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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 12-02-2005**

A continuación se muestra el listado de un programa cuyo objetivo es calcular, dada una lista de números, cuáles de ellos son divisibles por otro determinado número. Para llevar a cabo este procesamiento se diseña un procedimiento llamado *CalculaDivisibles*. Este procedimiento recibe los parámetros que se describen a continuación, en el orden indicado:

1) Número del cual queremos obtener los divisibles que se encuentran en la lista de números. (Si queremos calcular los divisibles por 4, este número sería 4).
2) Dirección de la lista de números a procesar.
3) Dirección de un *buffer* (*array*) en el que se almacenarán los números divisibles encontrados.
4) Número de elementos que se pueden almacenar en el *buffer* anterior. El objetivo de este parámetro es poder finalizar la ejecución de *CalculaDivisibles* cuando se agota el espacio en el *buffer* para almacenar números divisibles.

La lista de números a procesar puede tener cualquier tamaño, pero debe estar formada por números positivos y terminada con el número 0. Tanto la lista a procesar como el buffer para almacenar los divisibles están formados por datos de tipo doble palabra.

El procedimiento *CalculaDivisibles* retorna en el registro EDX los siguientes valores:

- 0 **à** si el procedimiento falla, es decir, si se produce alguna incidencia en el cálculo de los divisibles. Las incidencias pueden ser: 1) se ha encontrado un número negativo en la lista de números a procesar; y 2) se ha agotado el *buffer* en el que se almacenan los números divisibles.
- 1 **à** si el procedimiento tiene éxito, es decir, si no se produce ninguna incidencia en el cálculo de divisibles.

El procedimiento *CalculaDivisibles* procesa la lista de números cuya dirección recibe como parámetro hasta que encuentre un 0 (que indica el final de la lista) retornando en este caso un 1 en EDX. El procedimiento finalizará antes de procesar la lista completa si encuentra un número negativo o si se llena el buffer, retornado entonces un 0 en EDX.

El procedimiento *CalculaDivisibles* utiliza otro procedimiento, denominado *divide*, para llevar a cabo las divisiones necesarias. El procedimiento *divide* recibe los siguientes parámetros, en el orden indicado:

1) Dividendo
2) Divisor
3) Dirección de una estructura de la sección de datos en la que el procedimiento deja el resultado.

Para llevar a cabo la división el procedimiento *divide* utiliza el siguiente algoritmo:

```
Cociente = 0
MIENTRAS Dividendo >= Divisor HACER
   Dividendo = Dividendo – Divisor
   Cociente = Cociente + 1
FIN_MIENTRAS
Resto = Dividendo
```

El procedimiento divide devuelve el resultado (cociente y resto) en la estructura de la sección de datos *ResulDiv*.

El programa principal simplemente llama al procedimiento *CalculaDivisibles*, pasándole los parámetros adecuados, para calcular los números de la lista *ListaNumeros* (de la sección de datos) que son divisibles entre 7, y almacenarlos en el buffer *DivisiblesSiete*.

Como datos adicionales se sabe que la sección de datos comienza en la dirección 00402000 y que justo al comienzo de la ejecución del programa, ESP = 0012FFC4.

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

.DATA
ResulDiv        DD  0   ; Cociente
                DD  0   ; Resto

ListaNumeros    DD  4, 7, 33, 515, 49, 63, 39, 0
DivisiblesSiete DD  0, 0, 0, 0

.CODE
divide PROC
  push ebp
  mov  ebp, esp
  push edx
  push esi
  push ebx
  push eax

  xor  edx, edx      ; cociente = 0
  mov  esi, [ebp+8]  ; cargar direccion ResulDiv
  mov  ebx, [ebp+12] ; cargar divisor
  mov  eax, [ebp+16] ; cargar dividendo

bucle1:
  cmp  eax, ebx
  jb   fuera1
  inc  edx           ; incrementar cociente
  sub  eax, ebx      ; dividendo = dividendo - divisor
  jmp  bucle1
```

A

```
fuera1:
  ; almacenar resultado (cociente y resto) en memoria
  (--1--)

  pop  eax
  pop  ebx
  pop  esi
  pop  edx
  pop  ebp
  ret  12 ◄◄◄
divide ENDP

CalculaDivisibles PROC
  push ebp
  mov  ebp, esp
  push ecx
  push edi
  push esi
  push eax
  push ebx

  mov  ecx, [ebp+8]   ; cargar tamaño lista divisibles
  mov  edi, [ebp+12]  ; cargar dirección lista divisibles
  mov  esi, [ebp+16]  ; cargar dirección lista numeros
  mov  ebx, [ebp+20]  ; cargar divisor  ♥♥♥
  xor  eax, eax       ; resetear auxiliar

bucle2:
  ; Si se acaba el buffer de divisible indicar error
  cmp  ecx, 0
  je   error
  ; Si el número a procesar es negativo indicar error
  cmp  DWORD PTR [esi], 0
  jl   error
  ; Si se acaba la lista de números abandonar el bucle
  je   fuera2
  ; llamar al procedimiento divide
  (--2--)
  call divide

  ; Si núemero divisible, almacenarlo en el buffer de divisibles
  ; usar 'eax' para mover el dato a dicho buffer
  (--3--)
  add  edi, 4
  dec  ecx ; decrementar tamaño restante

sigue:
  add  esi, 4
  jmp  bucle2

fuera2:
  ; no hay error
  mov  edx, 1
  jmp  RestaurarEstado

error:
  mov  edx, 0

RestaurarEstado:
  pop  ebx
  pop  eax
  pop  esi
  pop  edi
  pop  ecx
  pop  ebp
  ret  16
CalculaDivisibles ENDP

inicio:
  push 7                        ; divisor
  push OFFSET ListaNumeros
  push OFFSET DivisiblesSiete
  push 4                        ; tamaño buffer de divisibles
  call CalculaDivisibles

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

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

0,5

```
mov [esi], edx
mov [esi+4], eax
```

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

0,5

```
push DWORD PTR [esi] ; dividendo
push ebx             ; divisor
```

```
push OFFSET ResulDiv
```

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

0,5

```
cmp [ResulDiv+4], 0
jne sigue
mov eax, [esi]
mov [edi], eax
```

— Justo en el momento en que el programa empieza a ejecutarse, hay una posición de la sección de datos que contiene el byte '02'. ¿Cuál es la dirección de dicha posición? Contesta con 8 dígitos hexadecimales.

0,5

```
00402015
```

— Determina el valor máximo que alcanza el registro EDI durante la ejecución del programa. Contesta en hexadecimal.

0,5

```
00402034
```

— Determina el valor del registro ESP justo después de la ejecución de la instrucción "**ret 12**", marcada en el listado con el símbolo "◄◄◄". Contesta en hexadecimal.

0,5

```
0012FF98
```

— Codifica la instrucción "**mov ebx, [ebp+20]**", marcada en el listado con el símbolo "♥♥♥". Contesta en hexadecimal.

0,5

```
8B 5D 14
```

A continuación se muestra el listado de un programa cuyo objetivo es contar el número de veces que una determinada cadena de caracteres aparece en un fichero de texto. La cadena a buscar es introducida por el usuario en pantalla y el fichero en el que se busca la cadena debe llamarse "texto.txt".

Para llevar a cabo el procesamiento requerido por este programa, se llevan a cabo las siguientes operaciones fundamentales: 1) se calcula el tamaño de fichero; 2) se compromete la memoria virtual necesaria para almacenar el fichero en memoria; 3) se copia el fichero en la memoria comprometida; 4) se agrega un terminador de cadena para poder tratar todo el fichero como una cadena de caracteres; 5) se pide al usuario que introduzca la cadena a buscar; y finalmente 6) se busca en la copia del fichero almacenada en memoria todas las apariciones de la cadena.

Para encontrar las sucesivas apariciones de la cadena se utilizan las funciones *strstr()* y *strlen()* de la librería estándar del C. Los prototipos de estas funciones se indican a continuación:

```
char *strstr(char *string, char *strCharSet);
int strlen(char *string);
```

La función *strstr()* retorna un puntero a la primera ocurrencia de la cadena *strCharSet* dentro de la cadena *string*, o bien NULL si *strCharSet* no se encuentra dentro de *string*. La función *strlen()* retorna la longitud de la cadena que se le pasa como parámetro.

Si llamamos a la cadena que contiene en memoria una copia del fichero, cadena A, y a la cadena que queremos buscar, cadena B, lo que se pretende es encontrar en la cadena A, todas las apariciones de la cadena B. Para conseguir esto se programa una estructura iterativa. En cada iteración se posiciona el puntero en una nueva aparición de la cadena B y luego se le suma la longitud de esta cadena. Así en la siguiente iteración se buscará en la parte restante de la cadena A.

Teniendo en cuenta toda esta información completa los huecos existentes en el listado del programa. **Ten en cuenta los comentarios y utiliza exclusivamente las variables definidas en el listado**.

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

main()
{
    int  numCadenas;    // Contador de cadenas
    char cadena[80];    // Para almacenar la cadena a buscar
    FILE *p_fich;       // Para manejar el fichero
    int  car;           // Auxiliar para recibir los car. del fich.
    unsigned numCar=0; // Para contar el numero de car. del fich.
    char *p;            // Para manejar en memoria los datos del fich.
    int  i;             // Contador auxiliar

    p_fich = fopen("texto.txt", "r");

    // Contar el numero de caracteres del fichero
```

0,5

```
    while( (car = getc(p_fich)) != EOF )
        numCar++;
```

A

```
    // Reservar y comprometer la memoria necesaria (ni mas ni memos)
    // para tratar el fichero. El S.O. elige la region
    p = VirtualAlloc( NULL,                                          0,25
                      numCar,
                      MEM_RESERVE | MEM_COMMIT,
                      PAGE_READWRITE );

    if (p==NULL)
    {
        printf("\nLa reserva de memoria ha fallado");
        close(p_fich);
        exit(0);
    }

    // Poner puntero del fichero al comienzo
    fseek( p_fich, 0, 0);

    i=0;
    // Almacenar los caracteres del fichero en memoria virtual
    // No modificar el puntero p, usar contador auxiliar
    while( (car = getc(p_fich) ) != EOF )                            0,5
    {
        p[i] = car;
        i++;
    }

    // Agregar terminador de cadena
    p[i] = '\0';

    // Cerrar el fichero
    fclose(p_fich);

    // Introducir por pantalla la cadena a buscar
    printf("Introducir cadena: ");                                   0,25
    scanf("%s", cadena);

    // Calculo del numero de apariciones de la cadena
    numCadenas = 0;
    do
    {
        p = strstr( p, cadena );                                     0,5
        if ( p != NULL )
        {
            numCadenas++;
            p += strlen(cadena);
        }
    }
    while( p != NULL );
    printf("Numero cadenas = %d", numCadenas);
}
```

A continuación se muestra el listado de un programa que muestra por pantalla el contenido del su propio bloque de entorno. Para ello, utiliza la función *GetEnvironmentStrings()* mediante la que obtiene la dirección del bloque de entorno y, a continuación, ejecuta dos bucles anidados. En cada iteración del bucle interior se imprime, en una línea, una variable junto con su contenido. En cada iteración del bucle exterior se cambia de línea. A continuación se muestra el prototipo de la función *GetEnvironmentStrings()*.

```
void *GetEnvironmentStrings(void);
```

Teniendo en cuenta la información anterior, rellena los huecos del listado siguiente.

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

main()
{
    char *p;

    // Cargar en p la direccion del bloque de entorno
    p = GetEnvironmentStrings();                                     0,25

    while(*p != '\0')
    {
        while(*p != '\0')
        {
            printf("%c", *p);                                        0,5
            p++;

        }
        printf("\n");                                                0,25
        p++;

    }
}
```

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 12-02-2005**

Los parámetros del sistema de gestión de memoria virtual de un determinado computador son los siguientes:

Tamaño de página = 1K

Tamaño de las direcciones físicas = 18 bits

Tamaño de las direcciones virtuales = 22 bits

El sistema de translación de direcciones funciona siguiendo un esquema de doble nivel, con un esquema de funcionamiento igual al de la arquitectura IA-32, aunque con un directorio y unas tablas de páginas más pequeñas.

En la dirección virtual, los bists del campo "Número de página virtual" se reparten a partes iguales entre los campos "Índice en el DP" e "Índice en la TP".

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

— A) Un proceso que se encuentra en ejecución en este sistema utiliza las siguientes direcciones de su espacio de direcciones:
Región de pila: 010000 – 011FFF
Región de código: 040000 – 041FFF
Región de datos: 0C0000 – 0E0FFF
Con objeto de controlar estas regiones, ¿cuántas tablas de páginas necesitará utilizar? (NO contar el directorio de páginas)

0,5

5

— B) Se sabe que todas las páginas de la región de pila se mapean en páginas consecutivas de la memoria física, siendo la primera página física utiliza por esta región la 08 (hexadecimal). El mapeo se realiza de modo que la página virtual menos significativa de esta región se ubica en la página física menos significativa de la región correspondiente y así sucesivamente hasta la página virtual más significativa que se mapea en la página física más significativa. Se conoce que una determinada variable A, ubicada en la pila, se encuentra en la dirección virtual 0100FC. Teniendo en cuenta toda esta información ¿En qué dirección de la memoria física estará almacenada? Contestar en hexadecimal.

0,5

020FC

— Contesta a las siguientes preguntas acerca de los procesos Windows:

0,5

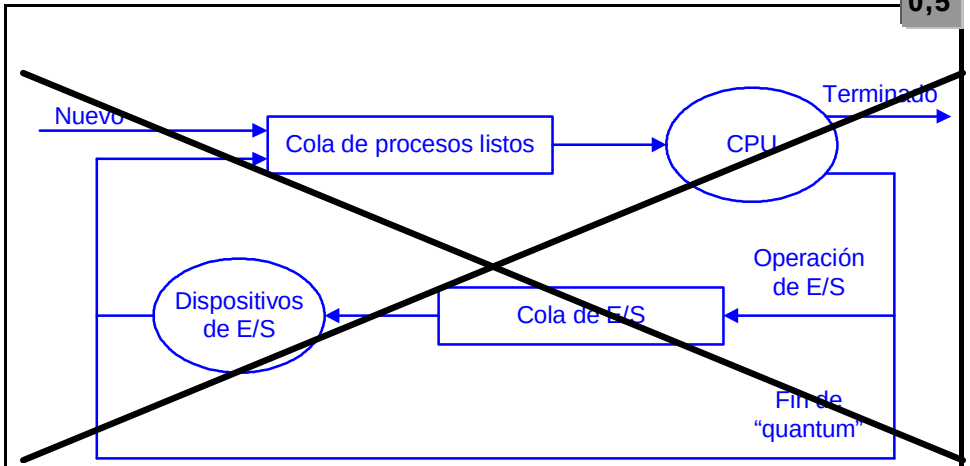
Nombre de la función de entrada de los programas de tipo GUI:
WinMain()

Rango de direcciones del proceso que se reserva para el sistema operativo (espacio de sitema):
80000000 - ffffffff

Indica la razón por la cual los primeros 64K del espcio de direcciones de un proceso no son accesibles:
Para detectar asignaciones NULL a punteros

— Dibuja el diagrama de colas simplificado (sin *swapping*) del mecanismo de planificación de procesos.

0,5



— Explica la diferencia fundamental existente entre el mecanismo de planificación de un sistema operativo multiprogramado y el de un sistema operativo de tiempo compartido.

0,5

En los sistemas multiprogramados, los programas sólo abandonan la CPU cuando esperan por un periférico.

En los sistemas de tiempo compartido, se asigna a los programas un tiempo máximo de ejecución continuada, que recibe el nombre de quantum. Entonces un programa abandona la CPU si tiene que esperar por un periférico, o bien si ha agotado su quantum de ejeucción.

A

— 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 ello. 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.

— 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.