Apéndice complementario a la práctica 6
Periféricos Avanzados.

"EL - DISCO"

LOS DISCOS.

ESTRUCTURA FISICA.

Los discos son el principal medio de almacenamiento externo de los ordenadores compatibles. Pueden ser unidades de disco flexible, removibles, o discos duros -fijos-. Constan básicamente de una superficie magnética circular dividida en pistas concéntricas, cada una de las cuales se subdivide a su vez en cierto número de sectores de tamaño fijo. Como normalmente se emplean ambas caras de la superficie, la unidad más elemental posee en la actualidad dos cabezas de lectura/escritura, una para cada lado del disco. Los tres parámetros comunes a todos los discos son, por tanto: el número de cabezas, el de pistas y el de sectores. El término cilindro *i* hace referencia a la totalidad de las pistas *i* de todas las caras. Bajo DOS, los sectores tienen un tamaño de 512 bytes (tanto en discos duros como en disquetes) que es difícil cambiar (aunque no imposible). Los sectores se numeran a partir de 1, mientras que las pistas y las caras lo hacen desde 0. El DOS convierte esta estructura física de tres parámetros a otra: el número de *sector lógico*, que se numera a partir de 0 (los sectores físicos les denominaremos a partir de ahora *sectores BIOS* para distinguirlos de los sectores lógicos del DOS). Para un disco de *SECTPISTA sectores BIOS por pista* y *NUMCAB cabezas*, los sectores lógicos se relacionan con la estructura física por la siguiente fórmula:

Sector lógico = (sector_BIOS - 1) + cara * SECTPISTA + cilindro * SECTPISTA * NUMCAB - X1

Es decir, el DOS recorre el disco empezando la pista 0 (la exterior, la más alejada del centro) y por la cara o cabezal 0, recorriendo todos los sectores; luego avanza una cara y recorre de nuevo todos los sectores; después pasa al siguiente cilindro... y repite de nuevo el proceso. De esta manera, varios cabezales podrían -hipotéticamente- leer bloques de información consecutivos simultáneamente. En los disquetes, X1=0, pero en los discos duros se resta un cierto factor de compensación X1, ya que éstos pueden estar divididos en varias *particiones* y la que usa el DOS puede no estar al principio del mismo. En general, un disco duro dividido en varias particiones de tipo DOS determina varias unidades lógicas de disco, cada una de las cuales dispone de un conjunto de sectores lógicos numerados a partir de 0 y un factor de compensación propio para la fórmula. Las siguientes fórmulas transforman sectores DOS en sus correspondientes BIOS:

```
Sector_BIOS = (sector MOD SECTPISTA) + 1
Cara = (sector / SECTPISTA) MOD  NUMCAB
Cilindro = sector / (SECTPISTA * NUMCAB) + X2
```

Como la partición del DOS no suele empezar en el cilindro 0 (reservado en gran parte para la tabla de particiones) sino más bien en el 1 ó en otro posterior (cuando hay más particiones antes que la del DOS) será necesario añadir un cierto valor adicional de compensación X2 a la última fórmula para calcular el cilindro efectivo; esto es así porque en la práctica las particiones suelen empezar y acabar ocupando cilindros enteros y *exactos* (aunque en realidad, y dada la arquitectura de la tabla de partición, podrían empezar y acabar no sólo en un determinado cilindro sino también en cierto sector y

cara del disco, pero no es frecuente). X1 y X2 se obtienen consultando e interpretando la tabla de particiones o el sector de arranque.

CABEZA 0. PISTA 0. SECTOR 1.

El primer sector **físico** de todos los discos contiene información especial (el sector_BIOS 1 del cilindro 0 y cabezal 0). Tanto en disquetes como en discos duros, contiene un pequeño programa que se encarga de poner en marcha el ordenador: es el *sector de arranque* de los disquetes, o bien el código de la *tabla de particiones* de los discos duros. En este último caso, ese programa realiza una tarea muy sencilla: consulta la tabla de particiones ubicada en ese mismo sector, determina cuál es la *partición activa* y dónde empieza y acaba; a continuación carga el sector **lógico** 0 de esa partición (*sector de arranque*) y lo ejecuta. En los disquetes no existe este paso intermedio: el sector físico 0 del disquete, en terminos absolutos, es ya el sector de arranque y no el de partición. Esto es así porque los disquetes contienen poca información y son baratos, no siendo preciso particionarlos para compartirlos con varios sistemas operativos. El programa ubicado en el sector de arranque busca el fichero oculto del sistema IBMBIO.COM o IO.SYS, lo carga y le entrega el control. El programa contenido en este fichero cargará a su vez IBMDOS.COM o MSDOS.SYS, el cual a su vez cargará finalmente el intérprete de comandos (normalmente, COMMAND.COM).

* Formato de la tabla de partición de los discos duros:

160; Esta tabla comienza en un offset 1BEh del sector (al principio está el código ejecutable); cada partición de las 4 posibles ocupa 16 bytes; al final de las cuatro está la marca 0AA55h, ubicada en el offset 1FEh, que indica que la tabla es válida. Los 16 bytes que la forman se interpretan como indica el cuadro:

byte 0: 0 para partición inactiva, 80h en la de arranque.
byte 1: cabeza donde comienza la partición.
byte 2: bits 0 al 5: sector de inicio de la partición; 6, 7: parte alta del número de cilindro.
byte 3: parte baja del número de cilindro de inicio de la partición.
byte 4: tipo de partición, las más comunes son 0: No usada; 1: DOS-12 (FAT 12 bits); 4: DOS-16 (FAT 16 bits); 5: DOS Extendida; 6:BIGDOS (más de 32Mb); 7: OS/2 HPFS ó WinNT NTFS; 0Ah: OS/2 Boot Manager; 0Bh: 32-bit FAT Win95 (0Ch con LBA); 0Eh y 0Fh (como 06 y 05 pero con LBA); 81h Linux; 82h Linux swap; 83h: Linux native; 0A5h: FreeBSD o BSD/386; 0F2h: partición secundaria .
byte 5: cabeza donde termina la partición.
byte 6: bits 0 al 5: sector de fin de la partición; 6, 7: parte alta del número de cilindro.
byte 7: parte baja del número de cilindro de fin de la partición.
bytes 8 al 11: Doble palabra que indica el sector relativo (en todo el disco) en que comienza la partición, expresado en sectores.
bytes 12 al 15: Doble palabra con el tamaño de esa partición en sectores.

Formato de la TABLA DE PARTICIÓN

Habitualmente, las particiones suelen empezar en el segundo cabezal del cilindro 0, con lo que toda la primera pista física del disco duro está vacía. Lugar ideal para virus, algunos fabricantes han utilizado esta interesante característica para mejorar el arranque, colocando una falsa tabla de partición que muestre un menú en pantalla y cargue después la partición de verdad, permitiendo también más de 4 particiones. Sin embargo, estas maniobras suelen reducir la compatibilidad. Existen también código de particiones sofisticado que permite seleccionar una de las 4 particiones manteniendo pulsada una

tecla en el arranque, sin tener que andar ejecutando FDISK para seleccionar la partición activa... ¡lo que se puede hacer con 400 bytes de código!. Realmente, la arquitectura global de las particiones de un equipo (en particular si tiene más de 4, una mezcla de sistemas operativos y/o varios discos duros), puede llegar a ser compleja: practíquese con un buen editor de disco para aprender más (ej. el DISKEDIT de las Norton Utilities o las PC-Tools).

Las particiones extendidas llevan su propio sector de partición adicional, en el que no hay código de programa sino, en su lugar, una lista de dispositivos. Hay dos entradas por cada dispositivo: la primera indica el tipo (1-FAT12, 4-FAT16); la segunda entrada apunta al siguiente dispositivo (caso de existir) o es 0 (no hay más dispositivos). El DOS 4.0 y posteriores eliminaron la limitación de los 32 Mb en las particiones y el software actual, ya actualizado, no da problemas con los discos de más de 32 Mb. Por ello, en discos de más de 32 ó 40 Mb lo normal es instalar DOS 4.0 ó superior.

* Formato del sector de arranque:

En el sector de arranque, además del sencillo programa de puesta en marcha del sistema, hay cierta información útil acerca de las características del disco o partición. Los primeros 3 bytes no son significativos: contienen el código de operación de una instrucción JMP que salta a donde realmente comienza el código, aunque conviene que dicha instrucción de salto esté al principio del sector de arranque para que algunos sistemas validen dicho sector (es válido un salto corto seguido de NOP o un salto completo de 3 bytes). A partir del cuarto (offset 3) se puede encontrar la información válida. En el sector de arranque del disquete está contenido el BPB (Bios Parameter Block) que analizaremos más tarde.

offset 3 (8 bytes): Identificación del sistema (ej., "IBM 3.3")
offset 11 (1 palabra): Bytes por sector, ej. 512.
offset 13 (1 byte): Sectores por cluster (ej. 2)
offset 14 (1 palabra): Sectores reservados al principio (1 en diquettes)
offset 16 (1 byte): Número de copias de la FAT (2 normalmente)
offset 17 (1 palabra): Número de entradas al directorio raíz (112 en discos de 360 Kb)
offset 19 (1 palabra): Número total de sectores del disco (0 en discos de más de 32 Mb)
offset 21 (1 byte): Byte de tipo de disco (véase tabla más adelante)
offset 22 (1 palabra): Número de sectores ocupados por cada FAT
offset 24 (1 palabra): Número de sectores por pista
offset 26 (1 palabra): Número de cabezas (2 en disquetes de doble cara)
offset 28 (2 palabras): Número de sectores especiales reservados. Nota: sólo se debe considerar la primera mitad de esta doble palabra en versiones del sistema 3.30 o anteriores (no hay problemas con DR-DOS, que en todas sus versiones, hasta la 6.0 incluida, es un DOS 3.31). El valor de este campo depende de la posición relativa que ocupe la partición dentro del disco duro (será 0 en los disquetes), este valor ha de sumarse al del número de sector del DOS antes de traducirlo a un número de sector de la BIOS.
offset 32 (2 palabras): Número total de sectores del disco en discos de más de 32 Mb (esta información sólo debe obtenerse de aquí si la palabra ubicada en el offset 19 es cero).
offset 36 (1 byte): Número de unidad física (a partir del DOS 4.0).
offset 37 (1 byte): Reservado.
offset 38 (1 byte): valor 29h desde DOS 4.0 (marca de validación que indica que los bytes ubicados desde el offset 36 al offset 61 están definidos).
offset 39 (2 palabras): Número de serie del disco (a partir de DOS 4.0).
offset 43 (11 bytes): Título del disco (desde DOS 4.0); por defecto se inicializa con "NO NAME ", aunque tanto el DOS 4.0 como el 5.0 y 6.X siguen empleando además las tradicionales etiquetas de volumen.
offset 54 (8 bytes): Sistema de ficheros (a partir de DOS 4.0): puede ser "FAT12 " o "FAT16 ".

Formato del SECTOR DE ARRANQUE

El byte del tipo de disco (offset 21) intenta identificar el tipo de disco, aunque no lo consigue en muchos casos dada la ilógica utilización que se ha hecho de él. La

recomendación es hacer lo que viene haciendo el DOS desde la 3.30: no hacer caso de lo que dice este byte para identificar los discos. La única excepción tal vez sea el valor 0F8h que identifica a los dispositivos *no removibles*:

```
+-----------------------------------------------------------------+
| 0FEh - discos de 5¼-160 Kb (1 cara, 8 sectores/pista, 40 pistas)    |
| 0FFh - discos de 5¼-320 Kb (2 caras, 8 sectores/pista, 40 pistas)   |
| 0FCh - discos de 5¼-180 Kb (1 cara, 9 sectores/pista, 40 pistas)    |
| 0FDh - discos de 5¼-360 Kb (2 caras, 9 sectores/pista, 40 pistas)   |
| 0F9h - discos de 5¼-1,2 Mb (2 caras, 15 sectores/pista, 80 pistas)  |
| 0F9h - discos de 3½-720 Kb (2 caras, 9 sectores/pista, 80 pistas)   |
| 0F8h - discos duros y algunos virtuales                             |
| 0F0h - discos de 3½-1,44 Mb (2 caras, 18 sectores/pista, 80 pistas) |
| 0F0h - discos de 3½-2,88 Mb (2 caras, 36 sectores/pista, 80 pistas) |
| 0F0h - restantes formatos de disco                                  |
+-----------------------------------------------------------------+
```

Tipos de Discos

LA FAT.

Después del sector de arranque, aparecen en el disco una serie de sectores que constituyen la Tabla de Localización de Ficheros (File Alocation Table o FAT). Consiste en una especie de mapa que indica qué zonas del disco están libres, cuáles ocupadas, dónde están los sectores defectuosos, etc. Normalmente hay dos copias consecutivas de la FAT (véase el offset 16 del sector de arranque), ya que es el área más importante del disco de la que dependen todos los demás datos almacenados en él. No deja de resultar extraño que ambas copias de la FAT estén físicamente consecutivas en el disco: si accidentalmente se estropeara una de ellas (por ejemplo, rayando con un bolígrafo el disco) lo más normal es que la otra también resultara dañada. En general, muchos programas de chequeo de disco no se molestan en verificar si ambas FAT son idénticas (empezando por algunas versiones de CHKDSK). Por otra parte, hubiera sido mejor elección haberla colocado en el centro del disco: dada la frecuencia de los accesos a la misma, de cara a localizar los diferentes fragmentos de los ficheros, ello mejoraría notablemente el tiempo de acceso medio. Aunque cierto es que los *cachés* de disco y los *buffers* del config.sys pueden hacer casi milagros... a costa de memoria.

Antes de seguir adelante, conviene hacer un pequeño paréntesis y explicar el concepto de *cluster*: un cluster es la unidad mínima de información a la que accede el DOS, desde el punto de vista lógico. Normalmente consta de varios sectores (ver offset 13 del sector de arranque): dos en un disquete de 360 Kb, uno en un disquete de alta densidad, y entre 4 y 16 -normalmente- en un disco duro. El disco queda dividido, por tanto, en un cierto número de clusters. La FAT es realmente un mapa que contiene 12 ó 16 bits -como veremos- por cada cluster, indicando su estado:

```
          cluster libre: valor 0
          cluster defectuoso: valores 0FF7h (ó 0FFF7h).
          cluster no utilizable: valores 0FF5 al 0FF6h (ó 0FFF5 al
0FFF6h).
          último cluster del fichero: valor 0FF8 al 0FFFh (ó 0FFF8h al
0FFFFh).
          otro valor: puntero al siguiente cluster del fichero.
```

Los ficheros en disco no siempre ocupan posiciones contiguas: normalmente están más o menos fragmentados debido a que se aprovechan los huecos dejados por otros ficheros borrados, de ahí el auge de los programas que *compactan* los discos con objeto de acelerar el acceso a los datos. Por tanto, cada fichero consta de un cluster inicial indicado en la entrada del directorio -como se verá- que inicia una cadena tan larga

como la longitud del mismo (expresada en clusters), existiendo normalmente un valor 0FFFh ó 0FFFFh en el último cluster para señalar el final (del 0FF8h al 0FFEh y del 0FFF8h al 0FFFEh no se emplean). Consultando la FAT se puede determinar la ubicación de los fragmentos en que están físicamente divididos los ficheros en los discos, así como qué zonas están aún disponibles y cuáles son defectuosas en el mismo. Los cluster se numeran a partir de 2, ya que las dos primeras entradas en la FAT están reservadas para el sistema. Los clusters hacen referencia exclusiva a la zona de datos: el área que va detrás del sector de arranque, la FAT y el directorio. Por ello, en un disquete de 360 Kb, con clusters de 1 Kb y 354 Kb libres para datos, hay 354 clusters (numerados de 2 a 355) y los 6 Kb misteriosos que faltan son el sector de arranque, las dos FAT y -como veremos después- el directorio raíz. Puede ser válida, por ejemplo, la siguiente FAT de 12 bits habiendo un fichero A que ocupe los clusters 2, 3, 5 y 6:

Elemento de la FAT	**Valor**	**Interpretación**
0	FFD	El disco es de tipo 0FDh (despreciar restantes bits)
1	FFF	Entrada no utilizada
2	003	El siguiente cluster del fichero A es el 3
3	005	El siguiente cluster del fichero A es el 5
4	FF7	Cluster defectuoso
5	006	El siguiente cluster del fichero A es el 6
6	FFF	Este es el último cluster del fichero A
7	013	El siguiente cluster del fichero B es el 013
...	...	

Como se ve, el primer byte de la primera entrada a la FAT es inicializado con el mismo valor que el byte de tipo de disco del sector de arranque. Los restantes bits de las dos primeras entradas suelen estar todos a 1. Para determinar el número de clusters del disco, ha de restarse del número total de sectores la cifra correspondiente al número de sectores reservados (normalmente 1 en los disquetes, correspondiente al sector de arranque), los que ocupa la FAT y los empleados por el directorio raíz (que se verá más adelante); a continuación se divide ese número de sectores de datos resultante por el número de sectores por cluster.

El hecho de emplear FAT's de 12 bits es debido a que con menos bits (ej., un byte) sólo podría haber unos 250 clusters en el disco. En un disco de 1,2 Mb ello significaría que la unidad mínima de información sería 1200/250 = 5 Kb: el fichero más pequeño (de 1 byte) ocuparía ¡5 Kb!. Empleando FAT's de 16 bits se podrían hacer clusters incluso de tamaño menor que el sector (menos de 512 bytes), aprovechando más el espacio del disco. Sin embargo, ello haría que la propia FAT ocupase demasiado espacio en el disco. Por ello, en los disquetes se emplean FAT's de 12 bits (1 byte y medio): para un programa en código máquina ello no ralentiza los cálculos (aunque al ser humano no se le de muy bien trabajar con medios bytes). En la práctica, se toman palabras de 16 bits y se desprecian los 4 bits más significativos en los clusters pares y los 4 menos significativos en los impares.

A continuación se listan dos rutinas que permiten acceder a una FAT de 12 bits previamente cargada en memoria, con objeto de consultar o modificar alguna entrada.

Evidentemente, después habrá que volver a grabar la FAT en disco, tantas veces como copias de la misma existan en éste. Las rutinas necesitan que la FAT esté completamente cargada en memoria, lo cual no es un requerimiento demasiado costoso, habida cuenta de que no puede ocupar más de 4085 * 1,5 = 6128 bytes.

```
; ************ Escribir un elemento en una FAT de 12 bits
               ; Entrada: AX    = posición de dicho elemento
               ;          DS:BX = FAT completamente cargada en memoria
               ;          DX    = nuevo valor de dicho elemento

poke_fat       PROC
               PUSH  AX                ; preservar registros
               PUSH  BX
               PUSH  DX
               ADD   BX,AX             ; BX = BX + cluster
               SHR   AX,1              ; AX = cluster / 2
               PUSHF                   ; CF = 1 si impar
               ADD   BX,AX             ; BX = BX + cluster * 1,5
               MOV   AX,[BX]           ; AX = palabra con dato 12 bits
               POPF
               JC    poke_fat_imp
               AND   AX,1111000000000000b  ; preservar la otra entrada
               JMP   poke_fat_ok
poke_fat_imp:  AND   AX,0000000000001111b  ; preservar la otra entrada
               PUSH  CX
               MOV   CL,4
               SHL   DX,CL             ; colocarlo: 4 bits a la izda
               POP   CX
poke_fat_ok:   OR    AX,DX             ; «mezclar»
               MOV   [BX],AX           ; nuevo valor en la FAT
               POP   DX
               POP   BX
               POP   AX
               RET                     ; retorno sin alterar registros
poke_fat       ENDP

; ************ Leer un elemento de una FAT de 12 bits
               ; Entrada: AX    = posición de dicho elemento
               ;          DS:BX = FAT completamente cargada en memoria
               ; Salida:  DX    = valor de dicho elemento

peek_fat       PROC
               PUSH  AX                ; preservar registros
               PUSH  BX
               ADD   BX,AX             ; BX = BX + cluster
               SHR   AX,1              ; AX = cluster / 2
               PUSHF                   ; CF = 0 si par
               ADD   BX,AX             ; BX = BX + cluster * 1,5
               MOV   DX,[BX]
               POPF
               JNC   peek_fat_par
               PUSH  CX
               MOV   CL,4
               SHR   DX,CL             ; DX=DX/16: si DX=xyz0, DX=0xyz
               POP   CX
peek_fat_par:  AND   DH,00001111b      ; borrar posible dígito izdo
               POP   BX
               POP   AX
               RET                     ; retornar sólo DX modificado
peek_fat       ENDP
```

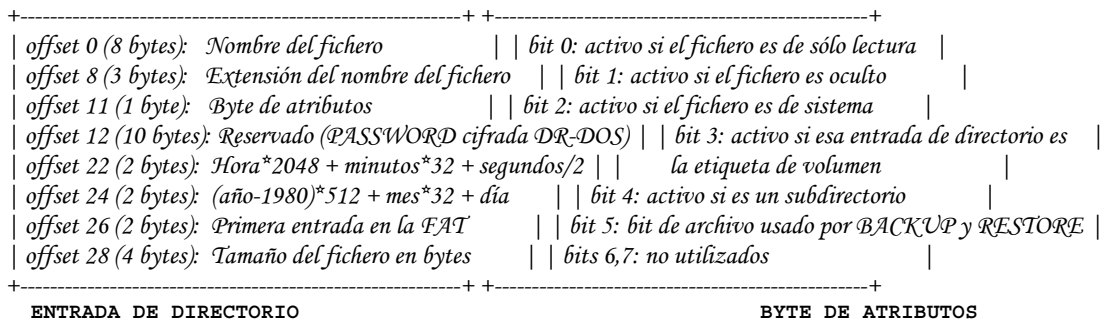
Tal vez, en futuros disquetes de elevada capacidad sea necesario pasar a una FAT de 16 bits, aparecida con el DOS 3.0, que es la usada por todos los discos duros excepto el de 10 Mb del XT original de IBM. Con una FAT de 12 bits el nº de cluster más alto posible es 4085, que se corresponde con un disco de 4084 clusters (numerados de 2 a

4085). En principio, no existe ninguna manera sencilla de averiguar el tipo de FAT de un disco, ya que el fabricante olvidó incluir un byte de identificación al efecto. La documentación publicada es contradictoria en las diversas fuentes que he consultado, y en todas es por desgracia incorrecta (unos dicen que la FAT 16 comienza a partir de 4078 clusters, otros que a partir de 4086, otros confunden el número de clusters con el número más alto de cluster...). Sin embargo, todas las versiones del DOS comprobadas (MS-DOS 3.1, 3.3, 4.0, 5.0 y DR-DOS 5.0 y 6.0) operan con una FAT de 16 bits en discos de 4085 clusters (inclusive) en adelante; esto es, a partir de 4086 como número de cluster más alto. Esto puede verificarse fácilmente creando discos virtuales con 4084/4085 clusters, copiando algunos ficheros y mirando la FAT con algún programa de utilidad (a simple vista se distingue si las entradas son de 12 ó 16 bits). Por desgracia, salvo en MS-DOS 3.3 y en DR-DOS 6.0, los comandos CHKDSK del sistema consideran erróneamente que los discos de 4085, 4086 y 4087 clusters ¡poseen una FAT de 12 bits!, lo cual resulta además completamente absurdo, dado que 4087 (0FF7h) es la marca de cluster defectuoso en una FAT de 12 bits y ¡en ningún caso podría ser un número de cluster cualquiera!. Sin embargo, pese a este problema de CHKDSK, los discos con más de 4084 clusters han de ser diseñados con una FAT de 16 bit, ya que es mucho más grave tener problemas con el DOS que con CHKDSK. Otra solución es procurar no crear discos de ese número crítico de clusters, o confiar que el usuario no ejecute el casi olvidado CHKDSK sobre ellos. Por fortuna, los discos normales no están por ahora en la frontera crítica entre la FAT de 12 y la de 16 bits, aunque con los discos virtuales sí se pueden crear unidades con esos tamaños críticos: la casi totalidad de los discos virtuales del mercado tienen problemas en estos casos. En algunos discos duros se puede determinar también el tipo de FAT consultando la tabla de particiones, aunque no es el método más conveniente. Debe tener en cuenta el lector que manipular una FAT sin conocer su tipo supone destrozar la información almacenada en el disco. Sin embargo, tampoco hay que tener tanto miedo: lo que sí puede resultar peligroso es llegar al extremo de preguntar al usuario el tipo de FAT...

Ahora puede surgir la pregunta: si la FAT mantiene una cadena que indica cómo está distribuido un fichero en el disco, ¿dónde se almacena el inicio de esa cadena, esto es, la primera entrada en la FAT del fichero?.

EL DIRECTORIO RAÍZ.

Inmediatamente después de la FAT y su(s) réplica(s) de seguridad viene el directorio raíz. Detrás de éste ya vienen los clusters conteniendo la información del disco propiamente dicha. El directorio consta de 32 bytes por cada fichero/subdirectorio (los subdirectorios no son más que un tipo especial de fichero). En los discos de 360 Kb, por ejemplo, el directorio se extiende a lo largo de 7 sectores (3584 bytes = 112 entradas como máximo). El tamaño y ubicación del directorio pueden obtenerse del sector de arranque, como se vio al principio. La información almacenada en los 32 bytes es la siguiente:



En el byte de atributos, varios bits pueden estar activos a un tiempo. El atributo de sistema no tiene un significado en particular, es una reliquia heredada del CP/M (los ficheros ocultos del sistema lo tienen activo). En un mismo disco sólo puede haber una entrada con el bit 3 activo; además, en este caso se interpretan el nombre y la extensión como un único conjunto de 11 caracteres. Las entradas de tipo subdirectorio (bit 4 del byte de atributos activo) tienen un valor cero en el campo de tamaño (offset 28): el tamaño de un fichero subdirectorio está determinado por el número de entradas que ocupa en la FAT (en la práctica, esto sucede con cualquier otro fichero, aunque si no es de directorio en el offset 28 esta información se indica con precisión de bytes).

El nombre del fichero puede comenzar por 0E5h, lo que indica que el fichero que estuvo ahí ha sido borrado. Si empieza por 2Eh (código ASCII del punto (.)) ó por 2Eh, 2Eh (dos puntos consecutivos) se trata de una entrada que referencia a un fichero subdirectorio.

LOS SUBDIRECTORIOS.

Como hemos visto, un subdirectorio en principio puede ser una simple entrada del directorio raíz. El subdirectorio, físicamente, es a su vez un fichero un tanto especial: contiene datos binarios ... que son nada más y nada menos que otras entradas de directorio para otros ficheros, de 32 bytes como siempre. Dentro de cada subdirectorio hay al menos dos entradas especiales: un fichero con un *nombre punto* (.) que referencia al propio subdirectorio -que así puede autolocalizarse- y otro con *doble punto* (..) que referencia al directorio padre -del que cuelga- siendo posible, gracias a ello, retroceder cuanto se desee por el árbol de directorios sin necesidad de que todos los caminos partan del raíz. Si la primera entrada en la FAT del fichero (..) es un 0, quiere decir que ese subdirectorio cuelga del raíz, de lo contrario apuntará al primer cluster del fichero subdirectorio padre.

El tamaño de un *fichero subdirectorio* es ilimitado -sin exceder, evidentemente, la capacidad del disco-. Por ello, en un subdirectorio puede haber una gran cantidad de ficheros (muchos más de 112 ó 500) sin problemas. Cada fichero que se crea en un subdirectorio aumenta el tamaño del fichero subdirectorio en 32 bytes. Por ello, en un disco de 360 Kb (354 Kb libres) se puede crear un subdirectorio y en él se pueden introducir, en caso extremo, 11326 ficheros (más el (.) y el (..)) de tamaño cero que paradójicamente llenarían el disco (recordar que cada entrada al directorio ocupa 32 bytes). Normalmente nadie suele cometer esos excesos. Si en un subdirectorio había demasiados ficheros y se borra una buena parte de los mismos, el tamaño del fichero

subdirectorio debería reducirse, pero en la práctica el DOS no se ocupa de estas pequeñeces, habida cuenta de que los ficheros subdirectorio son unos pequeños islotes en el gran océano disco (los usuarios más tacaños siempre pueden optar por crear un nuevo subdirectorio y mover todos los ficheros a él, borrando el anterior para recuperar el espacio libre).

Considerando el nombre completo de un fichero, con toda la trayectoria de directorios, el proceso a seguir para localizarlo en el disco es ir recorriendo los ficheros subdirectorio de uno en uno, hasta llegar al fichero subdirectorio donde está registrado el fichero y, en la posición correspondiente, obtener su punto de entrada en la FAT.

Dicho sea de paso, tal vez sea una pena que el disco no conste de un único «fichero raíz» privilegiado de directorio, que podríamos denominar «subdirectorio raíz». Ello permitiría también un número ilimitado de entradas (en vez de 112, 224, etc.) y sería más lógico que una ristra de sectores. Sin embargo, esta peculiar circunstancia también aparece en otros sistemas operativos, como el UNIX. Sus motivos tendrá.

EL BPB Y DPB.

El BPB (Bios Parameter Block) es una estructura de datos que contiene información relativa a la unidad de disco. El BPB es una pieza vital en los controladores de dispositivo de bloques, como veremos en un futuro capítulo, por lo que a continuación se expone su contenido (idéntico a una parte del sector 0):

```
+-------------------------------------------------------------------------+
|  offset  0    DW bytes_por_sector                                       |
|  offset  2    DB sectores_por_cluster                                   |
|  offset  3    DW sectores_reservados_al_comienzo_del_disco              |
|  offset  5    DB número_de_FATs                                         |
|  offset  6    DW número_de_entradas_en_el_directorio_raíz               |
|  offset  8    DW número_total_de_sectores (0 con n° de sector de 32 bits) |
|  offset 10    DB byte_descriptor_de_medio                               |
|  offset 11    DW numero_de_sectores_por_FAT                             |
|  -- A partir del DOS 3.0:                                               |
|  offset 13    DW sectores_por_pista                                     |
|  offset 15    DW número_de_cabezas                                      |
|  offset 17    DD número_de_sectores_ocultos                             |
|  -- A partir del DOS 4.0 (más bien DOS 3.31)                            |
|  offset 21    DD número_de_sectores (unidades con direccionamiento de   |
|                  sector de 32 bits)                                     |
|  offset 25    DB 6 DUP (?)    (6 bytes no documentados)                 |
|  offset 31    DW número_de_cilindros                                    |
|  offset 33    DB tipo_de_dispositivo                                    |
|  offset 34    DW atributos_del_dispositivo                              |
+-------------------------------------------------------------------------+
```

El DOS convierte internamente el BPB en DPB (Drive Parameter Block), una estructura similar con más información útil. Para obtener el DPB de una unidad determinada, puede utilizarse la función 32h del DOS, **Get Drive Parameter Block** (indocumentada); la cadena de DPBs del DOS puede recorrerse a partir del primer DPB (obtenido con la función 52h del DOS, **Get List of Lists**, también indocumentada).

LA BIOS Y LOS DISQUETES.

Resulta interesante conocer el comportamiento de la BIOS en relación a los disquetes, ya que las aplicaciones desarrolladas bajo DOS de una u otra manera habrán de cooperar con la BIOS por razones de compatibilidad (o al menos respetar ciertas especificaciones). El funcionamiento del disquete se controla a través de funciones de la

INT 13h, aunque esta interrupción por lo general acaba llamando a la INT 40h que es quien realmente gestiona el disco en las BIOS modernas de AT. Las funciones soportadas por esta interrupción son: reset del sistema de disco (reset del controlador de disquetes, envío del comando *specify* y recalibramiento del cabezal), consulta del estado del disco (obtener resultado de la última operación), lectura, escritura y verificación de sectores, formateo de pistas, obtención de información del disco y las disqueteras, detección del cambio de disco, establecimiento del tipo de soporte para formateo... algunas de estas últimas funciones no están disponibles en las máquinas PC/XT. La BIOS se apoya en varias variables ubicadas en el segmento 40h de la memoria. Estas variables son las siguientes.

Byte 40h:3Eh	Estado de recalibramiento del disquete. Esta variable indica varias cosas: si se ha producido una interrupción de disquete, o si es preciso recalibrar alguna disquetera debido a un reset anterior.
Byte 40h:3Fh	Estado de los motores. En esta variable se indica, además del estado de los motores de las 4 posibles disqueteras (si están encendidos o no), la última unidad que fue seleccionada y la operación en curso sobre la misma.
Byte 40h:40h	Cuenta para la detención del motor. Este byte es decrementado por la interrupción periódica del temporizador; cuando llega a 0 todos los motores de las disqueteras (realmente, el único que estaba girando) son detenidos. Dejar el motor girando unos segundos tras la última operación evita tener que esperar a que el motor acelere antes de la siguiente (si esta llega poco después).
Byte 40h:41h	Estado de la última operación: se actualiza tras cada acceso al disco, indicando los errores producidos (0 = ninguno).
Bytes 40h:42h	A partir de esta dirección, 7 bytes almacenan el resultado de la última operación de disquete o disco duro. Se trata de los 7 bytes que devuelve el NEC765 tras los principales comandos.
Byte 40h:8Bh	Control del soporte (AT). Esta variable almacena, entre otros, la última velocidad de transferencia seleccionada.
Byte 40h:8Fh	Información del controlador de disquete (AT). Se indica si la unidad soporta 80 cilindros (pues sí, la verdad) y si soporta varias velocidades de transferencia.
Byte 40h:90h	Estado del soporte en la unidad A. Se indica la velocidad de transferencia a emplear en el disquete introducido en esta unidad, si precisa o no saltos dobles del cabezal (caso de los disquetes de 40 cilindros en unidades de 80), y el resultado de los intentos de la BIOS (la velocidad puede ser correcta o no, según se haya logrado determinar el tipo de soporte).
Byte 40h:91h	Lo mismo que el byte anterior, pero para la unidad B.
Byte 40h:92h	Estado del soporte en la unidad A al inicio de la operación.
Byte 40h:93h	Estado del soporte en la unidad B al inicio de la operación.
Byte 40h:94h	Número de cilindro en curso en la unidad A.
Byte 40h:95h	Número de cilindro en curso en la unidad B.

Además de estas variables, la BIOS utiliza también una tabla de parámetros apuntada por la INT 1Eh. Los valores para programar ciertas características del FDC según el tipo de disco pueden variar, aunque algunos son comunes. Esta tabla determina las principales características de operación del disco. Dicha tabla está inicialmente en la ROM, en la posición 0F000h:0EFC7h de todas las BIOS compatibles (prácticamente el 100%), aunque el DOS suele desviarla a la RAM para poder actualizarla. El formato de la misma es:

byte 0: Se corresponde con el byte 1 del comando 'Specify' del 765, que indica el *step rate* (el tiempo de acceso cilindro-cilindro, a menudo es 0Dh = 3 ó6 ms) y el *head unload time* (normalmente, 0Fh = 240 ó480 ms).

byte 1: Es el byte 2 del comando 'Specify': los bits 7..1 indican el *head load time* (normalmente 01h = 2 ó4 ms) y el bit 0 suele estar a 0 para indicar modo DMA.

byte 2: Tics de reloj (pulsos de la interrupción 8) que transcurren tras el acceso hasta que se para el motor.

byte 3: Bytes por sector (0=128, 1=256, 2=512, 3=1024).

byte 4: Sectores por pista.

byte 5: Longitud del *GAP* entre sectores (normalmente 2Ah en unidades de 5¼ y 1Bh en las de 3½).

byte 6: Longitud de sector (ignorado si el byte 3 no es 0).

byte 7: Longitud del GAP 3 al formatear (80 en 5¼ y 3½-DD, 84 en 5¼-HD y 108 en 3½-HD).

byte 8: Byte de relleno al formatear (normalmente 0F6h).

byte 9: Tiempo de estabilización del cabezal en ms.

byte 10: Tiempo de aceleración del motor (en unidades de 1/8 de segundo).

El tiempo de estabilización del cabezal es el tiempo que hay que esperar tras mover el cabezal al cilindro adecuado, hasta que éste se *asiente*, con objeto de garantizar el éxito de las operaciones futuras; esta breve pausa es establecida en 25 milisegundos en la BIOS del PC original, aunque otras BIOS y el propio DOS suelen bajarlo a 15. Del mismo modo, el tiempo de aceleración del motor (byte 10) es el tiempo que se espera a que el motor adquiera la velocidad de rotación correcta, nada más ponerlo en marcha. En cualquier caso, es norma general intentar tres veces el acceso a disco (con resets de por medio) hasta considerar que un error es real. En general, pese a estos valores usuales, la flexibilidad del sistema de disco es extraordinaria y suele responder favorablemente con unos altísimos niveles de tolerancia en las temporizaciones. Una excepción quizá la constituye el valor de *GAP* empleado al formatear, al ser un parámetro demasiado importante.

DISQUETES FLOPTICAL 3½ DE 20 MB.

Las unidades que soportan estos disquetes, que también admiten los de 720K y 1.44M (aunque a menudo no los de 2.88M) trabajan con controladoras SCSI e incorporan una BIOS propia para dar soporte a estos dispositivos. El secreto de estos disquetes está en el posicionamiento óptico del cabezal, lo que permite elevar notablemente el número de pistas. Por ejemplo, las unidades de 20 Mb parecen estar equipadas con 753 cilindros y 27 sectores/pista. Aunque en el sector de arranque indica

que posee 251 cilindros y 6 cabezales, el sentido común nos permite deducir que esto no puede ser así. Lo de los 27 sectores por pista parece indicar que la velocidad de transferencia de estos disquetes es exactamente un 50% mayor que la de los convencionales de 1.44M (750 Kbit/seg frente a 500 Kbit/seg).

El FORMAT del DOS 5.0 y posteriores puede formatear los disquetes floptical, pero lo hace a bajo nivel, con lo que tarda cerca de 30-45 minutos en inicializarlos. Como ya vienen formateados de fábrica, en realidad basta con añadirles un sector de arranque e inicializar la FAT y el directorio raíz. También se puede verificar la superficie magnética para detectar posibles sectores defectuosos. Los programas de utilidad que acompañan estas unidades realizan todas estas tareas en unos 4 minutos. El tipo de FAT asignado puede ser seleccionado por el usuario (12 ó 16 bits), así como otros parámetros técnicos (tamaño de clusters, etc.).

Las tarjetas controladoras suelen permitir un cierto grado de flexibilidad, de cara a seleccionar la letra de unidad que se desea asignar al floptical. Configurándolo como A: se puede incluso arrancar desde un disquete de éstos.

EJEMPLO DE ACCESO AL DISCO A ALTO NIVEL.

Se puede acceder a varios niveles, siendo mejor el más alto por razones de compatibilidad:

1) Programando directamente el controlador de disquetes/disco duro para acceder a sectores físicos.
2) Llamando a la BIOS para leer cierto sector, de cierta cara y cierto cilindro.
3) Llamando al DOS para leer un sector lógico determinado en la unidad que se le indique.
4) Llamando al DOS para acceder a un fichero por su nombre y ruta.

El método (1) es apropiado para realizar formateos especiales en sistemas de protección anticopia; el (2) es útil para acceder a otras particiones de otros sistemas operativos o a disquetes formateados por otros sistemas operativos; las opciones (3) y (4) son las más cómodas e interesantes. En general, en la medida de lo posible es conveniente no bajar del nivel (3); de lo contrario se pierde la posibilidad de acceder a ciertas unidades (por ejemplo, un disco virtual no existe en absoluto para la BIOS).

A continuación se muestra un programa de ejemplo que solicita el nombre de un fichero y lo visualiza por pantalla, cargándolo por fragmentos y apoyándose en las funciones del DOS que se comentan en el apéndice que resume las funciones del sistema operativo. Paradójicamente, el acceso se realiza a alto nivel pese a tratarse de un programa en ensamblador. Como se puede observar, al final del programa se definen dos buffers de datos de 80 y 2048 bytes. Si no se desea que estos buffers alarguen el tamaño del programa ejecutable, pueden definirse de la siguiente manera:

```
fichnom     EQU     $
buffer       EQU     $+80
```

Sin embargo, si se procede de esta última manera convendría asegurarse primero de que existen 2128 bytes de memoria libres tras el código del programa, ya que de esta

manera el DOS no realiza la comprobación por nosotros (se limita a cargar cualquier programa que quepa en memoria). De todas maneras, normalmente suele haber más de 2128 bytes libres de memoria tras cargar cualquier programa... Conviene hacer notar que si en lugar de DUP (0) se coloca DUP (?), el linkador de Borland (TLINK 3.0), al contrario que el LINK de Microsoft, TAMPOCO reserva espacio efectivo para esas variables. Esto sólo sucede, lógicamente, cuando el DUP (?) está al final del programa y no hay nada más a continuación -ni más código ni datos que no sean DUP (?)-.

```
; ********************************************************************
; *                                                                  *
; *  MIRA.ASM  -  Utilidad para visualizar ficheros de texto.        *
; *                                                                  *
; ********************************************************************

mira            SEGMENT
                ASSUME CS:mira, DS:mira

                ORG   100h              ; programa de tipo .COM
inicio:
                LEA   DX,input_txt      ; mensaje
                MOV   AH,9              ; función de impresión
                INT   21h               ; llamar al DOS
                LEA   DX,fichnom        ; dirección para el «input»
                MOV   BYTE PTR [fichnom],60  ; no más de 60 caracteres
                MOV   AH,10             ; función de entrada de teclado
                INT   21h               ; llamar al DOS
                MOV   BL,[fichnom+1]    ; longitud efectiva tecleada
                MOV   BH,0              ; en BX
                ADD   BX,OFFSET fichnom ; apuntar al final
                MOV   BYTE PTR [BX+2],0 ; poner un cero al final

                LEA   DX,fichnom+2      ; offset a cadena ASCIIZ nombre
                MOV   AL,0              ; modo de lectura
                MOV   AH,3Dh            ; función para abrir fichero
                INT   21h               ; llamar al DOS
                JC    error             ; CF=1 --> error
                MOV   handle,AX         ; código de acceso al fichero

trocito:        MOV   BX,handle         ; código de acceso al fichero
                MOV   CX,2048           ; número de bytes a leer
                LEA   DX,buffer         ; dirección del buffer
                MOV   AH,3Fh            ; función para leer del fichero
                INT   21h               ; llamar al DOS
                JC    error             ; CF=1 --> error
                MOV   CX,AX             ; bytes leídos realmente
                JCXZ  cerrar            ; no hay nada que imprimir
                PUSH  AX                ; preservarlos
                LEA   BX,buffer         ; imprimir buffer ...
imprime:        MOV   DL,[BX]           ; carácter a carácter
                MOV   AH,2              ; ir llamando al servicio 2 del
                INT   21h               ; DOS para imprimir en pantalla
                INC   BX                ; siguiente carácter
                LOOP  imprime           ; acabar caracteres
                POP   AX                ; recuperar n° de bytes leídos
                CMP   AX,2048           ; ¿leidos 2048 bytes?
                JE    trocito           ; sí, leer otro trocito más

cerrar:         MOV   BX,handle         ; código de acceso al fichero
                MOV   AH,3Eh            ; cerrar fichero
                INT   21h               ; llamar al DOS
                JC    error             ; CF = 1 --> error
                INT   20h               ; fin del programa

error:          LEA   DX,fallo_txt      ; mensaje de error
                MOV   AH,9              ; función de impresión
                INT   21h               ; llamar al DOS
```

```
                CMP   handle,0          ; ¿fichero abierto?
                JNE   cerrar            ; sí: cerrarlo
                INT   20h               ; fin del programa

; ------------ datos y variables

handle          DW    0            ; handle de control del fichero
input_txt       DB    13,10,"Nombre del fichero: $"
fallo_txt       DB    13,10,"*** Error ***",13,10,10,"$"
fichnom         DB    80 DUP (0)   ; buffer para leer desde el teclado
buffer          DB    2048 DUP (0) ;   "     "     "     "  el disco

mira            ENDS
                END   inicio
```

EJEMPLO DE ACCESO AL DISCO A BAJO NIVEL.

El programa de ejemplo desarrollado requiere un adaptador VGA ya que utiliza el modo de 640 por 480 con 16 colores para obtener una representación gráfica de alta calidad del contenido del disco, en lugar de la tradicional y pobre representación habitual en modo texto. Además, se reprograman los registros de paleta y el DAC de la VGA para elegir colores más atractivos. El funcionamiento del programa se basa en acceder a la FAT y crear una imagen gráfica de la misma. Para ello, calcula cuantos puntos de pantalla debe trazar por cada cluster de disco (utiliza una ventana de 636x326 = 207336 puntos). Aunque este número no es entero, por razones de eficiencia se trabaja con fracciones para evitar el empleo de coma flotante. Muchas veces el ensamblador no es suficiente para asegurar la velocidad: la primera versión del programa tardaba 18 segundos en dibujar un mapa en un 386-25, con una rutina escrita en su mayor parte en ensamblador. Tras mejorar el algoritmo y optimizar el código en la zona crítica donde se trazan los puntos, se redujo a menos de 0,66 segundos el tiempo necesario (¡314000 puntos por segundo a 25 MHz!). Para leer los sectores del disco no se utiliza la función absread() del Borland C 2.0, ya que posee una errata por la que falla con unidades de más de 32767 clusters. En su lugar, una rutina en ensamblador se encarga de llamar a la interrupción 25h teniendo cuidado con el tipo de disco (particiones de más de 32 Mb o de menos de esa cantidad). La FAT se lee en una matriz, ya que no ocupa más de 128 Kb en el peor de los casos. Se lee de tres veces para evitar que en un sólo acceso a disco, vía INT 25h, se rebasen los 64 Kb permitidos si la FAT ocupa más de 64 Kb (el puntero al buffer apunta al inicio del segmento al ser de tipo HUGE). A continuación, se interpreta la FAT (según sea de 12 ó 16 bits) y se crea otra matriz de tamaño equivalente al número de clusters del disco. Esta última matriz -que indica los clusters libres, ocupados y defectuosos- es la que se volcará en pantalla adecuadamente. El programa también imprime información general sobre el disco, utilizando la función de impresión de la BIOS. Se imprime todo lo necesario antes de dibujar ya que para trazar los puntos es preciso programar el adaptador de vídeo de una manera diferente a la que emplea la BIOS (por razones de velocidad): después de ejecutar prepara_punto(), la BIOS no es capaz de escribir en pantalla

EL PSP.

Como se vio en el capítulo anterior, antes de que el COMMAND.COM pase el control al programa que se pretende ejecutar, se crea un bloque de 256 bytes llamado **PSP** (*Program Segment Prefix*), cuya descripción detallada se da a continuación.

La dirección del PSP en los programas COM viene determinada por la de cualquier registro de segmento (CS=DS=ES=SS) nada más comenzar la ejecución del mismo. Sin embargo, en los programas de tipo EXE sólo viene determinada por DS y ES. En cualquier caso, existe una función del DOS para obtener la dirección del PSP, cuyo uso recomienda el fabricante del sistema en aras de una mayor compatibilidad con futuras versiones del sistema operativo. La función es la 62h y está disponible a partir del DOS 3.0.

En la siguiente información, los campos del PSP que ocupen un byte o una palabra han de interpretarse como tal; los que ocupen 4 bytes deben interpretarse en la forma segmento:offset. En negrita se resaltan los campos más importantes.

- offsets 0 al 1: palabra 20CDh, correspondiente a la instrucción INT 20h. En CP/M se podía terminar un programa ejecutando un salto a la posición 0. En MS-DOS, un programa COM ¡también!.

- **offsets 2 al 3**: una palabra con la dirección de memoria (segmento) del último párrafo disponible en el sistema. Teniendo en cuenta dónde acaba la memoria y el punto en que está cargado nuestro programa, no es difícil saber la memoria que queda libre. Supuesto ES apuntando al PSP:

```
MOV     AX,ES:[2]    ; párrafo más alto disponible
MOV     CX,ES        ; segmento del PSP
SUB     AX,CX        ; AX = párrafos libres
MOV     CX,16
MUL     CX           ; DX:AX bytes libres
```

- offset 4: no utilizado.

- offsets 5 al 9: salto al despachador de funciones del DOS (en CP/M se ejecutaba un CALL 5, el MS-DOS ¡también lo permite!). No es recomendable llamar al DOS de esta manera. Los PSP creados por la función 4Bh en algunas versiones del DOS no tienen correctamente inicializado este campo.

- offsets 0Ah al 0Dh: contenido previo del vector de terminación (INT 22h).

- offsets 0Eh al 11h: contenido previo del vector de Ctrl-Break (INT 23h).

- offsets 12h al 15h: contenido previo del vector de manipulación de errores críticos (INT 24h).

- offsets 16h al 17h: segmento del PSP padre.

- offsets 18h al 2Bh: tabla de trabajo del sistema con los ficheros (Job File Table o JFT) : un byte por handle (a 0FFh si cerrado; los primeros son los dispositivos CON, NUL, ... y siempre están abiertos). Sólo hasta 20 ficheros (si no, véase offset 32h).

- **offsets 2Ch al 2Dh**: desde el DOS 2.0, una palabra que apunta al segmento del espacio de entorno, donde se puede encontrar el valor de variables de entorno tan interesantes como PATH, COMSPEC,... y hasta el nombre del propio programa que se está ejecutando en ese momento y el directorio de donde se cargó (no siempre es el

actual; el programa pudo cargarse, apoyándose en el PATH, en cualquier otro directorio diferente del directorio en curso). Véase el capítulo 8 para más información de las variables de entorno.

- offsets 2Eh al 31h: desde el DOS 2.0, valor de SS:SP en la entrada a la última INT 21h invocada.

- offsets 32h al 33h: desde el DOS 3.0, número de entradas en la JFT (por defecto, 20).

- offsets 34h al 37h: desde el DOS 3.0, puntero al JFT (por defecto, PSP:18h). Desde el DOS 3.0 puede haber más de 20 ficheros abiertos a la vez gracias a este campo, que puede ser movido de sitio. Sin embargo, es sólo a partir del DOS 3.3 cuando en un PSP hijo (por ejemplo, creado con la función EXEC) se copia la información de más que de los 20 primeros ficheros, si hay más de 20. Se puede saber si un fichero es remoto (en la MS-net) comprobando si el byte de la JFT está comprendido entre 80h-0FEh, aunque es mejor siempre acceder antes a las funciones del DOS.

- offsets 38h al 3Bh: desde el DOS 3.0, puntero al PSP previo (por defecto, 0FFFFh:0FFFFh en las versiones del DOS 3.x); es utilizado por SHARE en el DOS 3.3.

- offsets 3Ch al 3Fh: no usados hasta ahora.

- offsets 40h al 41h: desde el DOS 5.0, versión del sistema a devolver cuando se invoca la función 30h.

- offsets 42h al 47h: no usados hasta ahora.

- offset 48h: desde Windows 3, el bit 0 está activo si la aplicación es no-Windows.

- offsets 49h al 4Fh: no usados hasta ahora.

- offsets 50h al 52h: código de INT 21h/RETF. No recomendado hacer CALL PSP:5Ch para llamar al DOS.

- offsets 53h al 5Bh: no usados hasta ahora.

- offsets 5Ch al 7Bh: apuntan a los dos FCB's (File Control Blocks) usados antaño para acceder a los ficheros (uno en 5Ch y el otro en 6Ch). Es una reliquia en desuso, y además este área no se inicializa si el programa es cargado en memoria superior con el comando LOADHIGH del MS-DOS 5.0 y posteriores, por lo que no conviene usarlo ni siquiera para captar parámetros, al menos en programas residentes -susceptibles de ser instalados con LOADHIGH-. Si se utiliza el primer FCB se sobreescribe además el segundo.

- offsets 7Ch al 7Fh: no usados hasta ahora.

- **offsets 80h al 0FFh**: es la zona donde aparecen los parámetros suministrados al programa. El primer byte indica la longitud de los parámetros, después vienen los mismos y al final un retorno de carro (ASCII 13) que es un tanto redundante -a fin de cuentas, ya se sabe la longitud de los parámetros-. Ese retorno de carro, sin embargo, no

«se cuenta» en el byte que indica la longitud. Téngase en cuenta que no son mayusculizados automáticamente (están tal y como los tecleó el usuario), y además los parámetros pueden estar separados por uno **o más** espacios en blanco o tabuladores (ASCII 9).

En general, comprobar los valores que recibe el PSP cuando se carga un programa es una tarea que se realiza de manera sencilla con el programa DEBUG/SYMDEB. Para ello basta una orden tal como "DEBUG PROGRAMA.COM HOLA /T": al entrar en el DEBUG (o SYMDEB) basta con hacer «D 0» para examinar el PSP de PROGRAMA. Para ver los parámetros (HOLA /T en el ejemplo) se haría «D 80».