, estos países han mostrado una mejor infraestructura en sus sistemas de justicia civil y penal así como de aplicación de leyes, para proteger la industria del software de quienes infringen las leyes de derechos de autor.

La piratería de software en la región ha disminuido de 80 a 60% en los últimos cuatro años.

Cada país realiza acciones encaminadas a promover el uso de software legal. En Argentina los jueces han efectuado docenas de allanamientos y ordenado el decomiso de máquinas con software ilegal. A finales de los noventa, las pérdidas originadas por la copia ilegal de programas en esta nación fueron de 122 millones de dólares y se estima que 71% de las aplicaciones en uso eran ilegales.

Cuando se comprueba que una empresa, sin importar su giro, utiliza software ilícito, debe cubrir las indemnizaciones correspondientes por la violación de los derechos de autor y propiedad intelectual de los miembros de la Business Software Alliance. Por ejemplo, los Tribunales de Justicia chilenos, mediante inspecciones sin previo aviso por sospecha de piratería de software, han logrado que los infractores paguen a los productores importantes indemnizaciones por el uso de software ilegal.

Las cifras más recientes revelan que las pérdidas en Chile superaron los 73 millones de dólares y que casi 62% de las aplicaciones en uso fueron ilegales. Este país ha realizado estrictas campañas antipiratería, entre las más destacadas está la efectuada en 1999 "80 días para dar la cara", así como el Programa de Identificación de Usuarios Ilegales. En los dos proyectos participaron la Asociación de Distribuidores de Software (ADS) de Chile y la BSA, cuyo objetivo era que las empresas y público en general revisaran y pusieran al día su situación de licenciamiento de software sin riesgos de procesos legales.

En el caso de Colombia, las cifras tampoco han sido muy alentadoras: en 1996 las pérdidas fueron mayores a 85 millones de dólares y 66% de las aplicaciones en uso eran copias ilegales.

La piratería de software es un delito grave que daña el desarrollo de estas industrias en el mundo. Si usted compra o utiliza programas copiados o falsificados, les niega a los programadores de software sus ingresos legítimos, perjudicando a todos los que participan en la industria de los programas de cómputo.

La Business Software Alliance confía en que el esfuerzo conjunto de organizaciones empresariales y de autoridades será la mejor manera de acabar con la piratería, creando conciencia de que el uso ilegal de los programas de cómputo es un delito que inhibe el desarrollo económico de los países y que dificulta llevar los beneficios de la tecnología a más personas en los hogares, las escuelas y en las empresas.

Al igual que otros países de Latinoamérica, México participa de manera activa en diversas campañas contra la piratería, ejemplo, como la tregua efectuada del 6 de marzo al 30 de abril de 2000, para que quienes tuviesen software ilegal lo regularizaran de manera voluntaria.

Pero a partir del mes de mayo la BSA, con el apoyo del Instituto Mexicano de la Propiedad Industrial y la Procuraduría General de la República, realizó una serie de operativos para detectar los principales centro de uso, venta y reproducción ilegal de programas de cómputo.

Esta acción permitirá fomentar y preservar los derechos de autor y los de propiedad intelectual, lo que producirá un impacto positivo en la economía nacional. Respecto al desarrollo económico de los países, es importante destacar que el software pirata representa una evidente evasión fiscal. El software es considerado un activo que genera ganancias pero que al no ser declarado incurre en dicho delito, lo que perjudica la economía de cada país

4.16 CONCLUSIONES

La tendencia mundial que favorece al régimen de las patentes como marco jurídico adecuado para proteger al software no se puede desconocer. La globalización del mercado exige la de las leyes.

El próximo paso será resolver todas las demandas que surjan en el ámbito internacional por violación de los derechos exclusivos del inventor, aún en los casos en que se haya desarrollado el mismo invento en forma independiente. Lejos de haberse cerrado el tema con la incorporación del software a la ley 11.723, parecería ser que el futuro trae consigo más interrogantes que respuestas.

CONCLUSIONES

Además de incrementar la comprensión total del sistema para mantenimiento y un nuevo desarrollo, la ingeniería inversa tiene otros objetivos importantes como son:

Disminuir la Complejidad. Disminución del volumen y la complejidad de los sistemas por medio de ciertos métodos, como el soporte automatizado. Los métodos y herramientas de la ingeniería inversa combinados con ambientes CASE, proveerán un camino para extraer información importante, para que los fabricantes puedan controlar el proceso y el producto en la evolución del sistema.

Generar vistas alternativas. Generación o regeneración de representaciones gráficas de otras formas, por medio de herramientas de reingeniería. Ya que las representaciones gráficas han sido aceptadas como ayudas de comprensión, sin embargo la creación y el mantenimiento lleva a un embotellamiento.

Recuperar información perdida. Recuperación de cualquier sistema existente, a partir de la recuperación del diseño, además deja obtener un manual de los sistemas para entender lo que se ha hecho o como sus programas individuales interactúan como un sistema.

Detectar efectos laterales. Encontrar ramificaciones no deseadas o efectos laterales, que aparecen en el diseño y las modificaciones sucesivas. Que impiden una adecuada ejecución del sistema.

Facilitar la Reutilización. Detectar candidatos para los componentes del software reutilizables de los sistemas presentes.

La recuperación del diseño es un subconjunto de la ingeniería inversa en la cual el campo del conocimiento, información externa, y deducción o razonamiento son adicionadas a las observaciones del sistema para identificar niveles mas altos de abstracción, además obtener estos directamente examinando el sistema por si mismo.

La recuperación del diseño recrea abstracciones diseñadas desde una combinación de código, documentación del sistema existente (si esta disponible), experiencia del personal y conocimientos en general sobre dominios de problema y aplicaciones. La recuperación del diseño debe reproducir toda la información requerida para que una persona entienda completamente lo que hace un programa, como lo hace, por que lo hace y así sucesivamente.

La recuperación del diseño debe reproducir toda la información requerida para que una persona entienda completamente lo que hace un programa, como lo hace, porque lo hace, etc. De este modo, trata con un rango mas amplio de información que encuentra en representaciones o código de ingeniería de software convencional.

La ingeniería inversa es un proceso muy importante para realizar el análisis de un sistema, que nos ayuda a detectar posibles fallas en cualquiera de las etapas de su ciclo de vida (requerimientos, diseño, implementación), para posteriormente realizar los arreglos necesarios, utilizando reestructuración y reingeniería, se espera que se aumente considerablemente el uso de la reingeniería, ya que disminuye los costos del software pues a través de la reutilización se ahorra el empezar de nuevo la creación de un producto de software.

La mayor parte de los grandes sistemas de información que están hoy funcionando en las empresas del país fueron desarrollados en los años ochenta. La irrupción de las tecnologías relacionadas con Internet, el paradigma de objetos, los componentes distribuidos y la nueva mentalidad empresarial que intenta ofrecer mejores servicios a sus clientes y durante más tiempo, han provocado que la información que permanecía en los viejos sistemas y que es totalmente aprovechable, sea objeto de diversos tratamiento para su recuperación.

Las necesidades de evolución y adaptación a los nuevos requerimientos tecnológicos y de negocio empujan al sistema a una nueva situación a la que no es posible llegar a través del mantenimiento clásico. En una primera aproximación, la solución a estos problemas se puede presentar a través de dos caminos: un nuevo desarrollo que incorpore nuevas tecnologías y funcionalidades o por medio de la aplicación de ingeniería inversa.

Actualmente, los presupuestos de las empresas no permiten desarrollos de gran amplitud en costo y tiempo, ante ésta situación, las organizaciones tienen que tomar una determinación que se resume en un cambio del sistema para adecuarlo a las nuevas necesidades. Este cambio puede concretarse en una de las opciones siguientes:

- ✓ Reingeniar el sistema
- ✓ Abandonar el sistema y sustituirlo por otro nuevo
- ✓ Optar por una solución híbrida entre las dos anteriores

Como denominador común de todas las opciones anteriores, la organización debe plantearse incluir dentro del proyecto correspondiente una buena gestión de la evolución del software que se produzca, para no volver a caer en la misma situación actual.

Para realizar mejoras de la calidad en operación, capacidad del sistema, funcionalidad, rendimiento o capacidad de evolución a bajo coste, con un plan de desarrollo corto y con bajo riesgo para el cliente.

Los criterios para su aplicación se basan tanto en técnicas heurísticas como la edad del código o el coste del personal de mantenimiento, como en modelos de coste más sofisticados. Si hubiese que reingeniar, habría que recopilar sus similitudes dentro de una misma arquitectura y tratarlos como si fueran una familia de sistemas y no aplicar soluciones aisladas a cada uno de ellos.

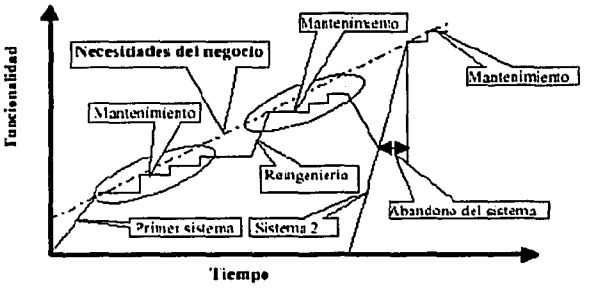
Un sistema se abandona, cuando no es capaz de seguir el ritmo de las necesidades del negocio y su ingeniería inversa, no es posible o no se puede hacer con un coste efectivo. Normalmente, estos sistemas padecen de documentación, de actualización y de capacidad de expansión.

El abandono de un sistema comporta riesgos:

- ✓ Es una actividad que consume muchos recursos y que implica a los mantenedores actuales del sistema. Estos recursos estarán ocupados en el mantenimiento del sistema y, seguramente, no conocerán las nuevas tecnologías que van a ser aplicadas en el desarrollo del nuevo sistema.

- ✓ Abandonar el sistema no significa empezar un desarrollo desde cero. No se puede pensar que los datos que contienen van a desecharse y que los valores que conservan los expertos del sistema van a perderse. El abandono del sistema debe venir precedido por una análisis exhaustivo que permitan decidir el método que se va a utilizar para la migración de la información del sistema actual al nuevo, incluyendo datos, funciones de negocio y arquitectura del sistema.

Durante el proceso de cambio del sistema hay que prever cuál va a ser la gestión de su evolución posterior, sobre todo para que la situación que se vive ahora no vuelva a repetirse o, por lo menos, sea menos traumática. La gestión de la evolución debe consistir en dar una respuesta rápida, preparada y eficiente a los cambios que se produzcan en el entorno ya sean estos de índole tecnológico o de gestión del propio negocio.
La evolución de un sistema es un concepto amplio; abarca desde una simple modificación para corregir un "bug" de un programa hasta una reimplantación completa del sistema. Las actividades de evolución que se pueden realizar sobre un sistema son de tres tipos: mantenimiento, ingeniería inversa y abandono, como se muestra en la siguiente gráfica.



Un producto software soporta una modificación en el código y su documentación asociada para la solución de un problema o por la necesidad de una mejora. Su objetivo es mejorar el software existente manteniendo su integridad.

El mantenimiento representa el porcentaje más alto del coste de todo el ciclo de vida de un sistema. Se proponen dividir en cuatro los tipos de mantenimiento:

- ✓ Adaptativo para adaptar el software a un cambio de entorno
- ✓ Perfectivo para suplir nuevos requerimientos
- ✓ Correctivo para solucionar errores puntuales de programas
- ✓ Preventivo para prever la aparición de problemas

Los dos primeros consumen el 75% de los recursos y el correctivo el 25%. Otros estudios proporcionan resultados similares. Todos muestran que, es la incorporación de nuevos requerimientos de usuario la causa principal de la evolución y el mantenimiento del software.

El problema fundamental es que, el elevado número de cambios que el software soporta a lo largo de su ciclo de vida, hace imposible una previsión en el diseño del sistema. Por tanto, el mantenimiento debe tener una consideración especial porque:

- ✓ Consume una gran parte del coste total del ciclo de vida del software
- ✓ Imposibilita un cambio rápido y fiable en el software haciendo perder oportunidades de negocio
- ✓ Dificulta la adopción de nuevas tecnologías
- ✓ Ocasiona daños en la integridad del sistema

Por ello, es previsible que la investigación en el campo del mantenimiento del software vaya en aumento en los próximos años, complementada por la ingeniería inversa de sistemas.

La redocumentación es también una forma de ingeniería inversa. Es el proceso mediante el que se produce documentación retroactivamente desde un sistema existente. Si la redocumentación toma la forma de modificación de comentarios en el código fuentes, puede ser considerada una forma suave de reestructuración. Sin embargo, puede ser considerada como una sub-área de la ingeniería inversa porque la documentación reconstruida es usada para ayudar al conocimiento del programa. Se piensa en ella como una transformación desde el código fuente a pseudo código, esta última considerada como más alto nivel de abstracción que la primera.
Aunque la aparición de multitud de herramientas facilita las labores de la ingeniería inversa, es la labor humana la decisiva a la hora de completar el estudio del sistema.

Con el paso del tiempo, se producen tres cambios en el desarrollo de sistemas que afectan a los aspectos de su evolución:

- ✓ La importancia del reflejo de la arquitectura en la documentación del sistema
- ✓ La aparición del paradigma de objetos
- ✓ Los componentes distribuidos

Aunque en un principio se pensó que el paradigma de objetos sería la solución al problema, se vio pronto que los problemas se agravaron al adaptar sistemas no concebidos para este paradigma a lenguajes como el C++ haciendo mal uso de mecanismos como encapsulación y, sobre todo, de la herencia. Al fin y al cabo, estamos ante el mismo problema de siempre: el paradigma puede tener beneficios si al desarrollarlo se aplican técnicas de ingeniería. En caso contrario, vuelven a aparecer de nuevo los problemas del mantenimiento del sistema.

En el caso de desarrollos orientados a objeto, la falta de experiencia del personal, el empleo de varios lenguajes para el desarrollo de un mismo sistema y la no incorporación de nuevas tecnologías (UML, CORBA), proporcionan la base para la construcción de sistemas con las mismas deficiencias que los actuales. Es necesario someter a técnicas de ingeniería inversa para su mantenimiento y evolución, FAMOOS, es un proyecto que recoge las premisas y particularidades de la tecnología para la ingeniería inversa orientada a objeto. En él se proponen como metas para esta ingeniería inversa la descomposición del sistema en subsistemas, el aumento del rendimiento, hacer el sistema más portable, extraer su diseño e incorporar tecnologías como UML y CORBA. Se considera que los problemas a la hora de obtener la arquitectura del sistema, se producen por una documentación insuficiente, falta de modularidad con un alto grado de acoplamiento entre clases y funcionalidad duplicada implementada con diferentes implementaciones. La falta de experiencia en la construcción de programas produce una nula o mala utilización de la herencia, operaciones que se definen fuera de la clase correspondiente, violación de la encapsulación y clases mal utilizadas (escribir C++ con estilo C). Para que ingeniería inversa orientada a objeto pueda ser aplicada ha de cumplir unos requisitos: diseño independiente del lenguaje de implementación, desarrollo escalable y herramientas que soporten las técnicas aplicadas.

La ingeniería inversa se ejecuta en seis pasos:

- ✓ Análisis de requerimientos que identifiquen las metas perseguidas por la ingeniería inversa
- ✓ Captura del modelo por medio del conocimiento y su documentación utilizando patrones de ingeniería inversa
- ✓ Detección del problema con la ayuda de herramientas de inspección, métrica y software de visualización
- ✓ Análisis del problema seleccionando la estructura que puede resolverlo
- ✓ Reorganización seleccionando la transformación óptima
- ✓ Propagación del cambio con metodología de ingeniería inversa

Para reingeniar estos sistemas se emplean técnicas métricas, de visualización de programas, de abstracción del código. Tanto para ingeniería inversa, se crean patrones para la resolución de problemas relacionados con estas técnicas.
La aparición de técnicas de modelado arquitectónico orientada a objeto y su documentación asociada (UML) facilitaron el desarrollo de técnicas de diseño con el paradigma de objetos.

Es así como se muestra un panorama general respecto a la ingeniería inversa, de éste trabajo de tesis se pueden obtener diferentes líneas de investigación que se pueden plasmar en trabajos de tesis posteriores, a su vez considero que es importante que éste tema se de a conocer a alumnos y profesores para ampliar el conocimiento en el salón de clases.

BIBLIOGRAFÍA:

Ambler, S. The Design of a Robust Persistence Layer For Relational Databases. 1999.
http://www.ambysoft.com/persistenceLayer.pdf

Arnold, R.S.; Software Reengineering, IEEE Computer Society Press, 1993.

Bennett, K.H., Rajlich, V.T., A new perspective on software evolution: the staged model, IEEE, 1999.

Bergey J., O'Brien L., Smith D., Mining Existing Assets for Software Product Lines. (CMU/SEI-2000-
TN-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Mayo 2000.

Bergey J., Smith D., Weiderman N., DoD Legacy System Migration Guidelines. (CMU/SEI-99-TN-013).
Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Septiembre, 1999.

Bergey, J.; Smith, D.; Weiderman, N.; & Woods, S.N. Options Analysis for Reengineering (OAR): Issues
and Conceptual Approach.(CMU/SEI-99-TN-014). Pittsburgh, PA: Software Engineering Institute,
Carnegie Mellon University, Octubre, 1999.

Bisdal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., O'Sullivan, D. An Overview of
Legacy Information System Migration, Proceedings of the 4 th Asian-Pacific Software Engineering and
International Computer Science Conference (APSEC 97, ICSC 97), 1997.

Brodie, M. and Stonebraker, M. Migrating Legacy Systems: Gateways, Interfaces and the Incremental
Approach. Morgan Kaufmann Publishers, 1995.

Carr, D. Web-Enabling Legacy Data When Resources Are Tight. Internet World (August 10 1998).

Chikofsky E. y Cross J., Reverse engineering and design recovery: A taxonomy. IEEE Software,
7(1):13{17, January 1990.

Comella-Dorda S., Lewis G., Place P., Plakosh D., Seacord R., Incremental Modernization of Legacy

Systems. 2001. (CMU/SEI-2001-TN-006). Pittsburgh, Pa.: SEI, Carnegie Mellon University.

Comella-Dorda S., Wallnau K., Seacord R., Robert J., A Survey of Legacy System Modernization
Approaches, Software Engineering Institute, Carnegie Mellon University, 2000, (CMU/SEI-2000-TN-003)

ESPRIT program project. The FAMOOS Object-Oriented Reengineering Handbook, 1999

Gamma et al, Design Pattern, Addison Weley, 1995

Kazman R., O'Brien L., Verhoef C., Architecture Reconstruction Guidelines. (CMU/SEI-2001-TR-026),
Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Agosto 2001

Lientz B. P., Swanson E. B. Software Maintenance Management. Addison Wesley, Reading, MA, 1980.

Oracle Corp. Oracle 8I Concepts.
http://technet.oracle.com/doc/server.805/a58227/ch_repli.htm.

Pooly R., Software Reengineering Pattern, 1998

Rodríguez, A.A., Márquez, A. Persistencia. Trabajo del curso Diseño de Software basado en métodos
formales. Mayo 2000. LSI. Facultad de informática. Sevilla.

Rumbaugh J., Blaha M.; Premerlani W.; Eddy F., Lorensen W., Modelado y diseño orientado a objetos.
Metodologóa OMT. Prentice Hall 1998

Seng, J., and Tsai, W. A Structure Transformation Approach for Legacy Information Systems- A Cash
Receipts/Reimbursement Example, Proceedings on the 32º Hawaii International Conference on System
Sciences, 1999.

Shklar, L. Web Access to Legacy Data. http://athos.rutgers.edu/~shklar/web-legacy/summary.html.

Tilley S., Storey M., Report of the STEP '97 Workshop on Net-Centric Computing. 1997. (CMU/SEI-97-
SR-016). Pittsburgh, Pa.: SEI, Carnegie Mellon University.

Tilley, Scott R. & Smith, Dennis B. Perspectives on Legacy Systems Reengineering (draft).
Reengineering Center, Software Engineering Institute, Carnegie Mellon University. 1995.

Wallnau K., Weiderman N., Northrop L., Distributed Object Technology With CORBA and Java: Key
Concepts and Implications, Software Engineering Institute, Carnegie Mellon University, 1997,
(CMU/SEI-97-TR-004).

Weiderman N., Bergey J., Smith D., Tilley S., Approaches to Legacy System Evolution, Software
Engineering Institute, Carnegie Mellon University, 1997, (CMU/SEI-97-TR-014)

Weiderman, N; Northrop, L.; Smith, D.; Tilley, S.; & Wallnau, K., Implications of Distributed Object
Technology for Reengineering ,(CMU/SEI-97-TR-005 ADA326945). Pittsburgh, Pa.: SEI, Carnegie
Mellon University.

Woods, S., Carriere, S.J., Kazman, R. A Semantic Foundation for Architectural Reengineering and
Interchange, 391-398. Proceedings of the International Conference on Software Maintenance (ICSM-99).
Oxford, England, August 30-September 3, 1999. Los Alamitos, Ca.: IEEE Computer Society, 1999.