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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 11-09-2006**

En el programa cuyo listado se muestra a continuación se encuentra definido el procedimiento *elimina*. El objetivo de este procedimiento es eliminar un determinado número de un array de números. Este procedimiento recibe dos parámetros a través de la pila: 1) la dirección del array de números en el que se desea llevar a cabo la eliminación y 2) el número que se desea eliminar. El array debe estar formado por números positivos y terminado con el dato -1. Este -1 no es un dato en sí del array, si no un elemento que actúa de terminador, y cuyo objetivo es facilitar el procesamiento del array.

A continuación se explica la forma de trabajar del procedimiento *elimina* utilizando como ejemplo el array *datos* de la sección de datos del programa. Para ello imagina que se le indica al procedimiento que se desea eliminar el dato 12 del array *datos*. El procedimiento va procesando uno a uno los elementos de *datos* hasta encontrar el 12. A partir de este punto copiaría el dato 9 en la posición ocupada por el 12, el 31 en la posición del 9 y el -1 en la posición del 31. Tras llevar a cabo este procesamiento el array *datos* quedaría de la siguiente manera:

```
  datos       DD    1, 7, 9, 31, -1, -1
```

Observa que al copiar los datos del array una posición a la izquierda a partir del 9, el terminador (-1) queda duplicado. Esto no tiene ninguna importancia ya que el array quedaría terminado por el primer terminador.

En el caso de que el número que se le pide eliminar al procedimiento no se encuentre en el array cuya dirección se le pasa como parámetro, el procedimiento no realiza modificación alguna del array.

El procedimiento *elimina* devuelve en el registro *eax* un valor que indica si se ha encontrado el número que se desea eliminar. Si el número se ha encontrado, el procedimiento retorna el valor 0. Si no se ha encontrado, retorna el valor -1.

El objetivo del programa principal es obtener una versión modificada del array *datos* en el que se haya eliminado el número 9. Dicha versión modificada debe quedar almacenada en el array *resultado* sin que resulte modificado el array *datos*. Para ello, el programa principal primero hace una copia del array *datos* en el array *resultado*, después llama al procedimiento *elimina*, pasándole como parámetros la dirección de *resultado* y el número a eliminar. Finalmente el programa principal chequea el valor devuelto por el procedimiento y escribe en la variable *eliminacion* de la sección de datos el carácter 'S' si el número que se ha solicitado eliminar al procedimiento se encuentra en el array cuya dirección se le pasa como parámetro, y el carácter 'N' en el caso contrario.

Teniendo en cuenta toda esta información contesta a las preguntas que se indican a continuación del listado.

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
  datos       DD    1, 7, 12, 9, 31, -1
  resultado   DD    0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  eliminacion DB    0

.CODE
elimina PROC
  push  ebp
  mov   ebp, esp
  push  edx
  push  esi
  push  edi
  push  ebx

  mov   edx, [ebp+8]                    ; 8B 55 08
  mov   esi, [ebp+12]                   ; 8B 75 0C

  ; Buscar el número a eliminar (esi quedará apuntando a ese numero)
bucle1:
  cmp   [esi], DWORD PTR –1             ; 83 3E FF
  je    SHORT no_encontrado             ; 74 09
  cmp   [esi], edx                      ; 39 16
  je    SHORT si_encontrado             ; 74 0C
  add   esi, 4                          ; 83 C6 04
  jmp   SHORT bucle1 ◄◄◄

no_encontrado:
  mov   eax, -1           ; para retornar el valor -1
  jmp   SHORT fin_proc

si_encontrado:
  mov   eax, 0            ; para retornar el valor 0

  ; El número se ha encontrado se pasa a su eliminación
  ; Se usan esi y edi para apuntar a los números y recolocarlos
  mov   edi, esi
bucle2:
  cmp   [esi], DWORD PTR -1
  je    fin_proc
  add   esi, 4
  mov   ebx, [esi]
  mov   [edi], ebx
  (--2--)
  jmp   bucle2
```

A

```
fin_proc:
  pop   ebx
  pop   edi
  pop   esi
  pop   edx
  pop   ebp
  (--3--)
elimina ENDP

inicio:
  ; Copiar array datos en array resultado
  xor esi, esi
  xor edi, edi
  xor eax, eax
bucle3:
  ; Coipar datos hasta que se alcance el terminador (-1)



  (--1--)



fuera:
  ; Copiar el terminador (-1)
  (--1--)

  ; Llamar al procedimiento elimina
  push  OFFSET resultado
  push  9
  call  elimina

  ; Se actualiza la variable eliminacion de la forma apropiada



  (--4--)



  ; Retorno al S.O.
  push 0
  call ExitProcess
END inicio
```

— En el hueco (—**1**—) faltan un conjunto de instrucciones, que son necesarias para copiar el array *datos* en el array *resultado*. Determina cuáles son estas instrucciones teniendo en cuenta las siguientes indicaciones:

- Los registros *esi* y *edi* participan en los modos de direccionamiento que se deben usar para acceder a los elementos de ambos arrays. Observa que estos registros se inicializan con el valor 0. Debes usar los modos de direccionamiento adecuados, teniendo en cuenta esta inicialización. Es decir, NO VALE inicializar de nuevo estos registros con las direcciones de comienzo de los arrays.
- Puedes utilizar o no factor de escala en los modos de direccionamiento, según estimes conveniente.
- Debes utilizar el registro *eax* como registro intermedio para mover los números entre el array *datos* y el array *resultado*.
- Ten en cuenta que en el listado del programa se encuentran definidas las etiquetas *bucle3* y *fuera*. Debes utilizar estas etiquetas en el código que falta.

1

```
Bucle3:
  ; Copiar datos hasta que se alcance el terminador (-1)
  cmp   DWORD PTR [datos+esi*4], -1
  je    SHORT fuera
  mov   eax, [datos+esi*4]
  mov   [reaultado+edi*4], eax
  inc   esi
  inc   edi
  jmp   bucle3


fuera:
  ; Copiar el terminador (-1)
  mov   eax, [datos+esi*4]
  mov   [resultado+edi*4], eax
```

— ¿Qué instrucción es necesaria en el hueco (—**2**—) del listado?

0,5

```
add edi, 4
```

— ¿Qué instrucción es necesaria en el hueco (—**3**—) del listado?

0,5

```
ret 8
```

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 11-09-2006**

— Escribe las instrucciones así como las etiquetas necesarias para modificar la variable *eliminación* de la forma apropiada, justo cuando se retorna del procedimiento *elimina*. Dichas instrucciones se ubican en el hueco (—**4**—) del listado. **0,5**

```
  cmp   eax, 0
  je    SHORT elimin_si
  ; No se produce la eliminación
  mov   [eliminacion], 'N'
  jmp   SHORT sigue
elimin_si:
  ; Sí se produce la eliminación
  mov   [eliminacion], 'S'
sigue:
```

— Determina el valor mínimo que alcanza el registro ESP durante la ejecución del programa. **0,5**

```
0012FFA4
```

— Determina la codificación de la instrucción "`jmp SHORT bulce1`" marcada en el listado con el símbolo ◄◄◄. Se sabe que el código de operación de esta instrucción es "EB". En el listado se indica mediante comentarios el código máquina de un conjunto de instrucciones que se encuentran justo antes de esta instrucción. Contesta en hexadecimal. **0,5**

```
EB F2
```

— Se pretende diseñar un procedimiento que utiliza tres variables locales, todas ellas de tipo doble palabra. Indica las instrucciones que de forma estándar (es decir, siguiendo el mecanismo estandarizado de creación y destrucción del marco de pila) se utilizarían para reservar espacio en la pila para estas variables y para eliminar el espacio ocupado por ellas. **0,5**

```
; Reservar espacio
sub esp, 12

; Eliminar espacio
mov esp, ebp
```

La MMU de un computador utiliza un sistema de traslación de direcciones con las siguientes características:

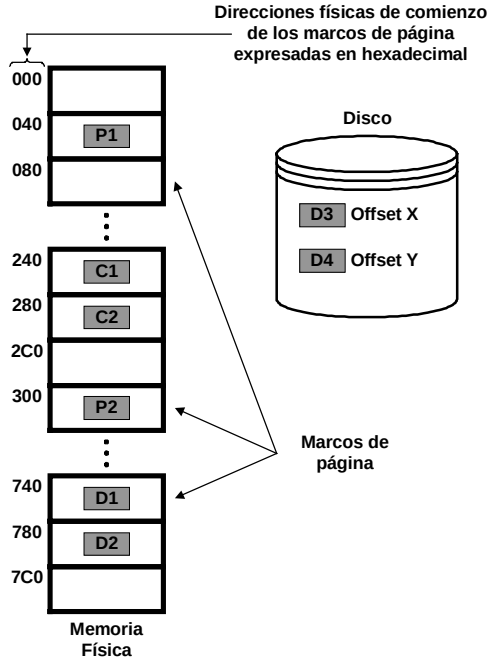
Dirección física: 12 bits

Dirección virtual: 14 bits

Tamaño de página: 64 bytes

Además se sabe que la memoria física ocupa la primera mitad del espacio de direcciones físico.

En la figura que se proporciona a la derecha se muestra cómo un proceso que se encuentra en ejecución en el sistema utiliza la memoria física y el disco. Tal y como se puede observar en la figura, el proceso requiere ocho páginas de espacio de almacenamiento: dos páginas de código (C1 y C2), cuatro páginas de datos (D1, D2, D3 y D4) y dos páginas de pila (P1 y P2).



A continuación se indica en una tabla qué páginas virtuales utiliza el proceso para direccionar sus páginas de código, datos y pila.

Página del proceso	Página virtual asignada
C1	20
C2	21
D1	22
D2	23
D3	24
D4	25
P1	80
P2	81

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

— A) A continuación se indica una parte de la tabla de páginas del proceso. Debes rellenar los campos Presencia y Nº pag. fis/Off. disco correspondientes a todos los números de página virtual indicados en la tabla. En el campo Nº pag. fis/Off. disco debes contestar con un guión si la página virtual correspondiente no es utilizada por el proceso. El número de página física, donde corresponda, debe indicarse en hexadecimal. En el campo Presencia debes contestar Si o No según corresponda.

A

Tabla de Páginas 1

Nº Pag. Virtual	Pre-sencia	Nº Pag. Fis. / Offset Dis.
00	No	---
01	No	---
•	•	•
22	Si	1D
23	Si	1E
24	No	Offset X
25	No	Offset Y
•	•	•
80	Si	01
81	Si	0C
•	•	•

— B) Determina cuál es la dirección más significativa del proceso que al ser enviada a la MMU provocaría una excepción de fallo de página. Contesta en hexadecimal.

0,5

097F

— Define la TLB indicando claramente su objetivo

0,5

Es una memoria de alta velocidad totalmente integrada con la MMU, que almacena las entradas de la tabla de páginas más frecuentemente utilizadas. Así las direcciones virtuales pueden ser traducidas sin necesidad de acceder a la memoria física. Un mecanismo hardware permite actualizar dinámicamente el TLB con las entradas de la tabla de páginas más frecuentemente utilizadas.

A continuación se proporciona el listado de un programa que utiliza una región de memoria virtual para llevar a cabo un determinado procesamiento. La región a utilizar por el programa debe ser de 64 páginas. El programa rellena esta región de memoria con el carácter '-' y, después, aplica un determinado procesamiento a la región. El procesamiento concreto a realizar, que no es relevante para la resolución del ejercicio, no se muestra en el listado.

Rellena los huecos que faltan en el listado siguiendo las indicaciones de los comentarios.

```
#include <windows.h>

main()
{
    char *p;  // Para recoger la dirección devuelta por VirtualAlloc
    int i;    // Para ser utilizada como contador

    // Reservar y comprometer una región de 64 páginas
    // El sistema operativo elige la dirección de la región
```

0,5

```
    p = VirtualAlloc( NULL,
                      64*4096,
                      MEM_RESERVE | MEM_COMMIT,
                      PAGE_READWRITE            );
```

```
    // Rellenar toda la región con el carácter '-'
```

0,5

```
    for ( i=0; i<(64*4096); i++ )
        *(p+i) = '-';

    // El programa lleva a cabo un determinado procesamiento
    // con la región de memoria. Este procesamiento no se muestra
    ...

    // Liberar completamente la memoria indicando si la operacion
    // tiene exito o no
```

0,5

```
    if ( VirtualFree( p, 0, MEM_RELEASE ) )
       printf("\nLiberacion OK\n");
    else
       printf("\nFallo en la liberacion de memoria\n");
}
```

— Suponiendo que en el programa anterior la dirección devuelta por *VirtualAlloc()* sea 00530000 (hexadecimal), determina cuál es el número de la última página de la región comprometida por *VirtualAlloc()*. Contesta en hexadedimal.

0,5

0056F

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 11-09-2006**

— Indica cuál o cuáles de las siguientes afirmaciones son CIERTAS. Contesta ninguna si crees que ninguna lo es.

A) En un sistema de proceso por lotes el sistema operativo primero carga todos los programas del lote en la memoria para su ejecución y después los ejecuta uno a uno generando sus resultados en una impresora.

B) Desde el punto de vista del uso de los recursos, el inconveniente de un sistema de proceso por lotes es el gran desequilibrio que se produce entre el uso de la CPU y de los dispositivos de E/S.

C) En un sistema multiprogramado se conoce como planificación de trabajos (*job escheduling*) al mecanismo mediante el cual el sistema operativo suspende la ejecución de un programa cuando éste hace una operación de E/S y le concede la CPU a otro programa.

D) En los *mainframes* controlados por un sistema operativo de tiempo compartido o multitarea los programas se cargan en la memoria utilizando un lector de tarjetas perforadas.

0,5

B

— Define los conceptos de arquitectura y organización de computadores.

0,5

Arquitectura:

Especificación del computador en su nivel de lenguaje máquina. Es decir, el juego de instrucciones, los tipos de operandos sobre los que éstas actúan y el espacio o espacios de direcciones.

Organización:

Conjunto de componentes físicos que conforman el ordenador, así como sus interrelaciones.

— Contesta a las siguientes preguntas acerca de la arquitectura IA-32.

0,5

Nombre de la estructura de datos que contiene las direcciones de los manejadores que atienden a interrupciones, excepciones y llamadas al sistema:
IDT (Interrupt Descriptor Table)

Nombre de la línea (señal) utilizada por la CPU IA-32 para señalizar que acpeta una interrupción:
INTA

Categoría de excepciones a la que pertenece la excepción de división por 0:
Fallo

— Indica cuáles son las razones para la computación concurrente a nivel de programa y explica cada una de ellas.

0,5

Mejorar el aprovechamiento de los recursos del computador: ya que los bloqueos en un flujo, debido por ejemplo a operaciones de E/S, no evita que otros flujos sigan avanzando.

Mejora de la interactividad con el usuario: ya que un flujo de ejecución puede dedicarse a atender al usuario, mientras otros flujos llevan a cabo otras computaciones requeridas por el programa.

Utilización de arquitecturas multiproceador: Posibilita que un programa pueda ser ejecutado por varios procesadores simultáneamente, dedicándose un procesador a cada flujo de ejecución.