Ejercicio de ficheros. Junio de 2007.

Se pretende modificar el sistema de ficheros tradicional de UNIX incorporando en el mismo la filosofía de que "cada parte del sistema de ficheros es un fichero". Con esta estrategia, todas las zonas que contienen metadatos (boot, superbloque, mapas de bits de bloques e inodos, y los inodos) son también ficheros, incluidos en el directorio raíz del sistema de ficheros (los ficheros se denominarán `/boot`, `/super`, `/mapa_bloques`, `/mapa_inodos` *e* `/inodos`, *respectivamente), tal que su contenido son los metadatos correspondientes. La nueva versión permite usar distintas disposiciones iniciales de las zonas de metadatos (exceptuando aquéllas zonas que tienen una ubicación fija por las propias restricciones del sistema, como ocurre con el boot), incluso la misma distribución que en la versión tradicional. Suponga un sistema de ficheros con las siguientes características:*

- *El tamaño del disco que lo soporta es de 4Gbytes.*
- *El tamaño del bloque es de 4Kbytes y el de las direcciones de bloque es de 32 bits.*
- *El tamaño del inodo es de 256 bytes con 16 punteros directos, uno indirecto simple, uno doble y uno triple.*
- *Tanto el boot como el superbloque ocupan 1 bloque.*
- *Hay una proporción de 1 inodo por cada 8192 bytes de espacio en disco.*

Se pide que responda a las siguientes cuestiones:

a) *En el caso de que se utilice el esquema tradicional, especifique cuál será la distribución en el disco de las distintas zonas del sistema de ficheros, indicando su tamaño y en qué número de bloque comienza cada una (se va a identificar el primer bloque de disco como "bloque 1"). Asimismo, detalle cuál es el contenido del disco después de crear un sistema de ficheros (*`mkfs`*), indicando qué bloques de datos e inodos están ocupados y cuál es su contenido (para los inodos especifique el valor de los punteros).*

b) *Suponiendo que se usa el nuevo esquema y que la disposición inicial de las distintas zonas del sistema de ficheros en el dispositivo es la misma que en la versión tradicional (misma ubicación y tamaño), especifique el contenido inicial del disco después de crear el sistema de ficheros indicando los mismos aspectos que en el apartado anterior (es decir, bloques e inodos ocupados y contenido de los mismos).*

c) *Considere el caso de que un programa quiera leer el bloque de boot. Para cada una de las versiones, especifique cómo lo haría, indicando qué llamadas al sistema de ficheros realizaría. ¿Se podría establecer que un cierto grupo de usuarios tuviera permiso de lectura del bloque de boot pero no pudiera acceder al resto del disco?*

d) *Suponga que en el momento de actualizar un bloque de un mapa de bits se detecta que el espacio de disco asociado al bloque está estropeado de forma permanente. Analice para ambos esquemas si el software del sistema de ficheros podría resolver el problema, sin un cambio radical en su diseño, y, en caso afirmativo, de qué manera lo haría.*

Analice a qué bloques del sistema de ficheros se accede (explicando previamente cómo se calcula cuáles son) en las siguientes llamadas en cada uno de los sistemas planteados, suponiendo que la caché de bloques está inicialmente vacía (para el nuevo sistema, considere además que ya están abiertos todos los ficheros correspondientes a zonas de metadatos). En primer lugar, analice el sistema tradicional y, a continuación, indique qué diferencias existirían en el nuevo sistema.

e) *La apertura de un fichero F, cuyo inodo es el 260 (se considera que el primer inodo es el 1), estando el inodo del directorio actual en memoria y teniendo dicho directorio un tamaño de 4KB.*

f) *La creación de un subdirectorio D en el directorio actual, suponiendo que el primer inodo libre disponible es el 70 (aunque el sistema de ficheros no lo sabe, evidentemente), y que el directorio actual, cuyo inodo está en memoria y tal que su tamaño es de 4KB, tiene espacio para la nueva entrada. Supóngase que los únicos bloques libres disponibles corresponden a bloques cuyo estado queda reflejado por el último bloque del mapa de bits de bloques.*

g) *Repita el apartado anterior, pero en el caso de que todos los inodos estén ocupados.*

h) *Por último, como reflexión final, analice qué ventajas, además de permitir usar distintas distribuciones iniciales de los metadatos, y desventajas tiene el nuevo esquema frente al tradicional.*

SOLUCIÓN

a) A continuación, se especifica el tamaño y la ubicación de cada una de las zonas de metadatos del sistema de ficheros planteado en el enunciado:

- El *boot* ocupa el primer bloque (B1).
- El superbloque ocupa el segundo bloque (B2).
- El mapa de bits de bloques requiere un bit por cada bloque del disco. En el disco hay 4GB/4KB=1M bloques. Por tanto, el mapa de bits de bloques ocupa 1Mbit, es decir, 128KB, que se corresponden con 128KB/4KB=32 bloques, desde el B3 al B34. Obsérvese que, dado que los bloques de metadatos están siempre ocupados, se podría plantear un mapa de bits que reflejará únicamente el estado de los bloques de datos propiamente dichos. Por tanto, habría dos alternativas:
  - Utilizar un mapa de bits para todos los bloques del disco, que ocuparía 32 bloques, donde todos los bits correspondientes a bloques de metadatos (es decir, los 32818 primeros bits) se marcarían como ocupados y la posición de cada bit dentro del mapa se correspondería con el número del bloque al que representa.
  - Usar un mapa de bits sólo para los bloques de datos, que tendría un tamaño de 31 bloques (se "ahorran" los 32818 bits que corresponden a bloques de metadatos, es decir, sólo un bloque). En este caso, el bit en la posición X representaría al bloque X+32818 y, por tanto, habría que sumar y restar esa cantidad para calcular la correspondencia entre bits y bloques.
- El mapa de bits de inodos tiene un bit por cada inodo. Dado que se plantea un sistema de ficheros con 1 inodo por cada 8K, habrá un total de 4G/8K=512K inodos. Por tanto, el mapa de bits de inodos ocupará 512Kbits=64KB, que requieren 64KB/4KB=16 bloques, desde el B35 al B50.
- La zona de inodos contiene 512K inodos cada uno de los cuales ocupa 256B, lo que requiere un total de 128MB, que se corresponden con 128MB/4KB=32K bloques, desde el B51 hasta el B32818 (=50+32K).
- La zona de datos comienza en el bloque B32819 y se extiende hasta el último bloque del disco (B1048576=2^{20}).

En cuanto al contenido inicial del disco, se habrá creado el directorio raíz que tendrá las siguientes características:

- Utilizará el primer nodo disponible (inodo 1, según el enunciado, aunque algunos sistemas UNIX usan el inodo 1 para enlazar los bloques defectuosos), que hará referencia en su primer puntero directo al bloque que contiene los datos del directorio, que se corresponderá con el primer bloque de la zona de datos (B32819).
- El bloque del directorio contendrá las entradas '`.`' y '`..`', tal que ambas harán referencia al inodo 1.

Habitualmente, en la mayoría de los sistemas UNIX, cuando se crea un sistema de ficheros, se incluye en el directorio raíz un subdirectorio denominado `/lost+found`, que usa la herramienta de reparación del sistema de ficheros (`fsck`) para almacenar en el mismo información que ha podido recuperar de un sistema de ficheros corrupto. Por tanto, en esos sistemas existe un segundo inodo (inodo 2) y bloque de datos (B32820) ocupados para almacenar dicho directorio, además de una tercera entrada en el directorio raíz que hace referencia el mismo. En el bloque de datos de este directorio se encontrarán las entradas '`.`' y '`..`', que en este caso harán referencia a los inodos 2 y 1, respectivamente.

b) Tal como se plantea en el enunciado, dentro de las diversas disposiciones iniciales que permite el nuevo esquema, consideramos una en la que la ubicación de las zonas de metadatos sea la misma que en el modelo tradicional.

Por lo que se refiere al contenido, habría un fichero por cada zona de metadatos del sistema de ficheros. Por tanto, el contenido inicial (obviando, por simplicidad, el subdirectorio `/lost+found`, en caso de que estuviera presente) sería el siguiente:

- Directorio raíz. Como en la versión tradicional ocupa el inodo 1, que hace referencia al bloque B32819. Sin embargo, este bloque de datos, en vez de sólo dos entradas, contiene 7:
  - Las entradas '`.`' y '`..`' apuntando al inodo 1.
  - La entrada `boot` que hace referencia al inodo 2.
  - La entrada `super` que hace referencia al inodo 3.
  - La entrada `mapa_bloques` que hace referencia al inodo 4.
  - La entrada `mapa_inodos` que hace referencia al inodo 5.
  - La entrada `inodos` que hace referencia al inodo 6.
- Fichero `/boot`. Su inodo incluye un puntero directo a B1.
- Fichero `/super`. Su inodo incluye un puntero directo a B2.
- Fichero `/mapa_bloques`. Dado el tamaño de este fichero, se necesita usar el puntero indirecto simple. Su inodo incluye 16 punteros directos a los bloques que van desde B3 hasta B18. Además, utiliza el puntero indirecto simple que hace referencia a un bloque de datos (B32820). Ese bloque de datos sólo tiene ocupadas las 16 primeras posiciones con punteros directos al rango de bloques que se extiende desde B19 hasta B34.
- Fichero `/mapa_inodos`. Su inodo incluye 16 punteros directos a los bloques que van desde B35 hasta B50.
- Fichero `/inodos`. El tamaño de este fichero hace necesario usar el puntero indirecto simple y el doble. Su inodo incluye 16 punteros directos a los bloques que van desde B51 hasta B66. Asimismo, usa un puntero indirecto

simple que hace referencia a un bloque de datos (B32821). Ese bloque de datos tiene ocupadas sus 1024 posiciones (4KB que tiene un bloque/4 bytes que ocupa el puntero) con punteros directos al rango de bloques que se extiende desde B67 hasta B1090. Además, se utiliza el puntero indirecto doble que hace referencia a un bloque de datos (B32822) que contiene 31 punteros indirectos. Cada uno de estos 31 punteros hacen referencia a su propio bloque de datos (desde el B32823 hasta el B32853) que contiene 1024 punteros directos (es decir, todos los punteros que caben en un bloque de datos), excepto el último de ellos que sólo requiere 1008 punteros. De esta forma, el puntero indirecto doble del inodo permite cubrir el rango de bloques que va desde B1091 hasta B32818.

c) Para leer el bloque de *boot* en un sistema tradicional, es necesario abrir el fichero que representa al disco que contiene el sistema de ficheros y leer la cantidad de bytes correspondiente al tamaño de dicho bloque:

```
d=open("/dev/disk1", O_RDONLY);
read(d, buf, TAM_BOOT);
```

Con el nuevo esquema basta con abrir el fichero `/boot` y leerlo completo, no siendo necesario, por tanto, conocer a priori su tamaño:

```
d=open("/boot", O_RDONLY);
while ((leido=read(d, buf, tam))>0)
```

En cuanto a la posibilidad de establecer permisos de acceso específicos para el bloque de *boot*, no es factible con el sistema tradicional, puesto que la unidad de protección en este caso sería todo el dispositivo.

El nuevo esquema planteado, al dar entidad de fichero a esta zona, sí permite fijar unos permisos específicos para la misma. En caso de que se quiera dotar de acceso de lectura de este bloque a un cierto grupo de usuarios, bastaría con establecer que el fichero `/boot` pertenece a dicho grupo (mandato `chgrp`) y dotarle de permiso de lectura para los usuarios de dicho grupo (`chmod`).

d) Con el esquema tradicional no se podría solventar el problema de fiabilidad planteado en el enunciado (al menos en el nivel del sistema de ficheros, puesto que en niveles inferiores, como el manejador de disco o el propio controlador hardware, se pueden usar técnicas que permitan renumerar los bloques físicos del disco de manera que se sustituya el bloque dañado por otro libre en buen estado, pero de forma transparente al sistema de ficheros).

Nótese que esta restricción se debe a que la gestión del sistema de ficheros con el esquema tradicional requiere que todos los bloques de una zona de metadatos estén en bloques contiguos del disco. Por tanto, no se puede sustituir el bloque defectuoso por otro libre que esté en buen estado, puesto que se perdería la contigüidad.

Con la nueva técnica, al dotar de entidad de fichero a cada zona de metadatos, se sigue requiriendo la contigüidad, pero a nivel lógico (es decir, dentro del fichero) en vez de físico (es decir, dentro del dispositivo). Por tanto, una posible solución del problema planteado consistiría en buscar un bloque de disco libre, copiar en el mismo los datos del mapa bits afectados y hacer que el puntero correspondiente del inodo del fichero `/mapa_bloques` apunte a este nuevo bloque en lugar de hacer referencia al bloque físico inservible.

e) A continuación, se detallan las operaciones que conllevaría la apertura del fichero F incluido en el directorio actual usando el esquema tradicional, centrándose en qué bloques del sistema de ficheros se acceden durante la misma:

- En primer lugar, se consulta el inodo del directorio actual, que está en memoria (el inodo del directorio actual se mantiene en `memoria`). Concretamente, se accede a su primer puntero directo, para averiguar en qué bloque de datos está almacenado el contenido del mismo.
- Se lee el bloque de datos del directorio y se busca en el mismo la entrada que contiene el nombre F, `obteniéndose` el número de inodo de F, que será el 260, tal como plantea el enunciado.
- Para leer el inodo, hay que determinar previamente en qué bloque se encuentra almacenado. Dado que hay 16 inodos por cada bloque, el inodo 260 estará almacenado en el bloque decimoséptimo de la zona de inodos. Puesto que esta zona comienza en el bloque B51, habrá que leer el bloque B67 y extraer del mismo el inodo 260, que será el cuarto inodo almacenado en ese bloque.

Las diferencias en cuanto al modo de operación entre la versión tradicional y la nueva aparecen en los accesos a las zonas de metadatos. En este caso, la divergencia surge a la hora de leer el inodo 260 de la zona de inodos:

- Como en la versión tradicional, se detecta que se trata del bloque decimoséptimo de la zona de inodos. Sin embargo, en esta caso hay que acceder al inodo del fichero `/inodos` para determinar la ubicación de dicho bloque. Al ser el bloque decimoséptimo del fichero, corresponde a un bloque asociado al punto indirecto simple, concretamente, al primer puntero almacenado en el bloque indirecto simple. Por tanto, habrá que leer el bloque de datos al que hace referencia el puntero indirecto simple, es decir, el bloque B32821, y consultar el primer puntero almacenado en ese bloque para determinar que el bloque que contiene el inodo es el B67.

Evidentemente, se trata del mismo bloque de inodos que con la versión tradicional, puesto que se ha utilizado la misma distribución inicial. Sin embargo, la diferencia está en cómo se ha determinado la ubicación de dicho bloque en cada versión.

f) A continuación, se detallan las operaciones que implicaría la creación del subdirectorio D incluido en el directorio actual usando el esquema tradicional, centrándose en qué bloques del sistema de ficheros se acceden durante la misma:

- Hay que buscar un inodo libre para el nuevo directorio. Para ello, se debe buscar en el mapa de inodos libres, cuyo primer bloque es el B35. Dado que el enunciado plantea que el inodo 70 es el primero que está libre, bastará con leer ese primer bloque del mapa de inodos libres para encontrarlo y marcarlo como ocupado.
- Se debe buscar un bloque libre para el nuevo directorio. Dado que el enunciado especifica que la caché está inicialmente vacía y que el primer bloque libre se corresponde con uno cuyo estado está almacenado en el último bloque del mapa, habrá que leer todos los bloques del mapa de bits de bloques para encontrar un bloque libre (Bx), es decir, desde el bloque B3 hasta el B34, y marcarlo como ocupado.
- En ese bloque libre Bx se escribirán las entradas '`.`' y '`..`' que harán referencia al inodo 70 y al correspondiente al directorio actual, respectivamente.
- Hay que escribir en el inodo 70 las propiedades del nuevo directorio (dueño, permisos, fechas asociadas, etc.) y la dirección de Bx en el primer puntero directo. La escritura en el inodo conllevará una actualización del bloque donde está incluido, que será el quinto dentro de la zona de inodos, es decir, el bloque B55.
- Se consultará el inodo del directorio actual (presente en memoria) concretamente, su primer puntero directo, para averiguar en qué bloque de datos está almacenado el contenido del mismo. Se leerá dicho bloque de datos buscando un hueco (que el enunciado asegura que existe) para incluir la nueva entrada, que relacionará D con el inodo 70. Habrá que actualizar el inodo del directorio actual incrementando el número de enlaces y las fechas asociadas al directorio. Dependiendo de cómo está implementado el sistema de ficheros, esa actualización puede ser inmediata o diferida. En cualquier caso, la actualización del inodo requeriría calcular en qué bloque está almacenado.

Las diferencias en cuanto al modo de operación entre la versión tradicional y la nueva aparecen en los accesos a las zonas de metadatos, concretamente, al mapa de inodos, al de bloques y a la propia zona de inodos:

- Para buscar un inodo libre, hay que recorrer el fichero `/mapa_inodos`. Como plantea el enunciado, bastará con leer el primer bloque del fichero, al que hace referencia el primer puntero directo (bloque B35).
- La búsqueda del bloque libre requiere recorrer el fichero `/mapa_bloques` hasta que se encuentre uno. Como especifica el enunciado, hay que leer todos los bloques del fichero para localizarlo. Los 16 punteros directos harán referencia a los bloques que se extienden desde el B3 al B18. A continuación, hay que leer el bloque de datos al que hace referencia el puntero indirecto simple, es decir, el bloque B32820, y consultar los 16 punteros almacenado en ese bloque, que harán referencia a los bloques que van desde el B19 hasta el B34.
- La actualización del bloque que contiene el inodo 70 requiere acceder al quinto bloque de inodos, que corresponde al bloque asociado al quinto puntero directo, es decir, al bloque B55. Asimismo, la actualización del inodo actual, sea inmediata o diferida, implicaría actualizar de forma similar el bloque de inodos correspondiente.

Hay que reiterar que también en este caso se trata de los mismos bloques del dispositivo que en la versión tradicional, puesto que se ha utilizado la misma distribución inicial. Sin embargo, la diferencia está en cómo se ha determinado la ubicación de los mismos en cada versión.

g) En la versión tradicional, si se agotan los inodos, no se puede crear ningún nuevo fichero o directorio en el sistema de ficheros, puesto que la zona de inodos no se puede expandir ya que se perdería la contigüidad requerida por el sistema de ficheros. Por tanto, la operación daría error, incluso aunque hubiera bloques de datos libres. Por ese motivo, es un punto crítico especificar a la hora de crear el sistema de ficheros un número razonable de inodos conforme al uso previsto del mismo.

Con el nuevo esquema, al eliminar el requisito de contigüidad en el espacio físico del dispositivo, bastando con que haya una contigüidad lógica en el espacio del fichero que representa la zona de metadatos correspondiente, se puede expandir la zona de inodos aprovechando la existencia de bloques libres en el disco. Además de expandir la zona de inodos, habrá que extender el mapa de bits de inodos de forma proporcional. Así, si se añade un nuevo bloque al fichero de inodos, se deberá incrementar en 2 bytes el fichero del mapa de bits de inodos (un bloque de inodos contiene 16 inodos y, por tanto, requiere 16 bits en el mapa). Ambas expansiones se realizan de la misma forma que con cualquier fichero convencional.

Con respecto a la expansión del fichero de inodos, suponiendo que aumenta su tamaño en un bloque (podría plantearse una expansión mayor), conllevaría las siguientes operaciones:

- Buscar un bloque libre consultando el fichero `/mapa_bloques`, marcándolo como ocupado.
- Actualizar el inodo del fichero `/inodos` aumentando su tamaño en un bloque y conectando el nuevo bloque mediante el puntero correspondiente. Esa conexión puede requerir la lectura y/o la reserva de bloques indirectos. Si se plantea una expansión sobre el ejemplo de zona de inodos planteado en el enunciado, habría que leer el bloque indirecto doble (B32822), así como el bloque apuntado por el último puntero válido de ese bloque indirecto (el puntero trigésimo primero que hace referencia al bloque B32853), añadiendo una referencia al nuevo bloque reservado en la posición 1009 de este último bloque indirecto de segundo nivel.

En cuanto a la expansión del mapa de bits, suponiendo que se aumenta en 2 bytes, requeriría las siguientes operaciones:

- Si el aumento requiere reservar un nuevo bloque de datos, como ocurre si se realiza una expansión sobre el ejemplo planteado en el enunciado, hay que hacer lo mismo que con la expansión del fichero de inodos: buscar un bloque libre, actualizar el inodo aumentando el tamaño (en este caso, sólo en 2 bytes) y conectando el nuevo bloque. Si se trata de una primera expansión sobre el ejemplo planteado, dado que el tamaño inicial del fichero del mapa de bits de inodos usa todos los punteros directos, será necesario reservar un bloque adicional para usarlo como indirecto y hacer que la primera posición de ese bloque indirecto apunte al nuevo bloque reservado para el fichero. Además, habría que poner a cero los dos nuevos bytes del fichero, para indicar que los nuevos 16 inodos están libres.
- En caso de que el aumento no requiera reservar un nuevo bloque de datos, como ocurriría si se realiza una segunda expansión sobre el ejemplo planteado, sólo sería necesario actualizar el tamaño del inodo y poner a cero los 2 nuevos bytes del mismo.

h) A lo largo de los distintos apartados del ejercicio se han podido observar algunas de las ventajas de este esquema, que se recapitulan a continuación:

- Mejoras en cuanto a la protección de la información. Con este esquema, al otorgar la categoría de fichero a cada zona de metadatos, se pueden establecer permisos específicos para cada una de ellas.
- Mejoras en aspectos de fiabilidad. Al tratar cada zona de metadatos como un fichero, el requisito de contigüidad física de la zona se transforma en el de contigüidad lógica, lo que permite sustituir bloques de metadatos dañados sin afectar al modo de operación del sistema de ficheros.
- Eliminación de algunas restricciones del sistema de ficheros en cuanto a su capacidad de crecimiento. Nuevamente, al reemplazar el requisito de contigüidad física de la zona por el de contigüidad lógica, se puede adaptar el tamaño de cada zona de metadatos a las necesidades que haya en cada momento. De esta forma, se eliminan las situaciones anómalas presentes en el esquema tradicional, en las que existiendo espacio libre en el dispositivo, no puede seguir creciendo el sistema de ficheros por falta de espacio en una zona de metadatos (concretamente, en la zona de inodos).

También se han podido detectar algunas desventajas, vinculadas con una mayor sobrecarga en espacio y tiempo:

- Existe un mayor consumo de espacio para los metadatos, debido a los bloques indirectos requeridos por algunas zonas de metadatos y los inodos requeridos por los ficheros de metadatos. En el ejemplo planteado, se han necesitado 34 bloques indirectos: uno para el fichero del mapa de bits y 33 para el fichero de inodos (1 indirecto simple y 1 indirecto doble que hace referencia a 31 indirectos simples). Asimismo, se han requerido 5 inodos, que ocupan menos de la mitad de un bloque. Hay que destacar que se trata de una sobrecarga perfectamente tolerable, del orden de un 0,003% del espacio del disco (aproximadamente, 30 bloques de 1.000.000).
- Los bloques indirectos, además de causar un peor aprovechamiento del espacio útil, conllevan una sobrecarga en la operación del sistema de ficheros, puesto que necesitar ser consultados a la hora de acceder a los metadatos asociados a los mismos. Así, por ejemplo, la lectura de un inodo puede requerir dos accesos a disco más que en un sistema tradicional, debido a los bloques indirectos involucrados. Esta sobrecarga es mucho más importante que la previa y requiere de técnicas de optimización específicas para tratar de paliarla. Por otro lado, la consulta de los inodos para resolver los accesos a los metadatos también impone una cierta sobrecarga, aunque manteniéndolos en memoria (es decir, manteniendo los ficheros abiertos) es perfectamente tolerable.

Para terminar, hay que resaltar que en el sistema de ficheros NTFS de Windows se sigue esta estrategia. Es conveniente destacar que en este caso no se producen las desventajas identificadas previamente, puesto que en NTFS un fichero que esté contiguo en el dispositivo sólo requiere una entrada en el descriptor del mismo (recuerde que en NTFS hay una entrada en el descriptor de fichero por cada región contigua o fragmento, *extent*, que tenga éste), eliminando la sobrecarga en espacio y tiempo que causan los bloques indirectos. Así, hipotéticamente, para una zona de metadatos como la de inodos planteada en el ejemplo, sólo se requeriría una entrada que relacionara los bloques lógicos del 0 al 32K con los bloques físicos desde el B51 hasta el B32818: [B. lógico 0 → B. físico 51; tamaño del fragmento 32768].