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

**Examen de Arquitectura de Computadores. (Telemática.)** **Convocatoria de septiembre: 10-09-2005**

A continuación se muestra el listado de un programa cuyo objetivo es calcular el factorial de cada uno de los números de una lista, almacenando los resultados obtenidos en otra lista. La lista que contiene los números cuyos factoriales queremos obtener se encuentra en la sección de datos del programa y se llama *lista_origen*. Los resultados se almacenan en *lista_destino*. Para llevar a cabo el cálculo requerido, el programa implementa los procedimientos *Factorial* y *Multiplica*.

El procedimiento *Factorial* recibe como parámetro a través de la pila el número cuyo factorial se desea calcular. Este procedimiento devuelve el resultado en el registro EDX. El procedimiento utiliza números de 32 bits para hacer los cálculos, por lo que el mayor número cuyo factorial se puede calcular sin que se produzca desbordamiento es 12 (decimal). Cuando el procedimiento recibe como parámetro un número mayor que 12, devuelve –1, lo que indica el correspondiente error.

Para llevar a cabo las multiplicaciones requeridas por el procedimiento *Factorial*, se implementa el procedimiento *Multiplica*. Este procedimiento recibe los dos números a multiplicar a través de la pila y devuelve el resultado en el registro EDX.

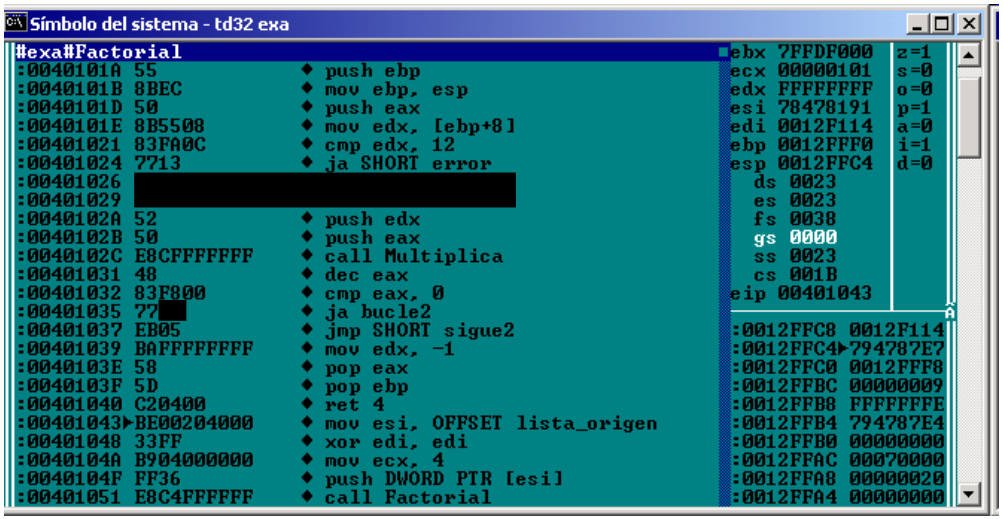
El programa principal procesa mediante un bucle todos los elementos de *lista_origen*, calculando el factorial de cada elemento y almacenándolo en el elemento correspondiente de *lista_destino*.

```
.386
.model flat
Extern ExitProcess:PROC

.DATA
lista_origen  DD  4, 8, 15, 6
lista_destino DD  0, 0, 0, 0

.CODE
Multiplica PROC
  push ebp
  mov  ebp, esp
  push eax
  push ecx
  xor  edx, edx
  mov  eax, [ebp+8]
  mov  ecx, [ebp+12]
  jcxz SHORT sigue1     ; No hacer el bulce si ecx=0

bucle1:
  add  edx, eax
  loop bucle1

sigue1:
  pop  ecx
  pop  eax
  pop  ebp
  ret 8
Multiplica ENDP

Factorial PROC
  push ebp
  mov  ebp, esp
  push eax

  mov  edx, [ebp+8]
  cmp  edx, 12
  ja   SHORT error

  (--1--)

bucle2:
  push edx
  push eax
  call Multiplica
  dec  eax
  cmp  eax, 0
  ja   bucle2  ◄◄◄
  jmp  SHORT sigue2

error:
  mov  edx, -1

sigue2:
  pop  eax
  pop  ebp
  ret  4
Factorial ENDP

Inicio:
  mov  esi, OFFSET lista_origen
  xor  edi, edi
  mov  ecx, 4

bucle:
  push DWORD PTR [esi]
  call Factorial

  (--2--)
  add  esi, 4
  inc  edi
  loop bucle
```

# A

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

END inicio
```

A continuación se muestra una captura TurboDebugger justo en el momento en el que el programa comienza a ejecutarse.




Teniendo en cuenta toda esta información contesta a las preguntas que se indican a continuación:

— ¿Qué instrucciones son necesarias en el hueco (—**1**—) del listado? No se puede utilizar más de dos instrucciones.

**0,5**

```
mov eax, [ebp+8]
dec eax
```

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

**0,5**

```
mov [lista_destino + edi*4], edx
```

— Codifica la instrucción *ja bucle2* marcada en el listado con el símbolo ◄◄◄.

**0,5**

```
77 F3
```

— Indica el rango de direcciones en el que se almacena el segundo parámetro (en orden de apilación) recibido por el procedimiento *Multiplica*. (Ejemplo de respuesta: 00402000 – 00402002).

**0,5**

```
0012FFAC – 0012FFAF
```

— Cuando se ejecuta la instrucción *call Factorial*, indica todos los registros y direcciones de memoria que cambian de valor, así como el valor que adquieren. (Ejemplo de respuesta: EAX = 0000 0000; Dir 0040 2000 = FFBB CCDD)

**0,5**

```
EIP = 0040101A;
ESP = 0012FFBC;
Dir 0012FFBC = 00401056;
```

— Indica el número de veces que la instrucción *ja bucle2* (marcada en el listado con el símbolo ◄◄◄) salta a la etiqueta *bucle2* durante la ejecución completa del programa. Contesta en DECIMAL.

**0,5**

```
12
```

— Imagina que en la sección de datos del programa se incluyen dos nuevas variables, de tipo doble palabra, denominadas *Aciertos* y *Fallos*. El objetivo de la variable *Aciertos* es almacenar el número de elementos de los que ha sido posible calcular el factorial durante el procesamiento. La variable *Fallos* almacenará el número de elementos de los que no ha sido posible calcular el factorial. Escribe un fragmento de código que se incluiría justo a continuación de la instrucción correspondiente al hueco (—**2**—) y que actualice las variables *Aciertos* y *Fallos* de la forma apropiada. No utilices más de 5 instrucciones. Puedes utilizar las etiquetas que estimes oportunas.

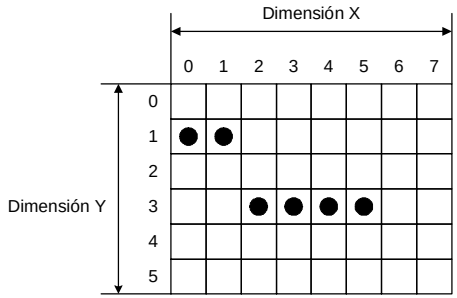
**0,5**

```
  cmp edx, -1
  jne Acierto
  inc [Fallos]
  jmp sigue
Acierto:
  inc [Aciertos]
sigue:
```

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

**Examen de Arquitectura de Computadores. (Telemática.)** **Convocatoria de septiembre: 10-09-2005**

Se ha diseñado un programa cuyo objetivo es generar ficheros que representan imágenes binarias. Una imagen binaria está formada por una matriz de puntos, pudiendo encontrarse cada punto sólo en dos estados: "apagado" o "encendido". La imagen a representar puede tener en principio cualquier tamaño, quedando éste definido por las dimensiones X e Y. A continuación se muestra un ejemplo de imagen binaria, con dimensiones X=8 e Y=6, y que contiene una línea horizontal de dos puntos entre las coordenadas (0, 1) y (1, 1) y otra de 4 puntos entre las coordenadas (2, 3) y (5, 3).




El programa representa la imagen anterior generando el siguiente fichero de texto:

```
00000000
11000000
00000000
00111100
00000000
00000000
```

Como se puede observar, los puntos de la imagen apagados se representan por un '0' (carácter '0', no número '0', ya que se trata de un fichero de texto) y los puntos encendidos, por un '1' (carácter '1').

Para generar la imagen, el programa utiliza memoria virtual, cuyo tamaño se ajusta en función de las dimensiones de la imagen. Dicha memoria se trata como un vector de bytes de una sola dimensión. Por ejemplo, para generar la imagen anterior, sería necesario un vector de 8 x 6 = 48 bytes. En los 8 primeros bytes del vector se almacena la fila 0 de la imagen, en los 8 siguientes la fila 1 y así sucesivamente hasta la fila 5.

Para llevar a cabo este programa, además de las funciones conocidas, se utilizan estas otras:

```
VOID FillMemory (
  PVOID Destination,  // puntero al bloque a rellenar
  DWORD Length,       // tamaño, en bytes, del bloque a rellenar
  BYTE  Fill    );    // valor del byte con el que se rellana
```

*FillMemoy* es una función de la API Win32 que rellena el bloque apuntado por *Destination* de longitud *Length* con el byte *Fill*.

```
int fputc( int car, FILE *stream );
```

*fputc* es una función de la librería estándar del C que escribe el byte *car* en el fichero apuntado por *stream*.

El esquema básico de funcionamiento del programa es el siguiente:

1) El usuario introduce las dimensiones de la imagen por pantalla.
2) Se reserva y compromete la región de memoria necesaria (ni más ni menos) para generar la imagen.
3) Se inicializa toda la memoria de la imagen con ceros (cero carácter).
4) Se realiza un bucle en el que el usuario puede escribir líneas horizontales en la imagen (rellenando con '1's). Para ello, el usuario proporciona las coordenadas de comienzo de la línea y su longitud.
5) Finalmente, se escribe la imagen en un fichero llamado *fichero.txt*.

El listado correspondiente a este programa se proporciona a continuación. Rellena los huecos que hay en él atendiendo a los comentarios.

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

main()
{
    byte *p;              // Para apuntar a la memoria virtual
    int  X_dim, Y_dim;    // Dimensiones de la representacion
    int  X_coor, Y_coor; // Corrdenadas X e Y
    int  lon_lin;         // Tamaño de linea
    FILE *p_fich;
    int  i, j;            // Contadores auxiliares

    // Pedir por pantalla las dimensiones X e Y de la representacion
    printf("Dimension X = ");
    scanf("%d", &X_dim);
    printf("Dimension Y = ");
    scanf("%d", &Y_dim);

    // Reservar y comprometer la region necesaria
    // El SO elige la direccion
```

**0,5**

```
    p=VirtualAlloc(NULL,
                   X_dim*Y_dim,
                   MEM_RESERVE|MEM_COMMIT,
                   PAGE_READWRITE);
```

# A

```
    // Rellenar con ceros toda la memoria de la imagen
    FillMemory(p, X_dim*Y_dim, '0');

    // Bucle de escritura de lineas horizontales
    printf("\nBucle de introduccion de lineas horizontales\n");
    do
    {
        // Pedir coordenada X
        printf("\nCoordenada X = ");
        scanf("%d", &X_coor);

        // Pedir coordenada Y
        printf("Coordenada Y = ");
        scanf("%d", &Y_coor);

        // Pedir la longitud de linea a representar
        printf("Longitud = ");
        scanf("%d", &lon_lin);

        // Escribir la linea en la memoria usando una llamada a
        // FillMemory
```

0,5

```
        FillMemory(p+(Y_coor*X_dim+X_coor), lon_lin, '1');
```

```
        // Pedir que se pulse 's' para salir del bucle
        // u otra tecla para realizar una nueva iteracion
        printf("\ns = salir; otra tecla = nueva iteracion: ");
    }
    while(getch()!='s');

    // Escritura en fichero de la representacion
    p_fich = fopen("fichero.txt", "w");

    for (j=0; j<Y_dim; j++)
    {
        for (i=0; i<X_dim; i++)

            // Escribir un byte de la memoria en el fichero
```

0,5

```
            fputc(*(p+(j*X_dim+i)), p_fich);
        fputc('\n', p_fich);
        i=0;
    }
```

```
    // liberar completamente la memoria indicando si se tiene exito
    // o hay fallo
```

0,5

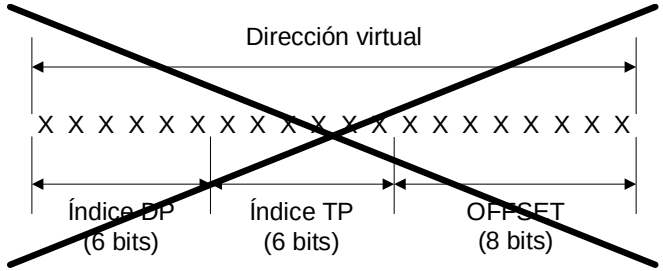
```
    if ( VirtualFree(p, 0, MEM_RELEASE) )
       printf("\nLiberacion OK\n");
    else
       printf("\nFallo en la liberacion de memoria\n");
}
```

— Se sabe que durante una ejecución del programa anterior el puntero *p* toma el valor 0x00530000 y las dimensiones de la imagen introducidas por el usuario son *X_dim=60*, e *Y_dim=500*. Determina cuál es la última dirección del bloque de memoria reservado por *VirtualAlloc()*. Contesta en hexadecimal.

0,5

**00537FFF**

En la figura siguiente se muestra la organización en campos de las direcciones virtuales manejadas por una MMU, que funciona utilizando un sistema de translación de direcciones de doble nivel.




El tamaño de las entradas del directorio de páginas y de las tablas de páginas es de 32 bits. Las direcciones físicas manejadas por este sistema son de 16 bits.

En un momento dado, un proceso que se encuentra en ejecución en este sistema utiliza los siguientes rangos de su espacio de direcciones:

Rango A: 01000 – 02DFF

Rango B: 14000 – 148FF

— Teniendo en cuenta esta información, determina el tamaño expresado en bytes del sistema de traslación de direcciones del proceso. Contesta en decimal.

0,5

**768**

— ~~Imagina que el directorio de páginas de este proceso se almacena en la página física número 7 (comenzando la numeración en 0). Determina la dirección física a partir de la cual se ubicaría la entrada más significativa de dicho directorio utilizada por el proceso (utilizada quiere decir que apunta a una tabla de páginas). Contesta en hexadecimal.~~

**0,5**

~~**0714**~~

— Pon un ejemplo de una instrucción de la arquitectura IA-32 que al ser ejecutada encontrándose la CPU en modo usuario genere siempre una excepción de violación de protección. Pon otro ejemplo de una instrucción que no la genere nunca.

**0,5**

**Instrucción que siempre genera excepción:**
**STI o CLI**
**Instrucción que nunca genera excepción:**
**Cualquiera que sólo trabaje sobre registros de aplicación. Por ejemplo inc eax**

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

A) Un proceso que se encuentre en el estado '*listo*' sólo puede pasar al estado '*en ejecución*'.

B) El *swapping* es el mecanismo utilizado por el sistema operativo para cambiar el proceso que se ejecuta en la CPU.

~~C) Utilizando un único *mailbox* (buzón), un proceso puede comunicarse con otros 2 o más procesos.~~

D) En un sistema Windows el proceso que controla la interfaz con el usuario se llama *explorer*.

**0,5**

**A, C, D**

— 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.**

— Explica los pasos correspondientes al protocolo de comunicación que se establece entre el PIC y la CPU IA-32, para que ésta pueda identificar al periférico que solicita interrupción. No dibujar la figura correspondiente, solamente indicar y explicar los pasos.

**0,5**

**1) El periférico solicita interrupción al PIC a través de una línea INTX.**

**2) El PIC pasa la solicitud de interrupción a la CPU a través de la línea INTR.**

**3) Si la CPU acepta la interrupción, se lo indica al PIC a través de la línea INTA.**

**4) El PIC envía a la CPU el número de interrupción correspondiente al periférico que solicitó la interrupción. Cada línea INTX tiene asociado un número distinto, identificándose de esta forma a los diferentes periféricos. La asociación de números de interrupción a las líneas INTX es programable.**

# A

— 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.**

— Indica cuáles son los tres objetivos básicos de un sistema operativo

0,5

**Proporcionar una interfaz amigable para la interacción entre el usuario y el computador.**

**Proporcionar un entorno de funcionamiento para los programas. Así el sistema operativo proporciona un conjunto de servicios que pueden ser solicitados por los programas.**

**Gestionar los recursos hardware del computador, de modo que éstos se repartan equilibradamente entre los procesos que se ejecutan en cada momento.**