Enunciado

En los sistemas de ficheros modernos se está planteando la inclusión de lo que se denominan *snapshots*, que son copias del estado del sistema de ficheros en un momento dado que se mantienen con los datos que había en dicho momento. Estas copias no se ven alteradas mientras que el sistema de ficheros se sigue utilizando (borrando, creando y modificando su contenido), permitiendo, si las circunstancias así lo requieren, recuperar el estado del sistema en dicho punto. Antes de nada vamos a analizar ciertas operaciones:

a). ¿Cuál es la información (datos y metadatos) que se modificaría en una operación de borrado de un fichero en un sistema de ficheros? ¿Y si se escriben más datos en un fichero ya existente?

b). Mover un fichero de un directorio a otro, ¿qué información del sistema de ficheros modifica?

Para la gestión de la funcionalidad de *snapshot* vamos a considerar las siguientes restricciones:

- La operación de creación de un *snapshot* se debe realizar con el sistema de ficheros desmontado.
- En el proceso de montaje del sistema de ficheros se podrá indicar si se desea montar el estado actual o un *snapshot* anterior.
- Los *snapshots* tienen que ser sistemas de ficheros completamente funcionales, aunque son de sólo lectura.
- Existe una operación (a realizar con el sistema de ficheros desmontado) que consiste en recuperar el estado del sistema de ficheros desde un *snapshot*. Esta operación descarta todas las modificaciones realizadas desde que se fijó dicho *snapshot* en adelante, haciendo que el sistema de ficheros coincida con el que había en ese momento, y que a partir de entonces se considere el estado actual.
- Para referirnos a un *snapshot* usamos un identificador (SnapID), que serán números correlativos en orden (SnapID mayor, implica copia más reciente).

Descartando la solución evidente de copiar completamente los contenidos en un sistema de *backup*, se pretende diseñar una solución que aproveche de forma eficiente el espacio de almacenamiento (replicando sólo los elementos modificados entre *snapshots*). Responda a las siguientes cuestiones:

c). ¿En cuáles de las estructuras del sistema de ficheros (datos y metadatos) habría que incluir el SnapID?

d). De las estructuras identificadas anteriormente, ¿cómo estarían relacionadas las estructuras análogas (de *snapshots* diferentes) entre sí? ¿tendrían que estar enlazadas o referenciadas entre sí de alguna manera?

e). Si se ha creado un *snapshot* anterior y, sobre el estado actual del sistema de ficheros, se quiere borrar un fichero. Indique los pasos a realizar para hacerlo, centrándose exclusivamente en las operaciones específicas para mantener las funcionalidades de los *snapshots*.

f). Repita la operación para la escritura de nuevos datos en un fichero ya existente.

g). ¿Cómo se podría montar en modo sólo lectura un *snapshot* antiguo? Indique los pasos a realizar para abrir un fichero y leer su contenido en esta situación.

h). Supongamos que existe una operación de purga, consistente en eliminar un *snapshot* anterior determinado. Indique en qué consistiría esta operación si eliminamos un *snapshot* intermedio de los guardados en el sistema.

i). ¿Plantea algún problema que los *snapshots* sean de escritura? ¿Cuál o cuáles?

Solución

a). ¿Cuál es la información (datos y metadatos) que se modificaría en una operación de borrado de un fichero en un sistema de ficheros? ¿Y si se escriben más datos en un fichero ya existente?

Borrar un fichero actualiza diferentes estructuras del sistema de ficheros almacenado en el soporte físico:

- Se marcan en el bitmap de bloques como libres todos los bloques ocupados por el fichero (es decir todos los bloques referenciados por sus punteros directos e indirectos, incluyendo aquellos usados en la indirección).
- Se marca en el bitmap de i-nodos como libre el i-nodo del fichero.
- Se accede el bloque de datos del directorio que contenía al fichero borrado. Los bloques de datos referenciados desde el i-nodo del directorio se corresponden con los contenidos (entradas) del directorio. En ese contenido se borrará la entrada con el nombre del fichero y que, a su vez, referenciaba al i-nodo ya liberado, modificamos uno (o varios) bloques de datos.
- Se actualiza el i-nodo del directorio que contenía al fichero para actualizar fecha y hora de modificación.

A estas operaciones hay que añadir determinadas verificaciones, como son que el contador de enlaces del fichero pase de 1 a 0 (el fichero es borrable) que, en caso de no ser así, simplemente se decrementaría el contador y nada más. Otra verificación serían los permisos de acceso a ruta y fichero y por último se comprobaría si el fichero a borrar está actualmente en uso. De ser así, la entrada del directorio se borra pero los bitmaps no se modifican para no liberar los recursos hasta que el fichero se deje de usar.

La escritura de más datos en un fichero implica que:

- Si el incremento de espacio es menor que el espacio libre que queda en el último bloque del fichero, no hace falta pedir más bloques de datos.
- Si el tamaño extra no entra en el espacio libre del último bloque se solicitan uno o varios bloques de datos extra (reservándolos en el bitmap de bloques).
- Si la provisión de nuevos bloques requiere también reservar un nuevo bloque indirecto, éste se marcará como ocupado en el bitmap de bloques.
- Se actualizará la fecha de modificación del i-nodo del fichero.

b). Mover un fichero de un directorio a otro, ¿qué información del sistema de ficheros modifica?

El mover un fichero de un directorio a otro (dentro del mismo sistema de ficheros) no implica ninguna operación de copia y borrado de los datos. En realidad al fichero, propiamente dicho, se haría referencia desde un nuevo directorio:

- Se accede al bloque de datos del directorio original y se elimina una entrada en el contenido del directorio.
- En el directorio destino se crea una nueva entrada (modificando uno o varios bloques de datos del contenido del direcotrio). La nueva entrada apuntaría al i-nodo del fichero movido (que no se modificaría).
- Actualizamos fechas de modificación de los i-nodos de ambos directorios.

Sólo si la opción de mover un fichero se hace entre sistemas de ficheros diferentes sería el caso en el cual sí habría que copiar los contenidos. Sería análogo a crear un nuevo fichero y borrar el antiguo. Para este enunciado, y tratándose del estudio de un sistema de ficheros podemos no considerar este caso.

Consideraciones generales del diseño

Podemos considerar que identificamos cada versión del sistema de ficheros con un valor diferente de SnapID. Cuando indicamos al sistema de ficheros que con el contenido actual del sistema de ficheros queremos crear un *snapshot*, incrementaremos dicho indicador. Así pues, el SnapID será un número (correlativo) que iremos asignando a las entidades del sistema de ficheros en el momento en el que queramos indicar que dicha entidad existía en esa versión.

Para ver cómo gestionar los *snapshot* y sus correspondientes SnapID debemos tener en cuenta ciertas consideraciones:

- Las entidades sujetas a control de versiones son los ficheros y directorios, compuestos cada uno de ellos por un i-nodo y varios bloques de datos. Las otras estructuras de metainformación del sistema de ficheros (bitmaps o superbloque, por ejemplo) se verán modificadas sólo si es necesario para gestionar los elementos anteriores. Es decir que lo importante es que tengamos versiones de ficheros y directorios, no del superbloque (salvo que se necesite para tener las versiones de ficheros/directorios)
- Los i-nodos y los bloques de datos están enlazados entre sí:
  - Un i-nodo hace referencia a los números de bloque de datos (contenido de un fichero o directorio).
  - Los bloques de datos de un directorio registra las entradas de directorio que éste contiene (compuestas por nombre y número de i-nodo).
- Si un fichero cambia de contenido entre dos *snapshots* consecutivos podemos almacenar este cambio de dos posibles formas:
  a). [Versiones de bloques de datos] Únicamente la parte de contenidos modificada, lo cual sería parte de los bloques de datos, cada bloque de datos sería válido para una versión, la otra o para las dos (si su contenido no se ha modificado).
  b). [Versiones de i-nodos] La otra alternativa es tener una copia completa del contenido del fichero para cada versión, lo cual se podría realizar a nivel del i-nodo, tenido un i-nodo para cada versión. Cada i-nodo haría referencia a una serie de bloques de datos diferente.
- Independientemente de la alternativa que tomemos, toda entidad sujeta a cambio (debido a creación, borrado o modificación) debe considerarse que su estado y contenido será válido en un rango (continuo) de SnapIDs (el fichero existe con ese contenido de el *snapshot* X al *snapshot* Y).

Si analizamos las dos alternativas a) Versiones de bloques de datos frente a b) Versiones de i-nodos, vemos que es necesario asignar al que corresponda un rango de *snapshots* en el cual es válido (SnapID inicial a SnapID final), así pues tenemos los siguientes posibles diseños:

A. Versión de bloques de datos haciendo que cada bloque de datos contenga la pareja SnapID inicial y SnapID final en el cual es válido: Esta solución plantea serios problemas:
   - Consumimos parte del espacio del bloque de datos con estos dos contadores, lo cual implica que los bloques son de un tamaño menor, que al cargarlos en memoria hay que desplazarlos para que coincidan con páginas de memoria, dificultando mucho su gestión.
   - Cuando se haga referencia a un bloque de datos (desde un i-nodo), pongamos el primer bloque del fichero, este bloque será el válido para las versiones X a Y (según indique los dos valores de SnapID), ¿dónde guardamos y cómo referenciamos el primer bloque de versiones anteriores o posteriores?

   Diseño descartado.

B. Versión de bloques de datos haciendo que el i-nodo o el bloque indirecto que lo referencia contenga la pareja SnapID inicial y SnapID final. Con esta solución solventamos la primera de las pegas del diseño anterior, pero:
   - Seguimos teniendo el problema de hacer referencia a los bloque de varias versiones. Si lo que hacemos es substituir un puntero a un número de bloque por una tripleta que indique los dos SnapIDs y el número de bloque nos cargamos la indexación de bloques de datos. Los punteros directos, indirectos, etc. se referencian de forma directa, es decir, que si quiero acceder al byte X sé el bloque en el que se encuentra y lo indexo directamente (puntero X dividido el tamaño de bloque). Si ahora la lista de punteros a bloques se desglosa con el número de versión, entonces o multiplico su tamaño por el número de versiones (teniendo todos los punteros por todas las versiones) o la tengo que recorrer secuencialmente. El lastre para la indexación de posiciones dentro del fichero es enorme.

   Diseño descartado.

C. Versión de i-nodos con número de SnapID inicial y final en el propio i-nodo. Con este diseño modificamos el formato del i-nodo añadiendo estos dos campos. De forma que tendremos dos i-nodos diferentes para un fichero que se ha modificado, uno válido de las versiones X a Y y otro válido de la versión Z en adelante. Cada i-nodo apuntará a los bloques de datos correspondientes a cada versión. Existen también ciertas pegas a este diseño:

- Si una modificación de contenido sólo afecta a unos pocos bytes del fichero en relación a un *snapshot* anterior, entonces tendríamos duplicados muchos bloques de datos con la misma información. Esta pega puede ser asumible dependiendo del tamaño medio de los ficheros, de la naturaleza de los cambios en los mismos y la frecuencia con la que se hagan *snapshots*.
- Ahora que tenemos varios i-nodos con el mismo fichero (en versiones diferentes) debemos ver cómo los referenciamos. Eso se hace desde el directorio, es decir que el directorio, en su contenido, tendrá un solo puntero a un i-nodo para cada fichero. Ese podría ser el de la primera o de la última versión, pero ¿cómo recuperamos los de otras versiones? Una opción sería que los i-nodos de ficheros de versiones anteriores estén enlazados (una lista enlazada, por ejemplo), de forma que el i-nodo deberá tener un nuevo campo que referencie a la versión anterior.

Diseño aplicable

D. Versión de i-nodos con número de SnapID inicial y final en el directorio que lo contiene. Al igual que con el caso de bloques, podemos propagar los valores de los dos SnapID al nivel anterior, ene l caso del i-nodo, el nivel anterior es el de contenido de directorio. Deberíamos definir un nuevo formato de contenido de directorio que contenga no sólo el nombre y el i-nodo sino que además contenga los SnapID inicial y final que para un nombre dado se traducen en un número de i-nodo determinado. Las diferencias con la alternativa anterior son:

- No tocamos la estructura i-nodo que sería igual que en el caso estándar. El diseño se simplifica.
- Sí habrá que cambiar cómo se accede a los contenidos de un directorio, la llamada *readdir*, y el proceso de *look-up*, especialmente. En este caso, comparado con el caso de los punteros a bloques de la opción B no tenemos merma en el rendimiento porque el contenido de un direcotrio no se accede de forma indexada, se recorre secuencialmente cuando se busca una entrada. En este caso sólo habrá que considerar las entradas cuyo rango de validez de SnapID inicial-final nos interese.

Diseño más apropiado

Hay que tener en cuenta que esta solución sería asimilable a guardar en el directorio una copia de un fichero cuando se va a modificar y nos interese tener la versión anterior. Eso lo hacen sistemas de ficheros del tipo VMS, aunque de forma trasparente y gestionado bajo otras condiciones. A lo largo de los siguientes apartados vamos a considerar que usamos la opción de diseño D (anquen la C sería también aplicable en la mayoría de casos).

c). ¿En cuáles de las estructuras del sistema de ficheros (datos y metadatos) habría que incluir el SnapID?

Si tomamos la opción de diseño D, sólo habrá que incluir esa información en los bloques de datos de los contenidos de directorios (como formato de la tabla de entradas del directorio).

Es muy importante recalcar que no haría falta “versionar” ni incluir el SnapID en los bitmaps ya que son estructuras que indican recursos libres de cara a su asignación rápida. Un i-nodo o un bloque de datos estará ocupado si lo está usando un fichero o los datos de un fichero de una versión actual o posterior (no importa el *snapshot* que lo usaba, sólo que está usado y no se puede asignar a uno nuevo). Ya vernos más adelante cómo liberarlos.

En el superbloque sí sería necesario indicar cuál es el SnapID actual (última versión) y cuáles son SnapID válidos (no purgados, ver más adelante).

d). De las estructuras identificadas anteriormente, ¿cómo estarían relacionadas las estructuras análogas (de *snapshots* diferentes) entre sí? ¿tendrían que estar enlazadas o referenciadas entre sí de alguna manera?

Considerando de nuevo el diseño D, dos versiones de un mismo fichero de snapshots diferentes compartirán el mismo directorio, pero su rango de validez será diferente. Algo del tipo:

Contenido de directorio

Nombre entrada	**SnapID$_{ini}$**	**SnapID$_{fin}$**	**i-nodo**
…….			
Fichero.txt	0	-1	4656
Modificado.doc	0	7	6333
Modificado.doc	8	-1	16221
Nuevo.jpg	6	-1	9331
…….			

El fichero *Fichero.txt* existe desde la primera versión hasta la versión actual (que podemos designar con el SnapID=-1), el fichero *Modificado.doc* se modificó después del *snapshot* de la versión 8 (hay dos copias). Y El fichero *Nuevo.jpg* se creó después de hacer el *snapshot* de la versión 5.

e). Si se ha creado un *snapshot* anterior y, sobre el estado actual del sistema de ficheros, se quiere borrar un fichero. Indique los pasos a realizar para hacerlo, centrándose exclusivamente en las operaciones específicas para mantener las funcionalidades de los *snapshots*

Si borramos un fichero lo que debemos hacer es decir que el fichero era válido hasta el snapshot anterior. Supongamos que estamos en el SnapID=3 y tenemos este contenido de directorio:

Contenido de directorio

Nombre entrada	**SnapID$_{ini}$**	**SnapID$_{fin}$**	**i-nodo**
…….			
Datos.txt	0	-1	233
Temporal.raw	3	-1	442
…….			

Si borramos el fichero *Datos.txt* debemos dejar una copia válida hasta la versión 2. Ponemos el SnapID final a SnapID actual-1.

Contenido de directorio

Nombre entrada	**SnapID$_{ini}$**	**SnapID$_{fin}$**	**i-nodo**
…….			
Datos.txt	0	2	233
Temporal.raw	3	-1	442
…….			

En esta operación es muy importante constatar que ni el i-nodo 233 correspondiente a dicho fichero no se liberaría, tampoco sus bloques de datos. Si los borramos ya no sería posible recuperar ese fichero ya que nuevos ficheros podrían usar ese i-nodo o esos datos y machacar el contenido. A efectos del uso del disco esos recursos aún están ocupados.

Si borramos el fichero *Temporal.raw* el caso es diferente. El fichero no existe en *snapshot* anteriores, se ha creado después de que se haya fijado el *snapshot* 2. Eso implica que si se quiere borrar no hay que guardar copia anterior. Dicho fichero se borraría completamente liberando todos los recursos (i-nodo y bloques de datos que se marcaría también como libres en los bitmaps correspondientes). La condición de borrado de un fichero es que entre su SnapID inicial y su SnapID final existan *snapshot* válidos no purgados (ver más adelante). En el caso del fichero Temporal.raw sería SnapID inicial 3, SnapID final 2 (actual -1). No hay snapshots válidos en ese rango, el fichero se borra.

Contenido de directorio

Nombre entrada	**SnapID$_{ini}$**	**SnapID$_{fin}$**	**i-nodo**
…….			
Datos.txt	0	2	233
…….			

f). Repita la operación para la escritura de nuevos datos en un fichero ya existente.

Si lo que hacemos es escribir más datos en un fichero ya existente entonces tenemos que crear una nueva versión. Esa nueva versión será un nuevo i-nodo y el contenido de partida igual a los datos del fichero original que se irán modificando.

Por ejemplo si tenemos este directorio en el SanpID=3:

Contenido de directorio

Nombre entrada	$SnapID_{ini}$	$SnapID_{fin}$	i-nodo
…….			
Text1.doc	0	-1	2553
Log.txt	3	-1	5123
…….			

Si modificamos el fichero *Text1.doc* debemos crear otra entrada con otro i-nodo y dejar una versión anterior del viejo.

Contenido de directorio

Nombre entrada	$SnapID_{ini}$	$SnapID_{fin}$	i-nodo
…….			
Text1.doc	0	2	2553
Text1.doc	3	-1	6311
Log.txt	3	-1	5123
…….			

Si, por el contrario, modificamos *Log.txt*, como no hay que guardar copia anterior las modificaciones se harían sobre el fichero existente. Las reglas para determinar si se debe guardar versión anterior serían análogas a la de borrado vistas antes.

g). ¿Cómo se podría montar en modo sólo lectura un *snapshot* antiguo? Indique los pasos a realizar para abrir un fichero y leer su contenido en esta situación.

Montar un snapshot anterior es sencillo, puesto que eso se aplica sólo a las operaciones de recorrido y búsqueda sobre el directorio. Por ejemplo si el SnapID actual es 8 y queremos montar el SnapID 5, para cada una de las operaciones sobre el directorio:

Contenido de directorio

Nombre entrada	$SnapID_{ini}$	$SnapID_{fin}$	i-nodo
…….			
Datos.txt	0	2	2574
Datos.txt	3	6	4632
Datos.txt	7	-1	11234
Memoria.doc	6	-1	8711
Imagen.img	4	4	9723
Imagen.img	5	7	9932
Temporal.tmp	8	-1	12453
…….			

A la hora de recorrer o buscar en el directorio sólo consideraríamos las que existían en el SnapID 5, es decir la copia de *Datos.txt* del i-nodo 4632 y la de *Imagen.img* del i-nodo 9932. Hay que tener en cuenta que fichero, como éste último, no existe en la versión actual del sistema de ficheros (se borró después de hacer el *snapshot* de la versión 7). Otros ficheros como *Temporal.tmp* no existían hasta la versión 8.

Para poder realizar este proceso se debería modificar la función de recorrido de directorios y se debe mantener en el superbloque cargado en memoria (superbloque virtual) el SnapID de la versión montada, ya que ésta no es la actual.

h). Supongamos que existe una operación de purga, consistente en eliminar un snapshot anterior determinado. Indique en qué consistiría esta operación si eliminamos un snapshot intermedio de los guardados en el sistema.

La purga de un snapshot intermedio es la única operación que nos puede eliminar datos ya consolidados en versiones anteriores. Como no queremos alterar la numeración (eso implicaría modificar todos los SnapID de todos los directorios del sistema y un potencia problema de coherencia de nombrado), lo más fácil es mantener una lista de SnapID intermedios purgados.

Esta lista se podría guardar en el superbloque (como lista o como mapa de bits, según sea su tamaño y densidad).

Cuando purgamos un *snapshot*, incluimos su SnapID en la lista y recorremos el árbol de directorio desde la raíz y volvemos a aplicar la política de borrado (que el fichero exista en, al menos, un *snapshot* válido entre SnapID inicila y SnapID final).

De esta forma, por ejemplo, si un fichero sólo existe entre el SnapID=7 y el SnapID=9 y las versiones 7, 8 y 9 se han purgado, dichero fichero se puede borrar permanentemente.

i). ¿Plantea algún problema que los *snapshots* sean de escritura? ¿Cuál o cuáles?

El principal problema es que si estamos en el SnapID=7 (versión actual) y montamos un *snapshot* anterior, pongamos que el SnapID=4, y modificamos los contenidos del sistema de ficheros (creando, borrando y cambiando ficheros) esas modifcaciones se propagarían en versiones posteriores ya guardadas (por ejemplo, en el SnapID=6) pudiendo existir conflictos con contenidos posteriores (por ejemplo un fichero que se modificó en la versión 5).

Si además hacemos nuevos *snapshots* sobre esta versión recuperada y modificada la numeración ya no representará copias sucesivas y gran parte de la lógica de control de borrados y recorridos de directorios no valdría. Eso implicaría que la secuencia de SnapIDs no es una secuencia lineal sino que hay versiones sobre las cuales hay bifurcaciones (de la versión 4 pasamos a la 5 en un momento, pero al recuperarla y hacer otras modificaciones). Eso implicaría una estructura y lógica de control más complicada ((Como diría Doc: "*Si vas a hacer una máquina del tiempo hazla con estilo*")).