Ejercicio

Sobre el disco duro de un computador personal se han instalado dos sistemas operativos de forma que, durante el arranque, se le consulta al usuario cuál de ellos desea usar. El disco duro está particionado como se muestra a continuación. El campo "punto de montaje" indica en que directorio se monta esa partición cuando se arranca en Linux.

Partición	**Tamaño**	**Tipo de SF**	**Punto de montaje**
/dev/sda1	128 GiB	FAT32	/media/WDWS
/dev/sda2	48 GiB	Linux UFS	/
/dev/sda3	8 GiB	Linux swap	
/dev/sda4	128 GiB	Linux UFS	/home

Para el caso de la primera partición, destinada a contener un SO Windows:

a) Dibuje y dimensione el *layout* de este sistema de ficheros. Indique el tamaño de cada una de sus partes. Justifique las decisiones que tome.
b) Considerando recién abierto un fichero de 4 GiB ¿cuántos accesos a disco serían necesarios para acceder al último byte de este fichero? Considere el mejor y el peor caso posibles.

Para el caso de la última partición, destinada a las cuentas de usuario en Linux:

c) Dibuje y dimensione el *layout* de este sistema de ficheros. Indique el tamaño de cada una de sus partes. Justifique las decisiones que tome.
d) Considerando recién abierto un fichero de 4 GiB ¿cuántos accesos a disco serían necesarios para acceder al último byte de este fichero?

Siendo `"/C:"` un enlace a `"/media/WDWS/windows/Documents and Settings/"`, esto es, al directorio con las cuentas de usuario en Windows, existiendo todas las rutas necesarias, y disponiendo el usuario de todos los derechos necesarios, considere que un proceso realiza la siguiente llamada: `creat("/C:/usuario/50GiB.dat", 0666);`

e) Determine la secuencia de entradas de directorio que serán analizadas durante la decodificación de la ruta, indicando a cada paso el criterio para decidir pasar de cada una a la siguiente.
f) Suponga que en el momento de realizarse la llamada `creat` dicho archivo existía, y que otro proceso estaba realizando la lectura secuencial de su contenido. ¿Cuál sería el efecto de dicha llamada sobre este segundo proceso y porqué?

SOLUCIÓN

a) /dev/sda1 de 128 GiB con FAT32:
[BB][FAT1][FAT2][ROOTDIR][RESTO]

[BB] El Bloque de Boot es un bloque.
[FAT] Consideremos una unidad de gestión de espacio (la agrupación) de un tamaño razonable, digamos 8 KiB.
La FAT contiene una dirección de agrupación por agrupación del disco. En el disco habrá:

2^37 (B/disco) / 2^13 (B/agrp) = 2^24 (agrp/disco)

Las direcciones de la FAT32 son de 32 bits, y por lo tanto, cada una de las dos copias de la FAT ocupará:

2^2 (B/dir) * 2^24 (dir/FAT) = 2^26 (B/FAT)
= 64 MiB (FAT) = 2^13 (agrp/FAT)

[ROOTDIR] El directorio raíz empieza donde terminan las copias de la FAT, su tamaño empieza en una agrupación, pero podrá crecer sobre otras indicadas en la FAT.
[RESTO] Como el nombre indica, el espacio restante, asignable a ficheros y directorios:

2^24 (agrp/disco) – 2*2^13 (agrp/FAT) – 1(agrp/rootd) =
= 128 GiB (disco) – 128 MiB (2 FATs) – 8 KiB (rootd)

Estas cifras indican que este sistema de archivos utiliza aproximadamente 1 parte entre 1024 para metadatos, esto es, estructuras de datos para gestionar los datos de los archivos de usuario.

b) Para acceder directamente al último byte de un archivo de 4 GiB hay que saber la agrupación del disco en que se encuentra, o dicho de otro modo, qué número de agrupación del disco es la agrupación (4 Gi (B/fich) / 8 Ki (B/agrp) =) 2^19 asignada al archivo.

Dado que la FAT contiene una lista encadenada de las agrupaciones asignadas a los archivos y considerando que recién abierto el archivo, la entrada de directorio nos informa de cuál es la primera agrupación que le corresponde, para conocer cual es la segunda, habrá que acceder a la FAT en la entrada indicada por el número de la primera agrupación. Con el número de la segunda agrupación accedemos a la FAT para localizar la tercera y así sucesivamente hasta localizar la entrada No 2^19. Esto supone 2^19 accesos a la FAT, pero no 2^19 accesos a disco dado que para algo está la cache de bloques.

Los casos mejor y peor se corresponden con los casos de acierto máximo y mínimo en la cache de bloques. Si el archivo fue ubicado próximo en el disco, sus agrupaciones formarán un grupo compacto y una zona muy próxima dentro de la FAT. En el mejor caso podrían ocupar: 2^19 (dir) * 2^2 (B/dir) / 2^13 (B/agrp) = 2^8 (agrp), esto es, sólo serán accesos a disco el equivalente en bloques de estas 256 agrupaciones.

El peor de los casos es cuando el archivo fue ubicado disperso sobre el disco (creció en diferentes momentos). En este caso, y como límite máximo podríamos considerar la necesidad de ir accediendo poco a poco hasta llevar a la cache de bloque toda la FAT. Esto suma accesos a disco por el equivalente en bloques de las 2^13 agrupaciones que ocupa la FAT.

c) /dev/sda4 de 128 GiB con UFS de Linux.:
[BB][SB][BME][BMI][VI][RESTO]

[BB] El Bloque de Boot es un bloque.
[SB] El Superbloque ocupa un bloque.
[BME] Este bitmap es para gestión de espacio libre, y la unidad de gestión es la agrupación. Para comparar, supondremos el mismo tamaño que para el caso anterior, agrupaciones de 8 KiB. Necesitamos un bit por cada agrupación del disco. todo ello ocupará: 2^24 (bits) / 2^3 (bits/B) / 2^13 (B/agrp) = 2^8 (agrp).
[BMI] Un bit por cada nodo-i del sistema de archivos. Preparamos un nodo-i para cada posible archivo de un tamaño medio de 1 agrupación. Esto nos da un número de nodos-i igual al número de agrupaciones y dos bitmaps de igual tamaño.
[VI] El vector de nodos-i tiene uno detrás de otro, espacio para contener todos los nodo-i que hemos previsto. Si los nodo-i son de 128 bytes: 2^24 (ino) * 2^7 (B/ino) = 2^31 B.
[RESTO] Siendo el vector de inodos la parte más voluminosa de los metadatos, podemos decir que este sistema de archivos consume aproximadamente algo más de 1 parte entre 64 en metadatos. El resto es espacio libre.

d) Como en el caso de la FAT anterior, para acceder directamente al último byte de un archivo de 4 GiB, es necesario localizar cuál es la agrupación donde está este byte en disco, o dicho de otro modo, qué número de agrupación del disco es la agrupación (4 Gi (B/fich) / 8 Ki (B/agrp) =) 2^19 asignada al archivo.

A diferencia de con la FAT, en este caso, la estructura de datos que nos asiste en la localización en disco de las agrupaciones asignadas a los archivos es el nodo-i, y esta estructura no precisa acceso secuencial. La capacidad de direccionamiento del nodo-i depende del número de direcciones que quepan en una agrupación y del número de niveles de indirección.

Direcciones directas: 10 (agrp)
Simple indirecto: 2^13 (B/agrp) / 2^2 (B/dir) = 2^11 (dir/agrp).
Doble indirecto: (2^11)^2 = 2^22 (agrp)
Triple indirecto: (2^11)^3 = 2^33 (agrp)

Para acceder a la agrupación número 2^19 de un archivo deberemos acceder a través del puntero doble indirecto de su nodo-i, y por lo tanto en sólo 3 accesos a disco habremos alcanzado el dato buscado.

e) En la tabla mostrada a continuación se analiza paso a paso las acciones realizadas durante la decodificación de la ruta derivada de la llamada creat.
Como se puede observar, la acción más común es, al llegar a un directorio, **profundizar**, pasando a buscar en él la siguiente componente de la ruta. Otras acciones menos usuales son: al paso por un enlace simbólico, **decodificar** (recursivamente) la ruta apuntada por el enlace y, al pasar por un directorio que sea punto de montaje, **saltar** al directorio raíz del sistema de archivos montado.

Dev	Dir	Buscar	Qué es	Acción
				decode(/C:/usuario/50GiB.dat)
sda2	/	C:	enlace	**decode**(/media/WDWS/windows/D&S/)
sda2	/	media	dir	
sda2	media	WDWS	dir	saltar al / de /dev/sda1
sda1	/	windows	dir	
sda1	windows	D&S	dir	**return**
sda1	D&S	usuario	dir	
sda1	usuario	50GiB.dat	file exist	**return**

f) La llamada al sistema `creat`, cuando no existe el archivo indicado, lo crea, pero si el archivo ya existe, como en este caso se indica, la llamada lo trunca. En ambos casos el archivo queda con tamaño 0 y en ambos casos la llamada termina bien.

Si esto sucede mientras un segundo proceso se encuentra leyendo el archivo de manera secuencial, y dado que esta es la política de coutilización UNIX, este segundo proceso se encontrará súbitamente con que el archivo pierde sus datos, y por lo tanto no habrá datos para leer, y su llamada de lectura `read` terminará, correctamente (sin error) devolviendo un 0, indicando fin de datos sobre ese descriptor. Lo normal es que este proceso concluya la lectura del fichero y continué con lo que sea, sin emitir ningún fallo por ello y termine adecuadamente.