A **Apellidos** _____ **Nombre** _____ **DNI** _____

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2006**

A continuación se muestra el listado de un programa cuyo objetivo es encontrar una palabra dentro de una cadena de caracteres de la sección de datos y copiar dicha palabra en otra zona de la sección de datos.

La cadena en donde se encuentra la palabra a buscar está marcada con la etiqueta *cadena*. Las palabras que integran la cadena se encuentran separadas por un guión (carácter '-'), y el terminador de cadena es el número 0. Las palabras dentro la cadena ocupan una posición, siendo la primera posición la número 0. Así en la cadena del programa la palabra que ocupa la posición 0 es "En", la que ocupa la posición 1, "un", la que ocupa la posición 2, "lugar", y así sucesivamente. La zona de memoria en la que se almacenará la palabra encontrada en la cadena está marcada con la etiqueta *palabra*. Se trata de una zona de 10 bytes inicializados con el valor 0. En la sección de datos también se observa la variable *pos_palabra*. Esta variable contiene la posición de la palabra que queremos encontrar en la cadena de caracteres.

Para llevar a cabo el procesamiento requerido, el programa utiliza el procedimiento *ObtenDirPalabra*. Este procedimiento recibe dos parámetros a través de la pila: 1) la dirección de la cadena en la que se encuentra la palabra a buscar, y 2) la posición que ocupa la palabra que queremos buscar en la cadena. El procedimiento devuelve como resultado en el registro EAX la dirección de memoria en la que se encuentra la palabra buscada. Para llevar a cabo su procesamiento, el procedimiento emplea una variable local en la pila de tipo doble palabra. Dicha variable será utilizada por el procedimiento como contador de palabras.

El procedimiento *ObtenDirPalabra* funciona según se explica a continuación. El procedimiento utiliza el registro ESI para direccionar los caracteres de la cadena a procesar. El procesamiento se realiza en un bucle. En cada iteración del bucle se procesa un carácter de *cadena*. Si el carácter no es un guión, simplemente se incrementa ESI para que apunte al siguiente carácter. Si el carácter es un guión, se incrementa ESI para que apunte al siguiente carácter y se incrementa el contador de palabras (variable local). Cuando este contador alcanza el valor del segundo parámetro recibido por el procedimiento (posición de la palabra que queremos buscar), se abandona el bucle. Así el registro ESI queda apuntando al comienzo de la palabra que buscamos dentro de la cadena. Después este valor se pasa a EAX para devolver el resultado.

El programa principal llama al procedimiento *ObtenDirPalabra* indicándole que busque en *cadena* la palabra situada en la posición indicada por la variable *pos_palabra*. Mediante este procedimiento se obtiene la dirección de la palabra buscada. Después, mediante un bucle, el programa principal copia la palabra en el área de la sección de datos marcada con la etiqueta *palabra*.

Contesta a las preguntas que se proporcionan después del listado. Allí donde sea pertinente, ten en cuenta los comentarios del programa en tus respuestas.

Datos adicionales Dirección de comienzo de la sección de datos: 00402000
Valor de ESP al inicio del programa: 0012FFC4

```
.386
.MODEL flat
EXTERN ExitProcess:PROC

.DATA
cadena      DB "En-un-lugar-de-la-mancha", 0
palabra     DB 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
pos_palabra DD 5

.CODE
ObtenDirPalabra PROC
  push ebp ♠♠♠
  mov  ebp, esp
  ; Reservar espacio para la variable local contador de palabras
  (--1--)
  ; Salvar registros
  push esi
  push edx

  ; Recuperar parametros
  mov  esi, [ebp+12]
  mov  edx, [ebp+8] ◄◄◄

  ; Contador de palabras = 0
  mov  [ebp-4], DWORD PTR 0
bucle1:
  cmp  [esi], BYTE PTR 0
  je   SHORT fuera1
  cmp  [esi], BYTE PTR '-'
  je   SHORT es_guion
  ; si no es guion,
  ; es un caracter de una palabra
  inc  esi
  jmp  bucle1
es_guion:
  inc  esi
  inc  DWORD PTR [ebp-4]  ; Incrementar contador palabras
  ; Si contador palabra = posicion palabra, abandonar bucle
  cmp  [ebp-4], edx
  je   SHORT fuera1
  jmp  bucle1
fuera1:

  ; Preparar el resultado para el retorno
  mov  eax, esi
  ; Preparacion del retorno
  pop  edx
```

A

```
  pop  esi
  mov  esp, ebp
  pop  ebp
  ret  8
ObtenDirPalabra ENDP

; Programa principal
inicio:
  ; Obtener en EAX la direccion de la palabra
  ; indicada por la varible pos_palabra
  (--2--)

  xor  ebx, ebx  ; resetear auxiliar
  xor  edi, edi  ; resetear indice

  ; bucle que copia la palabra encontrada
  ; en la zona marcada con la etiqueta palabra
bucle2:
  cmp  [eax], BYTE PTR 0
  je   SHORT fuera2
  cmp  [eax], BYTE PTR '-'
  je   SHORT fuera2
  mov  bl, [eax]
  mov  [palabra+edi], bl
  (--3--)
  jmp  bucle2
fuera2:

  ; Retorno al sistema operativo
  push 0
  call ExitProcess
  nop
END inicio
```

— ¿Qué instrucción (solo una) es necesaria en el hueco (—1—) del listado? **0,5**

```
sub  esp, 4
```

— ¿Qué instrucciones son necesarias en el hueco (—2—) del listado? **0,5**

```
push OFFSET cadena
push [pos_palabra]
call ObtenDirPalabra
```

— ¿Qué instrucciones son necesarias en el hueco (—3—) del listado? **0,5**

```
inc eax
inc edi
```

— Determina el rango de direcciones que ocupa la variable *pos_palabra*. (Ejemplo de respuesta: 00402000 – 00402007) **0,5**

```
00402023 - 00402026
```

— Codifica la instrucción "`mov edx, [ebp+8]`", marcada en el listado del programa con el símbolo ◄◄◄. Indica su codificación en hexadedimal. **0,5**

```
8B 55 08
```

— Determina el valor retornado por el procedimiento *ObtenDirPalabra*, una vez que ha sido ejecutado en el programa anterior. Contesta en hexadecimal. **0,5**

```
00402012
```

— Determina el valor que toma el registro ESP justo después de que se ejecute la instrucción "`push ebp`", marcada en el listado del programa con el símbolo ♠♠♠. **0,5**

```
0012FFB4
```

— A continuación se muestra el listado de un función *main()* escrita en lenguaje C en la que se observa una sentencia de asignación marcada con el símbolo ♦♦♦. Determina el código ensamblador que generaría el compilador de C para compilar dicha sentencia. (Se entiende que puedes utilizar cualquier código que lleve a cabo de forma correcta la sentencia indicada.)

```
int A=2, B=3;

main()
{
    A = B+5; ♦♦♦
}
```

0,5

```
mov eax, [B]
add eax, 5
mov [A], eax
```

A **Apellidos** _____ **Nombre** _____ **DNI** _____

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2006**

Página Virtual (Binario)	Página Física /Disco (Binario)
00000 00000	—
00000 00001	—
00000 00010	00 11 00
00000 00011	00 11 01
00000 00100	Disco
00000 00101	Disco
01000 00000	Disco
01000 00001	Disco
01000 00010	00 01 10
01000 00011	00 01 11
11111 11110	—
11111 11111	—

La MMU de un computador funciona con direcciones virtuales de 20 bits, direcciones físicas de 16 bits y páginas de 1 KB.

En la tabla adjunta se muestra el estado de la tabla de páginas de un proceso que se encuentra en ejecución en el sistema. En esta tabla, tanto las páginas virtuales como las físicas están expresadas en binario. Las entradas de la tabla de páginas que contienen un '-', así como aquellas que no se muestran, corresponden a páginas no utilizadas por el proceso; si contienen la palabra 'Disco', significa que se encuentran en el disco; y si en ellas se encuentra un número, éste representa la página física a la que se encuentra asignada la página virtual correspondiente.

Teniendo en cuenta toda esta información, contesta a las preguntas A y B.

— A) Cuando el estado de la tabla de páginas es el que se muestra, la MMU recibe la dirección virtual 40B7A (hex.). Indica si la MMU genera una excepción de protección, una excepción de fallo de página o si genera una dirección física. En este último caso, indica la dirección física generada expresada en HEXADECIMAL. **0,5**

1B7A

— B) Partiendo del estado de la tabla de páginas mostrado en la tabla adjunta, determina la dirección virtual más significativa que al ser recibida por la MMU generaría un fallo de página. Contesta en HEXADECIMAL. **0,5**

407FF

A continuación se proporcionan el prototipo de la función *GetLocalTime()* y la definición del tipo de estructura SYSTEMTIME, tal y como se encuentra en los ficheros de cabecera del sistema de desarrollo.

```
void GetLocalTime(
  LPSYSTEMTIME lpSystemTime
);
```

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

— Teniendo en cuenta la información anterior, escribe una función *main()* que imprima en pantalla la hora del día, generando un mensaje con la siguiente forma:

```
Hora: xx
```

En donde **xx** representa la hora. Para ello utiliza la función de la API Win32 *GetLocalTime()*.

NOTA: no es necesario que inicialices con el valor 0 los campos de la estructura del tipo SYSTEMTIME que utilices en tu programa. **1**

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

main()
{

    SYSTEMTIME tiempo;

    GetLocalTime(&tiempo);

    printf("Hora: %d\n", tiempo.wHour);



}
```

A

— A continuación se muestra el listado de un programa cuyo objetivo es solicitar al sistema operativo que reserve una región de memoria de 8 páginas a partir de la dirección contenida en el puntero *q* y con el atributo de protección de lectura/escritura. La dirección de la región reservada debe almacenarse en el puntero *p*. Después de realizar la reserva el programa envía un mensaje a pantalla indicando la dirección efectiva a partir de la cual se ha realizado la reserva. Teniendo en cuenta esta información, escribe la sentencia *VirtualAlloc()* necesaria en el listado siguiente.

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

main()
{
    void *p;
    void *q = (void *)0x00800534;

    p = VirtualAlloc( q, 8*4096, MEM_RESERVE, PAGE_READWRITE );

    printf("Direccion de reserva: %p", p);
}
```

0,5

— ¿Cuál será la dirección más significativa de la región reservada en la operación anterior? Contesta en hexadecimal.

00807FFF

0,5

— En la sección de datos de un programa ensamblador se encuentran definidas las variables que se indican a continuación:

```
.DATA
contador_A DD 0
contador_B DD 0
```

Escribe un fragmento de código ensamblador que en función del contenido del registro EAX, incremente el contenido de *contador_A*, si EAX >= 0, e incremente el contenido de *contador_B*, si EAX < 0. Utiliza las etiquetas que consideres oportunas.

```
  cmp  eax, 0
  jl   menor_cero
  inc  [contador_A]
  jmp  sigue
menor_cero:
  inc  [contador_B]
sigue:
```

0,5

— Contesta a las siguientes cuestiones sobre las interrupciones enmascarables utilizadas en los procesadores IA-32.

Objetivo:
son utilizadas por los periféricos para solicitar la atención del procesador.

Nombre y finalidad de las líneas de control usadas para gestionar este tipo de interrupciones:
INTR (Interrupt Request): Se utiliza para solicitar interrupción al procesador.

INTA (Interrupt Acknowledge): Es utilizada por el procesador para inidicar que ha aceptado una solicitud de interrupción.

Mecanismo de enmascaramiento:
Bit IF del registro de estado. Si IF=0 no se aceptan las interrupciones. Si IF=1 sí se aceptan.

0,5

A Apellidos _____ Nombre _____ DNI _____

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2006**

— Indica y describe brevemente los componentes básicos de un disco duro

0,5

Platos: Están formados por una aleación rígida de aluminio y recubiertos por una capa de material magnético sobre la que se graba la información.

Motor de giro: Su objetivo es hacer girar los platos a velocidad constante.

Cabezas de lectura/escritura: Su objetivo es escribir información sobre la superficie de los platos y leer de ellos. Hay una cabeza por cada superficie.

Brazo: Su objetivo es servir de soporte para la cabeza.

Actuador: Es un servomotor encargado de mover los brazos para posicionar las cabezas en las posiciones de los platos requeridas.

— Contesta a las siguientes preguntas acerca de los procesos Windows y a la API Win32.

0,5

Nombre de la función de la API Win32 utilizada para poner procesos en ejecución:
CreateProcess()

Nombre del fichero de cabecera (.h) que es nacesario incluir en los programas C cuando se hace uso de la API Win32:
Windows.h

Rango de direcciones correspondiente al espacio de usuario de un proceso Windows:
00000000-7FFFFFFF

— Explica los conceptos de localidad espacial y localidad temporal en el acceso a las instrucciones y datos de un programa.

0,5

Localidad espacial:
Cuando un programa accede a una instrucción o a un dato, existe una elevada probabilidad de que instrucciones o datos cercanos sean accedidos pronto.

Localidad temporal:
Cuando un programa accede a una instrucción o a un dato, existe una elevada probabilidad de que esa misma instrucción o dato vuelva a ser accedido pronto.

— En la definición técnica de un proceso se indica cuáles son los cuatro elementos fundamentales que lo forman. Indica a continuación dichos elementos y describe brevemente cada uno de ellos.

0,5

Imagen binaria de un programa. Ésta está formada por las instrucciones y datos del programa.

La pila. Área de memoria en la que se almacenan datos temporales.

La tabla de páginas. Traduce las direcciones virtuales generadas por el proceso en las direcciones físicas en la que se encuentra almacenado

El PCB. Estructura utilizada por el sistema operativo para controlar la ejecución del proceso.