Sesión 4

Análisis y uso del EDV de un proceso Win32

Objetivos

Entender que el espacio de direcciones de un proceso Win32 está divido en regiones de direcciones, que pueden encontrarse en diferentes estados.

Saber interrogar al sistema operativo acerca del estado de una dirección.

Aprender a reservar una región del espacio de direcciones que se encuentre libre.

Entender que las regiones reservadas no son utilizables por el proceso.

Aprender a comprometer una región de memoria.

Comprender que la memoria comprometida sí es utilizable por el proceso.

1 Conocimientos previos

Tal y como has visto en las clases de teoría, un programa Win32 cuenta con un espacio de direcciones de 2GB para ubicar todos sus elementos, entre los que se encuentran el código máquina de sus funciones y sus estructuras de datos locales y globales. Así cada instrucción máquina y cada variable del programa se ubicará en una determinada dirección dentro de los 2GB de su espacio de direcciones. Sin embargo, instrucciones y variables no son los únicos elementos que ocupan posiciones en el espacio de direcciones. Todo programa es controlado por el sistema operativo, y éste necesita utilizar una serie de estructuras de datos para controlar cada programa en ejecución. Dichas estructuras también se ubican en el espacio de direcciones de cada programa. Así mismo, hay librerías de funciones que pueden ser compartidas entre muchos programas. Se trata de las conocidas DLLs (*Dynamic Link Library*). Para que un programa pueda utilizar una DLL debe mapearla en su espacio de direcciones. La idea final es que el espacio de direcciones de un programa se encuentra fragmentado en un significativo número de regiones: unas ocupadas por las instrucciones y datos del programa, otras usadas por el sistema operativo y las DLLs, y otras se encontrarán libres.

En esta práctica comenzaremos investigando algunas regiones de un programa, comprobando el estado en el que se encuentran dichas regiones. Hay que recordar que los estados posibles de una región son tres: LIBRE, COMPROMETIDA o

RESERVADA. Después buscaremos una región libre en el espacio de direcciones y la utilizaremos para direccionar en ella nuevas páginas de información.

Desarrollo de la práctica

2 Análisis del EDV de un proceso

En esta parte de la práctica pondremos un programa en ejecución y analizaremos cómo está siendo utilizado su espacio de direcciones. Para ello hay que usar una función de la API Win32, denominada *VirtualQuery()*. Mediante esta función cualquier programa puede interrogar al sistema operativo acerca de cómo está siendo utilizado su espacio de direcciones. Sin embargo, *VirtualQuery()* es una función compleja de utilizar, así que se ha preparado otra función más sencilla para interrogar al sistema operativo. Se trata de la función *EstadoMem()*. El prototipo de esta función se muestra a continuación:

```
char *EstadoMem(void *dir);
```

Esta función recibe como parámetro la dirección sobre la que se desea información y retorna un puntero que apuntará a una de las siguientes cadenas: "FREE", "RESERVED", "COMMITED", "DESCONOCIDO". La cadena apuntada determina el estado en el que se encuentra la dirección pasada como parámetro a la función.

Así por ejemplo, si `p` es una variable de tipo puntero (que contendrá por tanto una dirección), para imprimir en la consola el estado en el que se encuentra la dirección apuntada `p` se puede ejecutar la siguiente sentencia:

```
printf("Estado direccion = %s", EstadoMem(p));
```

Recuerda que el especificador de formato `%s` en la cadena de control de *printf()* se utiliza para imprimir cadenas de caracteres. Si tienes alguna duda sobre la sentencia anterior, pregúntale a tu profesor.

El código máquina de la función *EstadoMem()* se encuentra en una librería denominada `libreria_mem.lib`. También sabes que toda librería tiene asociado un fichero de cabecera, en el que entre otras cosas se encuentran los prototipos de las funciones que forman parte de la librería. El fichero de cabecera asociado a `libreria_mem.lib` se llama `funiones_mem.h`. Ambos ficheros, `libreria_mem.lib` y `funciones_mem.h`, se encuentran en la carpeta *Archivos-de-practicas* (en el bloque 2), dentro de la carpeta de la asignatura. Enseguida utilizaremos estos archivos.

Ahora, en un programa muy simple vamos a utilizar la función *EstadoMem()* para conocer el estado de diversas direcciones del programa.

H Crea de la forma habitual el proyecto `2-4prog1`. Antes de escribir el programa, vamos a agregar al proyecto los ficheros `libreria_mem.lib` y `funiones_mem.h`, ya que ambos serán necesarios para poder utilizar la función *EstadoMem()* en el programa. Para ello, en primer lugar copia estos dos ficheros a la carpeta del proyecto. Ahora, en segundo lugar, tenemos que configurar las opciones de *linkado* (vinculación) del proyecto, para que el *linker* enlace de forma correcta el programa con `libreria_mem.lib`, que es donde se encuentra el código máquina de la función *EstadoMem()*. Para esto, pulsando con el botón derecho del

ratón sobre el proyecto, abre su *página de propiedades*. En la jerarquía de páginas de propiedades que se muestra en la parte izquierda de la ventana, selecciona

Propiedades de configuración → *Vinculador* → *General*

Debes entonces configurar la opción *Directorios de bibliotecas adicionales* para que incluya la ruta en donde se encuentra la librería `libreria_mem.lib`. Dicha ruta es la del proyecto, ya que es donde hemos ubicado la librería.

Ahora en la jerarquía de páginas de propiedades selecciona

Propiedades de configuración → *Vinculador* → *Entrada*

Debes entonces configurar la opción *Dependencias adicionales* para que incluya el nombre de la librería con la que deseamos enlazar, es decir, `libreria_mem.lib`.

Una vez que hemos configurado el proyecto de la forma apropiada, vamos a escribir el programa, que mediante la función *EstadoMem()*, nos va a permitir explorar el estado en el que se encuentra su espacio de direcciones. El esqueleto de este programa se muestra a continuación:

```
#include <stdio.h>

// Incluir fichero de cabecera para la funcion EstadoMem()
...

main()
{
     void *p;  // Para almacenar la direccion pedida por pantalla


     while(1)  // Se ejecuta simpre
     {
          // Pedir direccion por pantalla
          ...

          // Sacar por pantalla el estado de la direccion
     }
}
```

Para completar este programa debes tener en cuenta la siguiente indicación acerca del funcionamiento de la directiva `#include`: cuando el fichero que se indica a continuación de `#inlcude` se escribe entre ángulos (<>), el fichero se busca en la carpeta del sistema de desarrollo en la que se ubican los ficheros de cabecera del sistema, pero si el fichero se escribe entre comillas dobles (“”), éste es buscado en la misma carpeta en la que se encuentra el fichero fuente que hace referencia al fichero cabecera. Este es precisamente el caso de `funiones_mem.h`, que está en la misma carpeta en la que vas a crear el código fuente del programa. Esta carpeta es la del proyecto.

A continuación se muestra cómo debe ser la salida del programa en la consola para una dirección de ejemplo. La dirección sería introducida por el usuario en respuesta a un *scanf_s()* ubicado en el programa.

```
Direccion: 00401000
El estado de la direccion es: COMMITED
```

El programa tiene una estructura en bucle infinito. Sin embargo, puedes romper su ejecución pulsando `Ctrl-C`.

H Agrega al proyecto el fichero `2-4prog1.c`. Entonces, teniendo en cuenta las indicaciones anteriores, completa el listado del programa y almacénalo en este fichero. Obtén el ejecutable y prueba su correcto funcionamiento introduciendo la dirección 00401000, que debe ser una dirección comprometida porque es donde empieza el código del programa.

Ahora vas a calcular cuántas páginas ocupa la sección de código y datos del programa. Para calcular esto partimos de tres datos: 1) las secciones de código y datos se ubican en páginas consecutivas del espacio de direcciones, ubicándose la sección de código primero en el espacio de direcciones y la sección de datos después; 2) la sección de código comienza en la dirección 00401000; y 3) todas las páginas de las secciones de código y de datos deben estar en el estado COMMITED, ya que todas estas páginas representan páginas utilizadas por el programa (y sólo se pude utilizar aquellas páginas que están comprometidas). La idea ahora es interrogar al sistema operativo acerca del estado de las páginas que hay a partir de la dirección 00401000. Hay que interrogar sobre páginas consecutivas. Mientras el resultado de la consulta sea COMMITED, estaremos en una página de código o de datos (no hay forma de diferenciar la frontera entre código y datos). En el momento en que la página interrogada no esté COMMITED habremos entrado en otra región del espacio de direcciones y se habrán terminado las secciones de código y datos del programa.

En la figura 1 se muestran las direcciones de comienzo y final de un grupo de páginas situadas a partir de la dirección 00401000. Para interrogar sobre páginas consecutivas, puedes utilizar por ejemplo las direcciones de comienzo de las páginas (00401000, 00402000, etc.).

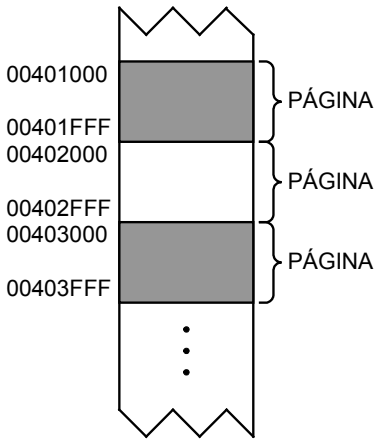

Figura 1: Direcciones de comienzo y final de un grupo de páginas

H Teniendo en cuenta toda la información anterior, ejecuta el programa `2-4prog1.exe` e interroga al sistema operativo para determinar: 1) la dirección de finalización de la región de memoria ocupada por las áreas de código y datos del programa, y 2) el número de páginas que ocupa dicha región. Escribe a continuación tus respuestas:

1)	2)

La región de pila

Ahora vamos a utilizar el programa `2-4prog1.exe` para investigar sobre la región del espacio de direcciones en la que se encuentra la pila del programa. Antes de esto comentaremos algunas cosas sobre esta región.

Previamente a la ejecución de un programa, no se sabe el tamaño que puede alcanzar su pila. Ten en cuenta que se trata de una estructura dinámica de información que se va llenando de cosas en la medida que el programa se ejecuta. Como no se conoce su tamaño a priori, el *linker* "se cura en salud", y reserva para ella una región muy grande en el espacio de direcciones del programa. El valor por defecto reservado para la pila es 1MB. Piensa que RESERVAR no cuesta nada, ya que el espacio de direcciones de un programa es enorme y sobra en él mucho sitio, y además, las direcciones reservadas no ocupan espacio físico de almacenamiento (ni en la memoria ni en el disco). Ahora bien, la memoria reservada no se puede utilizar: sólo puede usarse la memoria comprometida. Por tanto, de todas las páginas reservadas para la pila, en un momento dado algunas de ellas deben estar comprometidas para contener los parámetros y variables locales activas en ese momento. En la medida que la pila evoluciona, páginas reservadas pueden pasar a comprometidas y viceversa. La región de 1MB reservada por defecto para la pila en un programa Windows está delimitada por el intervalo de direcciones [00030000 – 0012FFFF]. Ten en cuenta que la pila crece hacia direcciones decrecientes. Por lo tanto, de la región anterior, la primera página que será comprometida para la pila es la que empieza en la dirección 0012F000.

H Teniendo en cuenta la información dada anteriormente y utilizando el programa `2-4prog1.exe`, determina: 1) región de la pila que se encuentra actualmente comprometida en el programa (debes contestar a esto con un intervalo de direcciones), y 2) número de páginas comprometidas. Escribe a continuación tus respuestas:

1)	2)

H Ahora debes comprobar que el resto de la región reservada para la pila se encuentra efectivamente reservada. Para ello puedes probar el estado en el que se encuentran las páginas que empiezan en las direcciones 000F0000, 00090000, 00060000 y 00030000.

H La página que empieza en la dirección 00030000 es la última reservada para la pila. Comprueba que la página justo anterior a ésta se encuentra libre.

3 Reservar memoria

Hasta ahora hemos analizado regiones utilizadas del espacio de direcciones de un programa. ¿Qué ocurre con las regiones libres? ¿Pueden ser utilizadas estas regiones? La respuesta es sí. Vamos a ver ahora cómo un programa puede empezar a utilizar estas regiones.

Lo primero que tenemos que hacer es localizar una región libre dentro del espacio de direcciones del programa. En los cinco primeros MB del espacio de direcciones, suele haber varias regiones reservadas, por lo que no es muy recomendable trabajar en esta zona. Nosotros trabajaremos en el MB 16 del espacio de direcciones, que se encuentra libre con toda seguridad. Este MB se ubica a partir de la dirección 01000000. La figura

2 muestra la ubicación de esta región en el espacio de direcciones, indicando sus direcciones de comienzo y finalización.

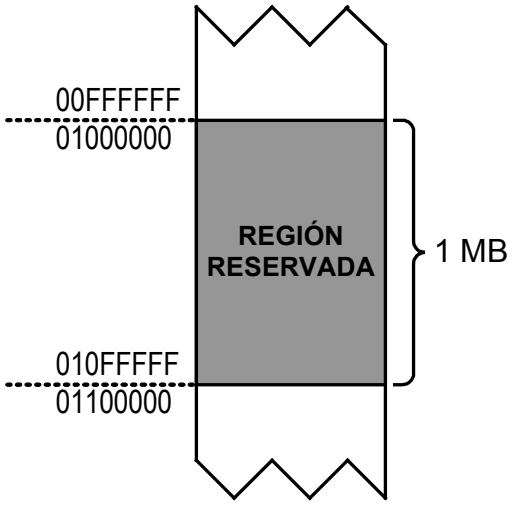

Figura 2: Región de 1 MB reservada a partir de la dirección 01000000

La idea entonces es hacer un programa que reserve un 1MB de memoria a partir de la dirección 01000000, utilizando para ello la función *VirtualAlloc()*. Para observar cómo la región cambia de estado, interrogaremos al sistema operativo sobre el estado de esta región antes y después de realizar la operación de reserva.

A continuación se muestra el esqueleto del programa a realizar perfilado mediante comentarios:

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "funciones_mem.h"

main()
{
     void *p;   // Para VirtualAlloc()
     void *q;   // Para Interrogra a la region reservada


    // Sacar por pantalla el estado inicial de la direccion 01000000
    ...

    // Pedir continuar
    printf("\n\n(Continua ...)");
    _getch();

    // Reservar region
    ...

    // Si VirtualAlloc() ha tenido éxito,
    // indicar la dirección devuelta por esta función.
    // En el caso contrario, indicar que se ha producido fallo
    // y abortar la ejecución del programa
    if (p ¡= NULL)
       ...
    else
    {
       ...
    }

    // Esta parte del programa sólo se ejecuta si VirtualAlloc()
```

```
    // ha tenido exito

    // Mensaje indicando que se pasa a interrogar a la memoria
    ...

    // Bucle infinito para interrogar el estado de la region reservada
    // (Totalmente similar al realizado en el programa 2-4prog1.c)
    ...
}
```

A continuación se indica, a modo de ejemplo, la salida que genera el programa anterior, cuando tras la operación de reserva (en la que *VirtualAlloc()* tiene éxito), se interroga al programa acerca del estado de las direcciones 01000000 y 00FFFFF. El programa realizado por el alumno tiene que generar una salida similar a ésta.

```
Estado inicial de la direccion 01000000 = FREE
(Continua ...)
Region reservada a partir de la direccion 01000000
Se pasa a interrogar el estado de la region
Direccion = 01000000
El estado de la direccion es: RESERVED
Direccion = 00FFFFFF
El estado de la direccion es: FREE
```

En cuanto a la función *VirtualAlloc()*, debes recordar que cuando su ejecución se realiza con éxito, la función devuelve la dirección de comienzo de la región reservada. Sin embargo, cuando falla, retorna NULL. En este programa deberás manejar el valor retornado por *VirtualAlloc()* para llevar a cabo diferentes acciones, en función de cuál sea el valor retornado por la función.

H Crea el proyecto `2-4prog2`. En este proyecto tendrás que hacer las mismas operaciones de configuración que en `2-4prog1`, con objeto de que se pueda utilizar en él la librería `librería_mem.lib`. Una vez configurado y copiado a su carpeta los ficheros necesarios (la librería y el fichero de cabecera), agrégale el fichero `2-4prog2.c`. Escribe en este fichero el programa anterior debidamente completado. Obtén el ejecutable del programa. Ejecuta el programa probando todas las direcciones que aparecen en la figura 2 y alguna otra dirección (a tu elección) que se encuentre situada en la zona central de la región reservada. Comprueba que el estado de todas las direcciones que están dentro de la región es RESERVED y que las direcciones 00FFFFF y 01100000 están FREE.

Fallo en la reserva de memoria

En el ejemplo anterior *VirtualAlloc()* se ha ejecutado con éxito, ya que se ha utilizado para reservar una zona de memoria libre. Ahora vas a llevar a cabo una prueba en la que *VirtualAlloc()* fallará. Para ello le pedirás a esta función que reserve una zona de memoria que ya se encuentre reservada. *VirtualAlloc()* indicará el fallo retornando el valor NULL. Para probar esto, puedes utilizar la región de pila [00030000 – 0012FFFF]. Por ejemplo, puedes pedirle a *VirtualAlloc()* que reserve 16 páginas a partir de la dirección 00050000 (dirección comprendida dentro de la región anterior). Para probar esto, utilizaremos un programa cuyo esqueleto perfilado mediante comentarios se muestra a continuación.

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

main()
{
     void *p;   // Para VirtualAlloc()

    // Reservar region
    ...

    // Si VirtualAlloc() ha tenido éxito,
    // indicar la direccion devuelta por esta funcion.
    // En el caso contrario, indicar que se ha producido fallo
    if (p!=NULL)
       ...
    else
       ...
}
```

H Crea el proyecto `2-4prog3` y agrégale el fichero `2-4prog3.c`. Entonces escribe en este fichero el listado anterior debidamente completado. Obtén el ejecutable y ejecútalo, comprobando que *VirtualAlloc()* falla.

Fallo en el acceso a una región reservada

En este ejemplo vamos a reservar una región con éxito y, después, vamos a ver que ocurre al intentar escribir o leer en esa región mediante el puntero devuelto por *VirtualAlloc()*. Para ello vas a utilizar el siguiente programa:

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

main()
{
    int *p;   // Para VirtualAlloc()
    int A;

    // Reservar region
    p=VirtualAlloc((void *)0x01000000, 0x100000, MEM_RESERVE,
                                                 PAGE_READWRITE);

    // Indicar que se ha reservado la region
    printf("\nRegion reservada a partir de la direccion = %p\n", p);

    // Intento de lectura
    ...
}
```

En este programa debes observar que el puntero `p` ha sido declarado de tipo `int`. Ten en cuenta que ahora pretendemos utilizar este puntero para acceder mediante él a la memoria, por tanto, hay que definirlo según el tipo de dato al que se desea acceder.

H Crea el proyecto `2-4prog4` y agrégale el fichero `2-4prog4.c`. Copia en este fichero el listado anterior. Escribe después del comentario \\ *Intento de lectura* una sentencia que lea de la región reservada mediante `p`, escribiendo en `A`. Compila y enlaza el programa y ejecútalo. ¿Qué es lo que ocurre? Anótalo a continuación.

H Modifica el programa anterior para escribir mediante `p` en la región reservada, en vez de leer. Compila el programa y ejecútalo observando lo que ocurre.

La conclusión que debes extraer de los experimentos anteriores es que **una región reservada no está disponible para trabajar con ella**.

4 Comprometer memoria

Ahora vamos a comprometer una página dentro de la región reservada.

H Crea un nuevo proyecto llamado `2-4prog5` y agrégale el fichero `2-4prog5.c`. Copia en él el contenido del fichero `2-4prog4.c`. Ahora realiza las siguientes modificaciones en `2-4prog5.c`.

- Elimina la declaración de la variable A. No va a ser necesaria.
- Declara otro puntero más de tipo `int`. Llámalo `q`. La idea es tener dos punteros: uno, llamado `p`, para apuntar a la región reservada, y otro, llamado `q`, para apuntar a la región comprometida dentro de la región reservada.
- Elimina la última sentencia en la que escribes o lees mediante `p` en la región reservada.
- En este punto del programa, escribe un *VirtualAlloc()* que comprometa la primera página la región reservada. Recoge la dirección devuelta por la función en el puntero `q`.
- Envía un mensaje a pantalla, indicando que se ha comprometido dicha página.
- Escribe mediante `q` el valor 27 en la página comprometida y escribe en pantalla mediante un *printf()* el contenido de la memoria apuntado por `q`.

Compila el programa, ejecútalo y comprueba que funciona correctamente.

La conclusión es que **la memoria comprometida está disponible para ser usada por el programa**.

5 Granularidad de reserva

La granularidad de reserva determina que sólo se pueden reservar regiones a partir de direcciones que sean múltiplos de 64K. Por consiguiente, cuando se le pide a *VirtualAlloc()* que reserve una región a partir de una dirección A, la dirección de reserva efectiva B sólo será igual a A, si B es múltiplo de 64K, es decir, si los cuatro últimos dígitos hexadecimales de A son 0000. En caso contrario, la dirección pasada a *VirtualAlloc()* es redondeada hacia abajo hasta obtener la dirección múltiplo de 64K más cercana a la dirección dada. Para probar esto se proporciona a continuación el esqueleto de un programa, cuyo objetivo es pedir al usuario que introduzca una dirección, reservar una página pasándole a *VirtualAlloc()* la dirección introducida y, finalmente, mostrar la dirección a partir de la que realmente se ha realizado la reserva.

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

main()
{
    void *q;   // Para recibir por pantalla la direccion
    void *p;   // Para recoger el valor devuelto por VirtualAlloc()


    // Pedir por pantalla la direccion
    (...)

    // Reservar una pagina a partir de la direccion introducida
    (...)
```

```
    // Imprimir direccion de reserva
    (...)
}
```

A modo de ejemplo, la salida que debería realizar el programa anterior cuando se le pasa la dirección 01000020 sería la siguiente:

```
Direccion: 01000020
Direccion de reserva = 01000000
```

H Crea el proyecto `2-4prog6` y agrégale el programa `2-4prog6.c`. Copia en este fichero el listado del programa anterior, completándolo con las sentencias necesarias. Obtén el ejecutable. Ahora vas a probar el programa pasándole las direcciones que se indican a continuación, pero antes piensa tú cuál sería la dirección de reserva para cada una de las direcciones pasadas a *VirutalAlloc()*. Escribe a continuación tus respuestas. Comprueba después que las respuestas son correctas ejecutando el programa `2-4prog6.exe`.

Dirección 01000020	**Dirección de reserva:**
Dirección 01000FFF	**Dirección de reserva:**
Dirección 01010FFF	**Dirección de reserva:**
Dirección 0110000A	**Dirección de reserva:**
Dirección 01201FFF	**Dirección de reserva:**

6 Ejercicios adicionales

E Tomando como base el programa almacenado en `2-4prog5.c`, vamos a realizar un conjunto de modificaciones para que se ejecute un algoritmo simple, que utilizará como zona de almacenamiento para sus datos la página comprometida. El procesamiento a realizar es el siguiente: 1) el programa pide por pantalla un número (`N`), que representa el número de elementos a procesar; 2) calcula el cuadrado de todos los números en el intervalo [0, `N`-1] y los almacena en la página comprometida; y 3) lee uno por uno los datos almacenados, representándolos en pantalla. A continuación se muestra, a modo de ejemplo, la salida realizada por este programa para `N`=6.

```
Elemento [0] = 0
Elemento [1] = 1
Elemento [2] = 4
Elemento [3] = 9
Elemento [4] = 16
Elemento [5] = 25
```

Crea el proyecto `2-4prog7` y agrégale el fichero `2-4prog7.c`. Copia en este fichero el programa almacenado en `2-4prog5.c`. Haz las modificaciones necesarias para llevar a cabo el procesamiento indicado. Compila y enlaza el programa y ejecútalo, comprobando su correcto funcionamiento.

Si tu programa está bien hecho, debe funcionar correctamente para todos los valores de `N` menores o iguales a 1024. Comprueba este hecho y contesta a las siguientes precuentas:

¿Qué ocurre cuando N es mayor de 1024?

¿Por qué el valor máximo que puede tomar N es 1024?