Ejercicio 3. En la siguiente figura se representa una tabla *FAT*. Al borde de sus entradas se ha escrito, como ayuda de referencia, el número correspondiente al bloque en cuestión. También se ha representado la entrada de cierto directorio. Como simplificación del ejemplo, suponemos que en cada entrada del directorio se almacena: Nombre de archivo/directorio, el tipo (F=archivo, D=directorio), la fecha de creación y el número del bloque inicial.

Tenga en cuenta que: - el tamaño de bloque es 512 bytes - el asterisco indica último bloque - todo lo que está blanco en la figura está libre

Nombre

Tipo

Fecha

Nº bloque

DATOS

F

8-2-90

3

DATOS1

F

1-3-90

4

DATOS2

F

2-3-90

5

D

D

3-3-90

18

CARTAS

F

13-3-90

12

Nº

Comienzo archivo

Contenido

1

2

3

DATOS

15

4

DATOS1

*

5

DATOS2

6

6

7

7

*

8

9

9

*

10

11

11

13

12

CARTAS

10

13

*

14

15

8

16

17

18

D

*

Ejercicio 4. Si usamos un Mapa de Bits para la gestión del espacio libre, especifique la sucesión de bits que contendría respecto a los 18 bloques del ejercicio anterior.

`00111111111101001`

Los 0 indican que el bloque correspondiente está vacío. Los 1 indican que el bloque correspondiente está ocupado.

Ejercicio 9. Un i-nodo de UNIX tiene 10 direcciones de disco para los diez primeros bloques de datos, y tres direcciones más para realizar una indexación a uno, dos y tres niveles. Si cada bloque índice tiene 256 direcciones de bloques de disco, cuál es el tamaño del mayor archivo que puede ser manejado, suponiendo que 1 bloque de disco es de 1KByte?

Tenemos que sumar por separado el tamaño máximo de un archivo direccionable por los 10 primeros bloques de datos y cada uno de los tamaños máximos correspondientes a la indexación de 1, 2 y 3 niveles.

$(10 \cdot 1KB + 256 \cdot 1KB + 256^2 \cdot 1KB + 256^3 \cdot 1KB = 16.843.018KB = 16.843018GB)$

Ejercicio 10. Sobre conversión de direcciones lógicas dentro de un archivo a direcciones físicas de disco. Estamos utilizando la estrategia de indexación a tres niveles para asignar espacio en disco. Tenemos que el tamaño de bloque es igual a 512 bytes, y el tamaño de puntero es de 4 bytes. Se recibe la solicitud por parte de un proceso de usuario de leer el carácter número N de determinado archivo. Suponemos que ya hemos leído la entrada del directorio asociada a este archivo, es decir, tenemos en memoria los datos PRIMERBLOQUE y TAMAÑO. Calcule la sucesión de direcciones de bloque que se leen hasta llegar al bloque de datos que posee el citado carácter.

Como cada bloque ocupa 512B y cada puntero ocupa 4B, cada bloque índice puede direccionar ($\frac{512}{4} = 128B$). Como se trata de un esquema en indexación a tres niveles y tenemos ya el primer bloque en memoria, necesitamos realizar tres accesos a disco. Esto quiere decir que es necesario conocer tres índices *i, j, k* para obtener el N-ésimo byte del archivo en cuestión.

$$i = \frac{N}{128 \cdot 128 \cdot 512} \quad N' = N\%(128 \cdot 128 \cdot 512)$$

$$j = \frac{N'}{128 \cdot 512} \quad N'' = N'\%(128 \cdot 512)$$

$$k = \frac{N''}{512} \quad N''' = N''\%(512)$$

Entonces, necesitamos traer de disco los bloques apuntados por las direcciones *i, j* y *k* y leer del bloque de datos con un desplazamiento igual a N'''.

Ejercicio 11. ¿Qué método de asignación de espacio en un sistema de archivos elegiría para maximizar la eficiencia en términos de velocidad de acceso, uso del espacio de almacenamiento y facilidad de modificación (añadir/borrar /modificar), cuando los datos son:

a) modificados infrecuentemente, y accedidos frecuentemente de forma aleatoria

b) modificados con frecuencia, y accedidos en su totalidad con cierta frecuencia

c) modificados frecuentemente y accedidos aleatoriamente y frecuentemente.

d) **Asignación contigua.** Como se accede infrecuentemente a ese archivo no es necesario mantener un método de asignación que permita modificarlo eficientemente. Además es necesario optimizar la velocidad de acceso directo y el espacio de almacenamiento. Es por esto que la mejor opción sería el método de asignación contigua.

e) **Asignación no contigua enlazada (tabla FAT).** Como accedemos al archivo en su totalidad, necesitamos un buen tiempo de acceso secuencial, junto con la posibilidad de modificar los datos. Esto es eficiente en espacio porque el archivo puede crecer dinámicamente sin necesidad de mantener los bloques en posiciones contiguas.

f) **Asignación no contigua indexada.** La asignación no puede ser contigua porque necesitamos facilitar la modificación de los datos. Esto garantiza una modificación eficiente porque el archivo puede crecer dinámicamente sin necesidad de mantener los bloques en posiciones contiguas. Además, la indexación garantiza accesos aleatorios rápidos.