Introducción al ensamblador en entorno Windows

Para adentrarnos en el fantástico mundo de la programación en ensamblador utilizaremos el MASM (Macro ensamblador de Microsoft). En la página de herramientas encontraréis el ensamblador y/o links a lugares donde se puede encontrar. En el paquete viene un pequeño y eficaz editor de texto (Qeditor) con opciones de linkado, así como librerías y otras herramientas que nos harán mas fácil la labor de la programación. Qeditor es el editor de archivos ASM que viene con el paquete y tiene, entre otras, la ventaja de poder configurar a nuestro gusto la ayuda, con lo que podréis incluir los HLP de las APIS de windows y todo lo que se os ocurra, en mi caso concreto tengo la ayuda del editor, la del resourcer (RCSTUDIO), las APIS de Windows, una tabla de caracteres ASCII, los opcodes del ensamblador, la ayuda del MASM y la ayuda de las librerías del MASM, de este modo lo tengo todo a mano sin tener que andar buscando algo concreto dentro de tanto archivo que hay en mi HD. Cualquiera que se decante por el Turbo Assembler, Visual Assembler u otros enlazadores de código ensamblador tendrá que modificar parte del código aquí escrito para que el el linkado no le dé problemas. En esta página vamos a hablar del MASM, no por nada en especial, simplemente es que había que elegir un ensamblador y ¿por qué no éste?

Bien, vamos a comentar línea a línea todo el contenido del PATCHER. Si alguien no sabe que es el Patcher que se pase por la sección de herramientas, es un pequeño pero eficaz programa que hemos escrito y retocado El Alfil, Marmota y Crack el Destripador (o sea, yo). A este proyecto le hemos dedicado unas cuantas horas, mucho entusiasmo y mucha colaboración, Espero que sepáis apreciar todo esto y os agradecería unas palabras de ánimo para mis colegas Crackianos que bien merecido lo tienen. Pensad que cuando se programa en cualquier entorno todo el código está siempre muy claro, el ensamblador es bastante más complejo, te puedes perder en cualquier momento y volver a pillar el hilo de lo que estabas haciendo cuesta en ocasiones tanto o más que comenzar de nuevo. Antes de comentar el código del PATCHER quiero hacer una especial y merecida mención a estos monstruos de la programación en ensamblador que tanto se lo han merecido.

GRACIAS MARMOTA POR TUS INAGOTABLES APORTACIONES

GRACIAS ALFIL POR TUS INFINITOS CONOCIMIENTOS

Preliminares:

Teniendo en cuenta que el paquete del MASM no tiene ninguna complicación con lo que respecta a su instalación nos saltaremos este paso. De momento, y hasta que el tiempo libre lo permita, iré lo más directamente al grano que pueda, poco a poco nos recrearemos dando explicaciones detalladas de absolutamente todo lo que es necesario

para poder ensamblar un pequeño fragmento de código, el resto correrá a cargo de vuestra imaginación y destreza.

El primer punto importante es aconsejaros abrir una carpeta donde guardar todos los componentes que formarán el programa a compilar, tened la precaución de no utilizar más de 8 caracteres en el nombre de las carpetas y archivos para evitar problemas con el Qeditor y los enlazadores de código, deberéis de hacer siempre copias de seguridad del último código que hayáis escrito y funcione correctamente, de lo contrario os podéis ver en una situación crítica retocando algo que después no podréis restaurar al estado en el que se encontraba antes de las modificaciones, si algo falla siempre podréis volver a utilizar el último código comprobado.

Componentes de un programa en ASM

Para realizar un programa en ASM tan solo es necesario escribir el código en un archivo de texto y darle la extensión ASM, en nuestro caso PATCHER.ASM. Si además queremos incluir un icono deberemos de colocarlo en la misma carpeta donde tengamos el ASM y escribir las líneas de código pertinentes para que éste sea compilado junto con el resto del código. Lo mismo sucede con el archivo RSRC.RC... tranquilos que vamos por partes y lo comentamos todo.

Un archivo ASM bien escrito es capaz de generar, con ayuda del compilador, código ejecutable por si solo pero tenemos la posibilidad de escribir otro archivo llamado RSRC.RC donde podemos definir qué aspecto ha de tener la ventana de diálogo principal de nuestro proyecto. Para aclarar esto nada mejor que dos ejemplos, uno de ellos constará exclusivamente de un archivo de texto con extensión ASM que será fácilmente compilable a código que Windows será capaz de interpretar y ejecutar, el otro será un proyecto de programa donde dividiremos por una parte el aspecto que ha de tener nuestro programa (en un archivo RSRC.RC) y por otra el código que se ha de ejecutar en esa ventana. Dentro de los archivos RC podemos definir el tipo de ventana que queremos que aparezca al comienzo del programa, el icono, los botones, los menús popup, Etc. Vamos a ver un par de variantes de archivos RC, uno de ellos nos dibujará un entorno de trabajo con botones, ventanas de diálogo, etc... El otro lo enfocaremos a un entorno de los de menús popup desplegables... Vamos a ver todo esto poco a poco.

Mas preliminares

Queda claro que en informática lo más importante es el orden. Si el usuario no lleva un orden dentro de su HD, está claro que perderá más tiempo buscando lo que sea que necesite que ejecutándolo, bien, pues la programación es aún más estricta en este aspecto, si no se sigue un orden riguroso y dejamos todo bien claro antes de abandonar el trabajo perderemos más tiempo averiguando dónde nos quedamos la última vez o qué error estábamos tratando de solucionar que si reprogramamos de nuevo. Para evitar esto tenemos dos armas, una es ordenar el código de forma que sea fácilmente localizable y no enredándolo dentro de un millón de instrucciones que nada tienen que ver con una rutina determianda, hay que aprender a separar el código que se ejecuta secuencial

mente del que se ejecuta más de una vez, si algún fragmento de código se ha de ejecutar más de una vez aunque sea con argumentos diferentes no es necesario rescribirlo de nuevo, lo lógico es hacer una rutina que capte los datos que varían y nos devuelva el resultado, un ejemplo sería una rutina que nos imprime un mensaje determinado siempre en el mismo lugar de la ventana, no tiene sentido rescribir el código tantas veces como necesitemos imprimir un mensaje en pantalla, lo lógico, ordenado, y fácil de seguir es hacer una rutina a la que llamamos con la cadena de caracteres que queremos hacer aparecer y llamarla tantas veces como sea necesario en lugar de repetir el código cada vez que queremos imprimir en la pantalla un mensaje. La otra arma de que disponemos es el punto y coma ";". El punto y coma nos sirve para poner comentarios en nuestro archivo ASM de tal modo que el compilador pasará por alto todo lo que se encuentre en la misma línea y a nosotros nos sirve para aclarar un poco la situación en la que nos encontramos en un momento determinado. Si además se trata de un comentario importante lo podemos resaltar con asteriscos y mayúsculas, como por ejemplo:

```
;****************          AQUÍ ME QUEDÉ ANOCHE       ***************

;**************** CONTINÚA FALLANDO EL RETORNO DE EAX ************

;**************** FALTA CORREGIR EL BUG DEL FLAG DE ACARREO **************
```

Estos comentarios nos servirán de gran ayuda a la hora de continuar programando después de un periodo de tiempo más o menos prolongado, además de aclarar el por qué hemos escrito unas líneas de código determinadas.

Bien, de momento me voy a limitar a escribir las líneas de código del PATCHER.ASM y el resourcer (RSRC.RC). Los comentarios sobre el código los iré actualizando a medida que el tiempo me lo permita. Os recuerdo que podéis bajaros el ejecutable y las fuentes AQUÍ.

Comenzaremos con el archivo resourcer RSRC.RC

----------8<----------

```
; El compilador utilizará el archivo RESOURCE.H donde están definidas una serie de variables que utilizaremos
; tales como ws_caption, ws_minimizebox Etc.

#include <\masm32\include\Resource.h>

; Identificamos cada componente de la caja de dialogo de nuestro programa.

#define IDD_DIALOG 1000
#define IDC_STATIC -1
```

```
#define IDC_EXIT 1003
#define IDC_ABOUT 1004
#define IDC_BUSCAR 1005
#define IDC_EJECUTA 1016
#define IDI_ICON1 10
#define IDC_STATUS 1002
#define IDC_FICHERO 1024
#define IDC_LISTBOX 1022
#define IDC_PARCHEAR 1023

; Nombre del icono, en tiempo de compilación será unido al ejecutable.

IDI_ICON1 ICON DISCARDABLE "ICONO.ICO"

; Aspecto de la ventana de dialogo, posición X e Y donde se mostrará, ancho y largo de la ventana
; Ventana visible y botones de cerrar, minimizar, Etc. Todo definido en resource.h


IDD_DIALOG DIALOGEX MOVEABLE IMPURE LOADONCALL
DISCARDABLE 100, 140, 307, 138
STYLE 0x0004 | WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU |
WS_VISIBLE | WS_OVERLAPPED

; definimos el título de la ventana de dialogo

CAPTION "Herramientas Crack made in Spain"

; el tipo de fuente y tamaño del texto

FONT 8, "MS Sans Serif", 700, 0 /*FALSE*/

; y los tipos de controles, aspecto, situación, Etc.

BEGIN

GROUPBOX "Parcheador Universal v1.0a", 65535, 2,7,229,89, 0, , 0
LTEXT "Archivo:", 65535, 8,25,28,9, SS_LEFT, , 0
LTEXT "Estado:", IDC_STATIC, 8,82,28,8, SS_LEFT, , 0
EDITTEXT IDC_FICHERO, 39,21,185,14, ES_READONLY | ES_LEFT, , 0
PUSHBUTTON "&Acerca de", 1004, 6,100,45,12, 0, , 0
PUSHBUTTON "&Localizar", 1016, 56,100,44,12, 0, , 0
PUSHBUTTON "&Salir", 1003, 104,100,45,12, 0, , 0
DEFPUSHBUTTON "&Buscar Archivo", 1005, 153,100,70,12, 00x00002000, , 0
ICON 10, 65535 43,116,18,20, , 0
LTEXT "Cadena de bytes a buscar", 65535, 8,46,84,12, SS_LEFT, , 0
LTEXT "Cadena de bytes a parchear", 65535, 8,60,90,8, SS_LEFT, , 0
EDITTEXT 1020, 96,43,128,13, ES_AUTOHSCROLL | ES_LEFT, , 0
EDITTEXT 1021, 96,57,128,12, ES_AUTOHSCROLL | ES_LEFT, , 0
GROUPBOX "Localizadas:", IDC_STATIC, 238,6,63,106, 0, , 0
```

```
LISTBOX IDC_LISTBOX, 242,18,54,74, LBS_DISABLENOSCROLL |
LBS_NOINTEGRALHEIGHT | LBS_SORT | WS_VSCROLL | WS_TABSTOP, , 0
PUSHBUTTON "&Parchear", IDC_PARCHEAR, 242,96,54,12, 0, , 0
EDITTEXT IDC_STATUS, 39,78,185,14, ES_AUTOHSCROLL | ES_READONLY |
ES_LEFT, , 0

END
```

----------8<----------

Código del PATCHER.ASM

----------8<----------

```
; definimos el  tipo de procesador, el modelo de memoria, y la opción casemap
(distinguir mayúsculas y minúsculas)

.486p
.model flat,stdcall
option casemap:none

; añadimos los includes (archivos donde están definidos los prototipos de funciones que
vamos a usar)

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\shell32.inc
include \masm32\include\comctl32.inc
include \masm32\include\comdlg32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\shell32.lib
includelib \masm32\lib\comctl32.lib
includelib \masm32\lib\comdlg32.lib
includelib \masm32\lib\masm32.lib

; definimos el prototipo de función de nuestro dialogo principal

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
```

```
; Bloque donde están las constantes que vamos a utilizar.
; Estas constantes tienen que tener su equivalente en el archivo rsrc.rc mediante la directiva #define

.const

IDD_DIALOG EQU 1000
IDC_STATIC EQU 1001
IDC_STATUS EQU 1002
IDC_EXIT EQU 1003
IDC_ABOUT EQU 1004
IDC_BUSCAR EQU 1005
IDC_PROGRESS EQU 1006
IDC_TIMER EQU 1008
IDC_EJECUTA EQU 1016
IDC_EDIT1 EQU 1020
IDC_EDIT2 EQU 1021
IDI_ICON1 EQU 10
IDC_LISTBOX EQU 1022
IDC_PARCHEAR EQU 1023
IDC_FICHERO EQU 1024

; Bloque correspondiente a la sección .data del ejecutable (Datos inicializados)
; Estos datos son los que suelen aparecer en las "string references" al usar un desensamblado

.data

ofn OPENFILENAME <>          ; estructura
FilterString db "Archivos ejecutables *.exe",0,"*.exe",0
db "Todos los archivos *.*",0,"*.*",0,0
About1 db "Herramienta creada por:",13,10,13,10
db "Crack el Destripador & Marmota & El Alfil",13,10,13,10
db "Mayo de 2000",13,10,13,10
db "http://welcome.to/craaaack",0
About2 db "Parcheador Universal v1.0a",0
Fich_ok db "Fichero encontrado...",0
Mem_nok db "Problema de memoria...",0
Mem_ok db "Memoria ok...",0
uncrk db "Restaurando...",0
crk db "Crackeando...",0
Hecho db "Finalizado",0
ProgressClass db "msctls_progress32",0
TimerID dd 0
pocos_bytes1 db "Al menos necesitamos 5 bytes a buscar",0
pocos_bytes2 db "La cantidad de bytes a buscar y parchear son diferentes",0
cadena_invalida db "La cadena de bytes a buscar es inválida",0
cadena_invalida1 db "La cadena de bytes a cambiar es inválida",0
falta_archivo db "Fallo al abrir el archivo o no está seleccionado",0
BUSCAR_HEX db 20 dup(0),0 ; bytes en hexadecimal
```

```
BYTES_BUSCAR db 21 dup(0),0
BYTES_CAMBIAR db 21 dup(0),0
CAMBIAR_HEX db 20 dup(0),0
CADENA_HEX dd 0
byte_ascii db "00",0
LONGITUD_BUSCAR dd 0
formato db "%.8lX",0
mensaje db "Cadena localizada en %u ocasiones.",0
salida db 12 dup(0)
no_sel db "Debe seleccionar una localización.",0
error_listbox db "Error al acceder al ListBox.",0
crk_ok db "Parchando Un momento, estoy currando... ;-)",0

; Bloque correspondiente a la sección .data? del ejecutable (Datos no inicializados)

.data?

cadena dd ?
buffer db 512 dup(?)
memptr dd ?
fhandle dd ?
fsize dd ?
hwndProgress dd ?
hwndStatus dd ?
CurrentStep dd ?
hInstance dd ?
hIcon dd ?
result dd ?
bread dd ?
bwrite dd ?
tope_buscar dd ?
otra_cadena dd ?
offset_cadena dd ?
hwndParchear dd ?
cadena_salida db 256 dup(?)

; Bloque correspondiente a la sección .code del ejecutable (Donde esta el código)

.code

; La directiva start: nos indica el EP (Entry point o Punto de Entrada)
; Es la dirección donde empieza a ejecutarse el código

start:

; Obtener manejador de nuestro modulo (hInstance).
; Esto es indispensable para usar otras APIS.

invoke GetModuleHandle,NULL
mov hInstance,eax
```

```
; Llamar al procedimiento del dialogo principal

invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL

; Cuando salgamos del procedimiento anterior, salir del programa.

invoke ExitProcess,NULL

; Esta funcion es para inicializar los controles usados por el dialogo principal

invoke InitCommonControls

; Procedimiento del dialogo principal.

;#########################################

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

; Definimos una variable local (contador)
; Este tipo de variables se almacenan en la pila

local contador: DWORD

; Un dialogo siempre esta esperando algo (una entrada de teclado, un click de ratón, etc.)
; ese algo se denomina "evento". Los diálogos procesan los eventos mediante mensajes y
; para saber que mensaje es, se crea un bucle sin fin que va comparando los mensajes que llegan al dialogo
; a ver si hay alguno que nos interesa procesar.

; Comienza el bucle de mensajes del dialogo.

; Si el mensaje es WM_CLOSE tendremos que cerrar ¿no?

.if uMsg==WM_CLOSE

    invoke EndDialog,hDlg,NULL

; Este mensaje se recibe al iniciar el dialogo y antes de que aparezca en pantalla
; Lo aprovechamos para poner el icono, crear la barra de progreso, etc.

.elseif uMsg==WM_INITDIALOG

    invoke CreateWindowEx,NULL,ADDR
ProgressClass,NULL,WS_CHILD+WS_VISIBLE,130,\
    237,320,15,hDlg,IDC_PROGRESS, hInstance,NULL

    mov hwndProgress,eax
    mov eax,50
```

```
    mov CurrentStep,eax
    shl eax,16

    invoke SendMessage,hwndProgress,PBM_SETRANGE,0,eax
    invoke SendMessage,hwndProgress,PBM_SETSTEP,1,0
    invoke LoadIcon, hInstance, IDI_ICON1
    mov hIcon, eax
    invoke SendMessage, hDlg, WM_SETICON, TRUE, hIcon

; Aquí inicializamos el control listbox, obtenemos el manejador del botón parchear y lo inhabilitamos.

    invoke SendDlgItemMessage, hDlg, IDC_LISTBOX, LB_RESETCONTENT,
NULL, NULL
    invoke GetDlgItem, hDlg, IDC_PARCHEAR
    mov hwndParchear, eax
    invoke EnableWindow, eax, FALSE

; Este mensaje es enviado por el Timer, y sirve como referencia a la barra de progreso.

.elseif uMsg==WM_TIMER

    invoke SendMessage,hwndProgress,PBM_STEPIT,0,0
    sub CurrentStep,1

    .if CurrentStep==0

        invoke KillTimer,hDlg,TimerID
        mov TimerID,0
        invoke SendMessage,hwndProgress,PBM_SETPOS,0,0
        invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR Hecho   ;poner texto en la
status
        ret

    .endif

; Aquí procesamos el mensaje WM_COMMAND, el cual es generado cuando se activa un control

.elseif uMsg==WM_COMMAND

    mov eax,wParam
    mov edx,eax
    shr edx,16

        .if dx==BN_CLICKED              ;se ha pulsado un boton del raton

            .if ax==IDC_EXIT              ;boton Salir
```

```
            invoke SendMessage,hDlg,WM_CLOSE,0,0          ;se envia el mensaje
WM_CLOSE

         .elseif ax==IDC_BUSCAR    ;boton buscar

; Rellenar la estructura OPENFILENAME, indispensable para llamar la funcion
GetOpenFileName

            mov ofn.lStructSize,SIZEOF ofn
            mov ofn.lpstrFilter, OFFSET FilterString
            mov ofn.lpstrFile, OFFSET buffer
            mov ofn.nMaxFile,512
            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 SetDlgItemText,hDlg,IDC_FICHERO,ADDR buffer    ;poner
nombre del archivo en el control
               invoke SendDlgItemMessage, hDlg, IDC_LISTBOX,
LB_RESETCONTENT, NULL, NULL
               invoke EnableWindow, hwndParchear, FALSE   ;Deshabilitar boton
               invoke GlobalFree,memptr       ;liberar memoria si antes habiamos
reservado
               invoke CloseHandle,fhandle     ;cerrar manejador

            .endif

.elseif ax==IDC_EJECUTA ;********** aquí parchea

   mov LONGITUD_BUSCAR, 0

;******** comprobamos al menos 5 bytes a buscar y maximo 10

   invoke GetDlgItemText,hDlg,IDC_EDIT1,ADDR BYTES_BUSCAR,21  ;obtener
texto del control

; Al regresar la función en eax tenemos la longitud del texto

            .if eax<10

               invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR pocos_bytes1  ;poner
texto en la statusbar
               ret

            .endif
```

```
    mov CADENA_HEX, eax

;******** Comprobamos bytes a parchear

    invoke GetDlgItemText,hDlg,IDC_EDIT2,ADDR BYTES_CAMBIAR,21
    cmp eax,CADENA_HEX   ; y la comparamos con bites _ cambiar
    je continua

    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR pocos_bytes2
    ret

Continua:

    ror eax,1 ; dividimos la cadena entre dos
    mov LONGITUD_BUSCAR, EAX ; colocamos el resultado en longitud_buscar

 ;************ Convertimos a mayusculas

    invoke CharUpper, ADDR BYTES_BUSCAR
    invoke SetDlgItemText,hDlg,IDC_EDIT1,ADDR BYTES_BUSCAR
    invoke CharUpper, ADDR BYTES_CAMBIAR
    invoke SetDlgItemText,hDlg,IDC_EDIT2,ADDR BYTES_CAMBIAR

;********** Comprobamos si es hexadecimal la cadena ascii a buscar

    xor ebx,ebx
    mov edi, offset BYTES_BUSCAR ; edi apunta a bytes_buscar en ascii
    mov esi, offset BUSCAR_HEX ; esi apunta a buscar_hex en hexadecimal

 Comenzamos:

     xor eax,eax
    mov al, byte ptr [edi]

        .if ( al>="0" && al<="9") || (al>="A" && al<="F")

            jmp byte_bueno

        .endif

    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR cadena_invalida
    ret

byte_bueno:

    mov byte_ascii, al
    inc edi
    mov al, byte ptr [edi]
    mov byte_ascii+1, al
    inc edi
```

```
    inc ebx
    push ebx
    push ecx
    push edi
    push edx
    push esi

    invoke htodw, ADDR byte_ascii  ;Esta función convierte una cadena hex en una
dword

    pop esi
    pop edx
    pop edi
    pop ecx
    pop ebx
    mov byte ptr [esi], al
    inc esi
    cmp ebx,LONGITUD_BUSCAR ;bug corregido
    jb comenzamos

    xor ebx,ebx
    mov edi, offset BYTES_CAMBIAR ; edi apunta a bytes_buscar en ascii
    mov esi, offset CAMBIAR_HEX ; esi apunta a buscar_hex en hexadecimal

comenzamos1:

    xor eax,eax
    mov al, byte ptr [edi]

    .if ( al>="0" && al<="9") || (al>="A" && al<="F") ; comprueba que la cadena
introducida

        jmp byte_bueno1 ; sea una cadena hex válida

    .endif

    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR cadena_invalida1
    ret

byte_bueno1:

    mov byte_ascii, al
    inc edi
    mov al, byte ptr [edi]
    mov byte_ascii+1, al
    inc edi
    inc ebx
    push ebx
    push ecx
    push edi
```

```
    push edx
    push esi

     invoke htodw, ADDR byte_ascii

    pop esi
    pop edx
    pop edi
    pop ecx
    pop ebx
    mov byte ptr [esi], al
    inc esi
    cmp ebx,LONGITUD_BUSCAR ; otro bug corregido
    jb comenzamos1

; Ya hemos comprobado que las cadenas introducidas son validas
; Ahora abrimos el archivo seleccionado

    invoke CreateFile,ADDR buffer, \
    GENERIC_READ+GENERIC_WRITE, \
    NULL, \
    NULL, \
    OPEN_EXISTING, \
    FILE_ATTRIBUTE_NORMAL, \
    NULL

    mov fhandle,eax    ;guardar manejador del archivo
    invoke GetFileSize,fhandle,NULL   ;obtener longitud del archivo
    mov fsize,eax      ;lo guardamos

;******* comprobamos que el archivo se ha abierto correctamente

    .if fsize==0FFFFFFFFH         ;no se ha abierto correctamente o no esta

       invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR falta_archivo
       ret

    .endif

    mov eax, fsize
    sub eax,LONGITUD_BUSCAR      ;le restamos al archivo la cadena para no pasarse
buscando
    mov tope_buscar, eax

    invoke GlobalAlloc, NULL,fsize   ;reservamos memoria para cargar el archivo
    cmp eax,0         ;si eax =0 fallo en la reserva de memoria
    jne memoria
    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR Mem_nok
    ret
```

```
memoria:

   mov memptr,eax                    ;guardamos el puntero
   invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR Mem_ok

   invoke ReadFile,fhandle,memptr,fsize,ADDR bread,NULL   ;leer el archivo

;*************************************************************************
******

;

; BUSCAMOS LA CADENA

;

;*************************************************************************
******

; Inicio cambios.

; El Alfil 19/05/00



; La búsqueda se divide en dos bloques. En el primer bucle se recorre el fichero
; buscando un byte igual primer byte de la cadena a buscar.

   mov edi,memptr        ; edi apunta al principio del programa a parchear
   mov al, byte ptr BUSCAR_HEX     ; al contiene el primer byte de la cadena
   mov ecx, fsize        ; guardamos en ecx el nº de bytes del fichero
   sub ecx, LONGITUD_BUSCAR     ; en donde se espera encontrar el inicio de
   inc ecx           ; la cadena a buscar. Tamaño-Tamaño_cadena+1
   pusha             ; Salvo todos los registros (los salvo porque
                     ; he tenido "peguillas" si no lo hacia)

   cld     ; Limpio el flag de dirección. Me aseguro que
           ; esi y edi se autoincrementen en lugar de
           ; autodecrementarse

   mov contador, 0
   jmp busca_inicio ; Comienza la búsqueda

restaura_busca:

   pop edi ; Restauro puntero a fichero
   pop ecx ; Restauro contador de bytes restantes.
   test ecx, ecx
   jz fin_busca
```

```
busca_inicio:

    repne scasb ; Bucle. No para hasta que el byte apuntado
                ; por 'edi' sea igual a 'al' ... o hasta que
                ; 'ecx' sea cero. En cada iteración se
                ; decrementa 'ecx' y se incrementa 'edi'

    dec edi  ; Esto es para evitar que no se detecten
    scasb    ; cadenas que estén justo al final del fichero
    je inicio_encontrado
    test ecx, ecx
    jz fin_busca ; Si el contador es cero termino de buscar.
                 ; En otro caso he encontrado un posible inicio
                 ; de la cadena. Por ahora, fin_busca=salir

; Segundo bloque. Una vez localizado el primer byte de la cadena en el fichero, se
; comprueban los siguientes bytes.

    inicio_encontrado:
    push ecx ; Salvo contador en pila
    push edi ; Salvo puntero a fichero en pila
    mov esi, offset BUSCAR_HEX+1 ; Apunto al segundo byte de la cadena a buscar
    mov ecx, LONGITUD_BUSCAR ; Meto en 'ecx' el número de bytes que quedan
    dec ecx ; por verificar.
    repe cmpsb ; Bucle comparador. Compara mientras que el
               ; byte apuntado por 'esi' sea igual que el
               ; apuntado por 'edi'. En cada iteración
               ; decrementa 'ecx' e incrementa 'esi' y 'edi'.

    test ecx, ecx ; Si he encontrado la cadena, el contador debe
    jnz restaura_busca ; ser cero. Si no lo es, reinicio la búsqueda.
    dec esi ; Verifico que el último byte comparado coincide
    dec edi ; con el último de la cadena a buscar (corrige
    cmpsb ; el caso en el que solo hay diferencia en el
    jne restaura_busca ; último byte)

; Cadena encontrada. Por ahora solo se muestra en pantalla y se sale. Más adelante ya
; veremos que 'maldad' se nos ocurre... ;-)

    pop edi
    push edi
    dec edi ; Recupero la posición de memoria en donde comienza
    sub edi, memptr ; la cadena y calculo el offset en el fichero.
    mov offset_cadena, edi
    invoke wvsprintf, ADDR salida, ; Convierto a texto el desplazamiento
    ADDR formato,
    ADDR offset_cadena

    invoke SendDlgItemMessage, hDlg,
    IDC_LISTBOX,
```

```
    LB_ADDSTRING,
    0,
    ADDR salida

    inc dword ptr contador
    mov al, byte ptr BUSCAR_HEX ; Restauro el primer byte de la cadena
    jmp restaura_busca ; y sigo buscando.


; Fin cambios 19/05/00

; Inicio cambios 25/05/00

    fin_busca:
    mov ebx, contador
    invoke wsprintf, ADDR cadena_salida,
    ADDR mensaje,
    ebx

    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR cadena_salida

    test ebx, ebx
    jz salir

    invoke EnableWindow, hwndParchear, TRUE

salir:

    popa
    ret



.elseif ax==IDC_PARCHEAR ;********** aquí parchea

    pusha
    invoke SendDlgItemMessage, hDlg,
    IDC_LISTBOX,
    LB_GETCURSEL,
    0, 0

    cmp eax, LB_ERR
    jne crackea
    mov eax, offset no_sel
    jmp fin_crackea

crackea:

    invoke SendDlgItemMessage, hDlg,
    IDC_LISTBOX,
```

```
    LB_GETTEXT,
    eax,
    ADDR salida
    cmp eax, LB_ERR
    jne parchea
    mov eax, offset error_listbox
    jmp fin_crackea

parchea:

    invoke htodw, ADDR salida
    add eax, memptr ; recuperamos dirección de la cadena
    mov edi, eax
    mov esi, offset CAMBIAR_HEX ; edx apunta a cadena parche
    mov ecx, LONGITUD_BUSCAR ; limpiar contador
    rep movsb

    invoke SetTimer,hDlg,IDC_TIMER,1,NULL
    mov TimerID,eax
    invoke SetDlgItemText,hDlg,IDC_STATUS,ADDR crk

; Ahora restauramos el puntero de fichero y escribimos el archivo en el disco

    invoke SetFilePointer,fhandle,0,NULL,FILE_BEGIN
    invoke WriteFile,fhandle,memptr,fsize,ADDR bwrite,NULL
    invoke SendDlgItemMessage, hDlg,                    ;limpiar la listbox
    IDC_LISTBOX,
    LB_RESETCONTENT,
    0, 0
    invoke EnableWindow, hwndParchear, FALSE        ;deshabilitar botón
    invoke GlobalFree,memptr                        ;liberar memoria
    invoke CloseHandle,fhandle                      ;cerrar manejador

    mov eax, offset crk_ok

fin_crackea:

    invoke SetDlgItemText,hDlg,IDC_STATUS,eax
    popa
    ret

.elseif ax==IDC_ABOUT    ;boton about

    invoke MessageBox,hDlg,ADDR About1,ADDR About2,MB_OK

.endif

; Aqui finaliza el bucle de mensajes. Las especificaciones para w2k y XP dicen
; que eax debe devolver 0 si no se ha procesado ningun mensaje y 1 si es lo contrario
```

```
.else
xor eax, eax

.endif
mov eax, 1

ret

DlgFunc endp

end start
```

----------8<----------