# Sistemas operativos

## Examen Diciembre 2020: Ejercicio 3

Seleccione un único enunciado y responda todas las preguntas.

**Enunciado A)**

Sea un sistema de archivos FAT que cuenta con la siguiente organización en el disco:

| Contenido | Sector de arranque | Comienzo de la FAT | Comienzo del sector de datos |
|---|---|---|---|
| Sector | 0 | 1 | 33 |

El sector de arranque comienza en el sector 0, la FAT en el sector 1, y la región de datos en el sector 33. Este sistema cuenta con las siguientes estructuras de datos:

```
type sector = array [0..4095] of byte;
type entrada_dir = record
        usado : bit;                                // entrada usada o no (1 bit)
        nom : array [0..19] of char;                // nombre del elemento (20 bytes)
        ext : array [0..2] of char;                 // extensión del elemento (3 bytes)
        tipo : {0=ARCHIVO, 1=DIRECTORIO};           // tipo de elemento (1 bit)
        inicio : int;                               // dirección de comienzo (4 bytes)
        tam : int;                                  // tamaño del archivo en bytes (4 bytes)
        reservado : array [0..5] bits;              // espacio reservado por el sistema (6
bits)
end; // 32 bytes

type fat = array [0..32767] of int;       // estructura de la FAT
var F : fat;
```

Notas generales:

- La variable F es global y siempre se encuentra cargada en memoria con el contenido de la FAT.
- Archivos y directorios son alojados en la región de datos usando la estructura de la FAT. El directorio raíz comienza en el primer sector de la región de datos.
- La FAT utiliza números físicos de sectores. El valor 0 indica un sector libre y el 1 indica el fin de un archivo o directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente. Pueden utilizar funciones auxiliares. La implementación debe ser eficiente.

Por otra parte se dispone de los siguientes procedimiento:

- Procedure **leerSect**(sec_num: int, Var buff: sector, Var ok: boolean); //Lee de disco el sector sec_num, pasado como parámetro, y retorna el contenido leído en el parámetro de salida buff. En el parámetro ok se retorna el éxito de la ejecución de la operación.
- Procedure **parteCamino**(camino: array of char, var base: array of char, var resto: array of char); //Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto. Por ejemplo: parteCamino('Users\sistoper\a.txt', 'Users', 'sistoper\a.txt'); parteCamino('a.txt', 'a.txt', '');
- Procedure **parteNombre**(nomext: array of char, var nom: array of char, var ext: array of char); //Recibe un nombre completo de archivo o directorio, y retorna su nombre en nom por un lado y su extensión en ext por otro. Por ejemplo: parteNombre('a.txt', 'a', 'txt'); parteNombre('sistoper', 'sistoper', '');

**Enunciado B)**
Sea un sistema de archivos FAT que cuenta con la siguiente organización en el disco:

| Contenido | Sector de arranque | Comienzo de la FAT | Comienzo del sector de datos |
|---|---|---|---|
| Sector | 0 | 1 | 35 |

El sector de arranque comienza en el sector 0, la FAT en el sector 1, y la región de datos en el sector 35. Este sistema cuenta con las siguientes estructuras de datos:

```
type sector = array [0..32767] of byte;
type entrada_dir = record
        usado : bit;                                  // entrada usada o no (1 bit)
        nom : array [0..19] of char;                  // nombre del elemento (20 bytes)
        ext : array [0..2] of char;                   // extensión del elemento (3 bytes)
        tipo : {0=ARCHIVO, 1=DIRECTORIO};             // tipo de elemento (1 bit)
        inicio : int;                                 // dirección de comienzo (4 bytes)
        tam : int;                                    // tamaño del archivo en bytes (4 bytes)
        reservado : array [0..5] bits;                // espacio reservado por el sistema (6
bits)
end; // 32 bytes
type fat = array [0..65535] of int;       // estructura de la FAT
var F : fat;
```

Notas generales:

- La variable F es global y siempre se encuentra cargada en memoria con el contenido de la FAT.
- Archivos y directorios son alojados en la región de datos usando la estructura de la FAT. El directorio raíz comienza en el primer sector de la región de datos.
- La FAT utiliza números físicos de sectores. El valor 0 indica un sector libre y el 1 indica el fin de un archivo o directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente. Pueden utilizar funciones auxiliares. La implementación debe ser eficiente.

Por otra parte se dispone de los siguientes procedimiento:

- Procedure **leerSect**(sec_num: int, Var buff: sector, Var ok: boolean); //Lee de disco el sector sec_num, pasado como parámetro, y retorna el contenido leído en el parámetro de salida buff. En el parámetro ok se retorna el éxito de la ejecución de la operación.
- Procedure **parteCamino**(camino: array of char, var base: array of char, var resto: array of char); //Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto. Por ejemplo: parteCamino('Users\sistoper\a.txt', 'Users', 'sistoper\a.txt'); parteCamino('a.txt', 'a.txt', '');
- Procedure **parteNombre**(nomext: array of char, var nom: array of char, var ext: array of char); //Recibe un nombre completo de archivo o directorio, y retorna su nombre en nom por un lado y su extensión en ext por otro. Por ejemplo: parteNombre('a.txt', 'a', 'txt'); parteNombre('sistoper', 'sistoper', '');

**Enunciado C)**
Sea un sistema de archivos FAT que cuenta con la siguiente organización en el disco:

| Contenido | Sector de arranque | Comienzo de la FAT | Comienzo del sector de datos |
|---|---|---|---|
| Sector | 0 | 1 | 43 |

El sector de arranque comienza en el sector 0, la FAT en el sector 1, y la región de datos en el sector 43. Este sistema cuenta con las siguientes estructuras de datos:

```
type sector = array [0..2047] of byte;
type entrada_dir = record
        usado : bit;                                  // entrada usada o no (1 bit)
        nom : array [0..19] of char;                  // nombre del elemento (20 bytes)
        ext : array [0..2] of char;                   // extensión del elemento (3 bytes)
        tipo : {0=ARCHIVO, 1=DIRECTORIO};             // tipo de elemento (1 bit)
        inicio : int;                                 // dirección de comienzo (4 bytes)
        tam : int;                                    // tamaño del archivo en bytes (4 bytes)
        reservado : array [0..5] bits;                // espacio reservado por el sistema (6
bits)
end; // 32 bytes
type fat = array [0..32767] of int;       // estructura de la FAT
var F : fat;
```

Notas generales:

- La variable F es global y siempre se encuentra cargada en memoria con el contenido de la FAT.
- Archivos y directorios son alojados en la región de datos usando la estructura de la FAT. El directorio raíz comienza en el primer sector de la región de datos.
- La FAT utiliza números físicos de sectores. El valor 0 indica un sector libre y el 1 indica el fin de un archivo o directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente. Pueden utilizar funciones auxiliares. La implementación debe ser eficiente.

Por otra parte se dispone de los siguientes procedimiento:

- Procedure **leerSect**(sec_num: int, Var buff: sector, Var ok: boolean); //Lee de disco el sector sec_num, pasado como parámetro, y retorna el contenido leído en el parámetro de salida buff. En el parámetro ok se retorna el éxito de la ejecución de la operación.
- Procedure **parteCamino**(camino: array of char, var base: array of char, var resto: array of char); //Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto. Por ejemplo: parteCamino('Users\sistoper\a.txt', 'Users', 'sistoper\a.txt'); parteCamino('a.txt', 'a.txt', '');
- Procedure **parteNombre**(nomext: array of char, var nom: array of char, var ext: array of char); //Recibe un nombre completo de archivo o directorio, y retorna su nombre en nom por un lado y su extensión en ext por otro. Por ejemplo: parteNombre('a.txt', 'a', 'txt'); parteNombre('sistoper', 'sistoper', '');

**Preguntas:**

1) ¿Cuántas entradas de directorio hay por sector?

2) Implemente la función buscar_arch que busca un archivo en el sistema de archivos. El cabezal de la función debe ser el siguiente:

Procedure **buscar_arch**(cam_arch: array of char, Var sector: int, Var indice: int, Var ok: bool);

El argumento cam_arch es la ruta completa al archivo. En los argumentos sector e índice debe indicarse el número de sector y el índice dentro del mismo donde encuentra la entrada de directorio (entrada_dir) del archivo, y en ok se debe retornar true si la operación fue ejecutada con éxito "y el archivo fue encontrado", o false en caso contrario.

---

**Solución pregunta 1)**

**Enunciado A)** Cada entrada de directorio ocupa $2^5$ bytes y cada sector tiene tamaño $2^{12}$ bytes, por lo tanto hay $2^7$ (128) entradas de directorio por sector.

**Enunciado B)** Cada entrada de directorio ocupa $2^5$ bytes y cada sector tiene tamaño $2^{15}$ bytes, por lo tanto hay $2^{10}$ (1024) entradas de directorio por sector.

**Enunciado C)** Cada entrada de directorio ocupa $2^5$ bytes y cada sector tiene tamaño $2^{11}$ bytes, por lo tanto hay $2^6$ (64) entradas de directorio por sector.

**Solución pregunta 2)**

**Enunciado A)**

```
Procedure buscar_arch(cam_arch: array of char, Var sector: int,
Var indice: int, Var ok: bool)
{
      var buff: array [0..4095] of byte;
      var dir_entries: array [0..127] of entrada_dir;
      var tipo: int;
      var nomext: array of char;
      var nombre: array of char;
      var ext: array of char;
      var i: int;

      ok = true;
      sector = 33;
      indice = -1;

      while (ok)
      {
            parteCamino(cam_arch, nomext, cam_arch);
            if (cam_arch == '')
            {
                  tipo = 0;
            }
            else
```

```
            {
                  tipo = 1;
            }
            while (indice == -1 && sector != 1)
            {
                  leerSect(sector, buff, ok);
                  if (!ok)
                  {
                        return;
                  }
                  dir_entries = (array [0..127] of entrada_dir)buff;
                  parteNombre(nomext, nombre, ext);
                  for(i=0; i < 128 && indice == -1; i++)
                  {
                        if (dir_entries[i].usado &&
dir_entries[i].tipo == tipo && dir_entries[i].nombre == nombre &&
dir_entries[i].ext == ext)
                        {
                              indice = i;
                        }
                  }
                  if (indice == -1)
                  {
                        sector = F[sector];
                  }
            }
            if (indice == -1)
            {
                  ok = false;
            }
            else
            {
                  if (tipo == 0)
                  {
                        return;
                  }
                  else
                  {
                        sector = dir_entries[indice].inicio;
                        indice = -1;
                  }
            }
      }
}
```