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

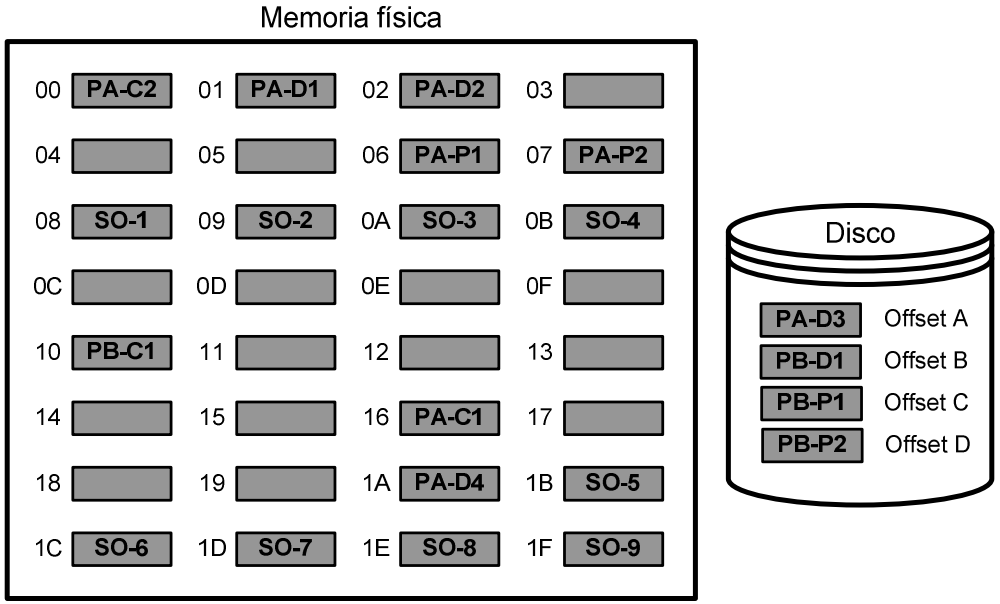
Examen de Arquitectura de Computadores. (Telemática) **Convocatoria de febrero, segunda parte: 08-02-2010**

Se dispone de un computador que utiliza un sistema de memoria virtual paginada con las siguientes características:

- Direcciones virtuales de 20 bits
- Direcciones físicas de 16 bits
- Cada página contiene 512 posiciones de memoria de 1 byte cada una

En este computador está instalado un sencillo sistema operativo multitarea. El único sistema de protección de memoria proporcionado por este sistema operativo es la utilización de tablas de páginas independientes para cada proceso.

En un determinado instante, además del sistema operativo, están en ejecución dos procesos, denominados PA y PB. Su disposición en memoria física (y disco) se muestra en la figura adjunta. Las páginas se nombran utilizando como prefijo el nombre del proceso (PA, PB o SO), seguido de una letra (C, D o P), que indica si la página es de código, datos o pila, y finalmente un número indicando el orden que ocupa la página en la sección del programa a la que pertenece. Así por ejemplo, las páginas PA-D1 y PA-D2 son las dos primeras páginas de la sección de datos del proceso PA, encontrándose ubicadas en el espacio de direcciones virtuales del proceso A primero PA-D1 y después, PA-D2.



Se sabe que la sección de código de todos los programas comienza siempre a partir de la dirección virtual 1A000h y la sección de datos a partir de la dirección virtual 60000h.

La mitad más alta del espacio de direcciones virtuales de cada proceso está siempre reservada para direccionar al sistema operativo, y la sección de pila de los procesos comienza siempre a partir de la dirección virtual más significativa del rango del espacio de direcciones virtuales reservado a procesos de usuario (la mitad más baja).

Todas las páginas virtuales que direccionan al sistema operativo se ubican de forma consecutiva a partir de la primera dirección del espacio de direcciones virtuales reservado al sistema operativo.

Nota: La sección de pila crece hacia direcciones menos significativas.

— Rellena la tabla de páginas del proceso B. Esta tabla debe direccionar las páginas que forman las secciones de código datos y pila del proceso B, además de las páginas del sistema operativo.

Tabla de Páginas de PB **1,25**

Nº Pag. Virtual	Pre-sencia	Nº Pag. Fis. / Offset Dis.
0D0	Sí	10
300	No	Offset B
3FE	No	Offset D
3FF	No	Offset C
400	Sí	08
401	Sí	09
402	Sí	0A
403	Sí	0B
404	Sí	1B
405	Sí	1C
406	Sí	1D
407	Sí	1E
408	Sí	1F

A

— En la figura anterior tan sólo se muestra una parte de la tabla de páginas, ya que ésta es muy grande. Teniendo en cuenta que para cada página virtual sólo se almacena el marco de página asociado a ella y el bit de presencia, ¿cuánto ocupa la tabla de páginas del proceso B?

2KB — 0,5

— ¿Qué porcentaje del espacio de direcciones físico está ocupado por la memoria física?

25% — 0,5

— Las dos primeras páginas del proceso A (PA-D1 y PA-D2) se utilizan para contener un array de 256 enteros (cada entero ocupa 4 bytes). ¿Qué rango de direcciones físicas se utilizan para almacenar el elemento vector[131]?

Nota: El primer elemento del vector es el que tiene el índice 0.

040C – 040F — 0,5

— Se sabe que la primera posición de la página SO-8 contiene información crítica del sistema. Si un usuario malintencionado quisiera borrar el contenido de esa posición para colgar el sistema (aprovechando la carencia de mecanismos de protección de memoria), ¿en qué dirección virtual tendría que escribir?

80E00 — 0,25

— Dibuja en el siguiente recuadro los estados por los que puede pasar un proceso durante su ciclo de ejecución y los eventos que pueden provocar que un proceso pase de uno de esos estados a otro.

0,75

— Indica cuáles son las tres operaciones básicas que llevan a cabo las herramientas de desarrollo (compilador + linker) durante el proceso de generación de un programa ejecutable:

0,5

Generan el código máquina, que está formado por instrucciones y datos.

Almacenan el código generado en el disco en un fichero.

Asignan direcciones virtuales a las instrucciones y datos generados.

— Responde a las siguientes preguntas breves:

0,5

¿Qué son los programas de sistema?
Son partes del sistema operativo que proporcionan servicios a los usuarios

¿Qué tipos de programas de sistema existen en los sistemas operativos Windows y Unix? Pon un ejemplo de cada uno.

Demonios o servicios. Ejemplo: explorer.exe (interfaz gráfica con el usuario)
Utilidades o accesorios. Ejemplo: ipconfig.exe (información sobre interfaces de red)

— Define los siguientes conceptos:

0,5

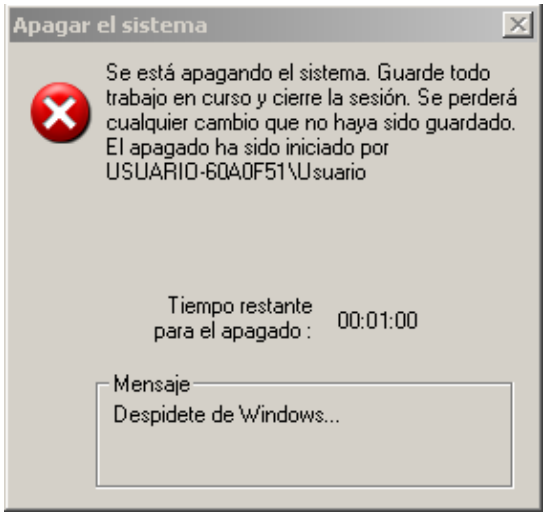
API:
Especificación de un sistema o componente software proveedor de servicios, así como los mecanismos necesarios para que dichos servicios puedan ser utilizados por otros componentes software.

Quántum de ejecución:
Es el tiempo máximo que un proceso puede ocupar la CPU de forma continuada.

— Dibuja el esquema de funcionamiento de una MMU. En dicho esquema debe aparecer la unidad de ejecución (EU), la propia MMU, la TLB, la memoria principal y la tabla de traducción.

0,5

Se desea programar una especie de virus molesto tipo *blaster*, que reinicie el computador cada vez que es ejecutado. En vez de reiniciarlo de forma instantánea, antes muestra por pantalla un mensaje intimidatorio con una cuenta atrás, igual que el que se muestra en la figura siguiente.



Si no consigue desactivar el virus, cuando se cumpla el tiempo se apagará el computador y posteriormente se reiniciará automáticamente. Si el usuario no guardó los cambios realizados en las aplicaciones que tenga abiertas, éstos se perderán.

Para realizar el apagado del computador, se utilizará la función de la API de Windows **InitiateSystemShutdown**. De manera auxiliar, se utilizará también la función **GetLastError** para obtener información sobre los errores que se pudieran producir. La especificación de ambas funciones se proporciona en el anexo.

A continuación se muestra un programa en C vacío. Completar este programa siguiendo las indicaciones de los comentarios.

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

int main () {

      // Declaración de las variables utilizadas para
      // recoger los valores de retorno de las funciones
      BOOL res;
      DWORD error;

      // Llamada a la función InitiateSystemShutdown. La llamada
      // producirá una ventana como la mostrada anteriormente con
      // una cuenta atrás de un minuto.
      res=InitiateSystemShutdown(NULL, "Despidete de Windows...",
                                 60 , TRUE, TRUE);

      // Si la función no se completó satisfactoriamente, mostrar
      // el código del error producido en la consola
      if (res==0) {
            error=GetLastError();
            printf("Error: %d", error);

      }

      return 0;
}
```

0,25

0,75

0,5

A

A continuación se muestra un código escrito en C para manejar vectores de números reales cuyo tamaño se desconoce en tiempo de compilación. Para ello se hace uso de una estructura con dos campos: un puntero a un número real (utilizado para apuntar a una zona de memoria reservada para albergar los elementos del vector) y un entero con el número de elementos que tiene el vector.

El número de elementos del vector lo introducirá el usuario por entrada estándar. Acto seguido, se reservará y comprometerá espacio suficiente para almacenar un vector de reales con tantos elementos como haya especificado el usuario. La zona de memoria reservada debe ser lo suficientemente grande, pero no más de lo necesario.

Una vez que la memoria haya sido reservada, un bucle inicializará cada uno de los elementos del vector a un valor. Posteriormente, otro bucle vuelve a recorrer el vector, mostrando por salida estándar el valor de cada uno de los elementos del vector.

Las lecturas y escrituras de cada elemento se hacen mediante dos funciones auxiliares, llamadas **leer** y **escribir**.

Rellena los huecos presentes en el código para que el programa realice estas tareas. Presta atención a las indicaciones dadas en los comentarios.

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

//Estructura utilizada para manejar el vector
typedef struct _vector {
	float * ptr;
	int size;
} vector;

//Prototipos de las funciones utilizadas para leer y escribir
float leer(vector v, int indice);
void escribir(vector v, int indice, float valor);


int main ()
{

	//Declaración de variables
	vector v;
	int i;
	float auxiliar;

	printf("Introduce el numero de elementos del vector: ");
	scanf_s("%d",&v.size);

	//Reserva y compromete la memoria
	//Utiliza el operador sizeof (especificación en el anexo)
	v.ptr=VirtualAlloc(NULL,sizeof(float)*v.size,
			MEM_RESERVE | MEM_COMMIT,PAGE_READWRITE);

	//Si falló la reserva, se muestra un mensaje de error
	if (v.ptr==NULL){
		printf("Se ha producido un error al reservar la
			memoria\n");
		_getch();
		return -1;
	}

	printf("Inicializacion del vector...\n");
	for (i=0;i<v.size;i++){
		printf("vector[%d]= ", i);
		scanf_s("%f",&auxiliar);
		escribir(v,i,auxiliar);
	}

	printf("Lectura del vector...\n");
	for (i=0;i<v.size;i++){
		printf("vector[%d]=%f\n",i,leer(v,i));
	}

	//Liberar la memoria reservada anteriormente. Si se produce
	//algún error durante la liberación, mostrar un mensaje
	//de error

	if (VirtualFree(v.ptr,0,MEM_RELEASE)==0){
		printf("Se ha producido un error al liberar la
			memoria\n");
	}

	_getch();
	return 0;

}


//CONTINÚA EN LA SIGUIENTE PÁGINA
```

0,75

0,5

```
float leer(vector v, int indice){

       //Si se intenta leer fuera del rango del vector
       //mostrar un mensaje de error
       if (indice>=v.size){
              printf("Acceso incorrecto\n");
              _getch();
              exit(-1);
       }

       //Retornar el elemento del vector indicado
       //HAZLO UTILIZANDO EL OPERADOR [] <----------------------------
       return v.ptr[indice];
}

void escribir(vector v, int indice, float valor)
{

       //Si se intenta escribir fuera del rango del vector
       //mostrar un mensaje de error
       if (indice>=v.size){
              printf("Acceso incorrecto\n");
              _getch();
              exit(-1);
       }

       //Escribir valor en el elemento indicado del vector
       //HAZLO UTILIZANDO EL OPERADOR * <----------------------------
       *(v.ptr + indice)=valor;
}
```

0,5

0,5

— Supóngase que se dispone de un programa igual al anterior, con la excepción de que maneja vectores de enteros (cada entero ocupa 4 bytes). En una ejecución de dicho programa, el usuario solicita memoria para almacenar un vector de 8192 elementos ($8192 = 2^{13}$). Si la dirección devuelta por la función **VirtualAlloc** es 0x00370000, ¿Cuál será la última dirección virtual del área reservada por **VirtualAlloc**? Contestar en hexadecimal.

0,5

0x00377FFF

InitiateSystemShutdown

La función **InitiateSystemShutdown** inicia el apagado, con reinicio opcional, del computador especificado. Para grabar un motivo por el apagado en el log de eventos, llama a la función **InitiateSystemShutdownEx.**

```
BOOL InitiateSystemShutdown(
  LPTSTR lpMachineName,
  LPTSTR lpMessage,
  DWORD dwTimeout,
  BOOL bForceAppsClosed,
  BOOL bRebootAfterShutdown
);
```

Parámetros

lpMachineName
[in] Puntero a la cadena de caracteres que especifica el nombre de red del computador a apagar. Si *lpMachineName* es NULL o una cadena vacía, la función apaga el computador local.

lpMessage
[in] Puntero a la cadena que especifica el mensaje a mostrar en el cuadro de diálogo que aparece al apagar el computador. Este parámetro puede ser NULL si no se requiere ningún mensaje.
Windows Server 2003 y Windows XP: Esta cadena también se guarda como comoentario en la entrada del log de eventos.
Windows Server 2003 y Windows XP SP1: Esta cadena está limitada a 3072 **TCHARs**.

dwTimeout
[in] Tiempo que se muestra en el cuadro de diálogo que aparece al apagar el computador, en segundos. Mientras se muestre este cuadro de diálogo, el apagado puede ser detenido por la función **AbortSystemShutdown**.
Si *dwTimeout* no es cero, **InitiateSystemShutdown** muestra un cuadro de diálogo en el computador especificado. El cuadro de diálogo muestra el nombre del usuario que llamó a la función, el mensaje especificado por el parámetro *lpMessage* y sugiere al usuario cerrar la sesión de trabajo. El cuadro de diálogo pita cuando se crea y permanece sobre todas las demás ventanas del sistema. El cuadro de diálogo puede moverse, pero no cerrarse. Un temporizador cuenta el tiempo que resta antes de que se fuerce el apagado del sistema.
Si *dwTimeout* es cero, el computador se apaga sin mostrar el cuadro de diálogo, y el apagado no puede ser detenido por **AbortSystemShutdown**.
Windows Server 2003 y Windows XP SP1: El valor de *dwTimeout* está limitado a MAX_SHUTDOWN_TIMEOUT segundos.
Windows Server 2003 y Windows XP SP1: Si el computador a apagar es un servidor de servicios, el sistema muestra un cuadro de diálogo a todos los usuarios locales y remotos avisándoles que el proceso de apagado ha comenzado. El cuadro de diálogo incluye quién solicitó el apagado, un mensaje (ver *lpMessage*), y cuanto tiempo falta antes de que el servidor se haya apagado.

bForceAppsClosed
[in] Si este parámetro es TRUE, las aplicaciones con cambios sin guardar se cerrarán forzosamente. Téngase en cuenta que esto puede suponer una pérdida de información.
Si este parámetro es FALSE, el sistema muestra un cuadro de diálogo ordenando al usuario que cierre todas sus aplicaciones.

bRebootAfterShutdown
[in] Si este parámetro es TRUE, el computador se reiniciará inmediatamente después de apagarse. Si este parámetro es FALSE, el sistema vuelca todas las cachés al disco y detiene el sistema de una forma segura.

Valores de retorno
Si la función tiene éxito, el valor de retorno es distinto de cero.
Si la función falla, el valor de retorno es cero. Para obtener información adicional sobre el error, se debe llamar a la función **GetLastError**.

GetLastError

La función **GetLastError** devuelve el código asociado al último error producido en el hilo llamador. El código del último error se mantiene para cada hilo en una tabla. Ningún hilo sobrescribe el valor del último código de error de ningún otro hilo.

```
DWORD GetLastError(void);
```

Parametros

Esta función no recibe parámetros.

Valores de retorno

El valor de retorno es el código del último error que se produjo en el hilo que invoca la función. Las funciones actualizan este valor mediante una llamada a la función **SetLastError**.

Operador sizeof

El operador **sizeof** devuelve el tamaño de su operando expresado en bytes.

Sintaxis:

```
sizeof expresión-unaria
```

o bien

```
sizeof ( tipo )
```

El operando de **sizeof** puede ser uno de los siguientes:
- El nombre de un tipo. Para utilizar **sizeof** con el nombre de un tipo, éste tiene que estar entre paréntesis.
- Una expresión unaria. Cuando se utiliza una expresión, ésta puede estar o no entre paréntesis. La expresión no se evalúa.

Retorno

El operador **sizeof** devuelve el número de bytes que ocupa el objeto/tipo que se le indica.