Ejercicio de ficheros del examen de Diseño de SS.OO. de junio de 2008

La caché del sistema de ficheros de UNIX ha ido sufriendo diversas modificaciones según ha ido evolucionando este sistema operativo. Vamos a analizar una de estas variaciones: el paso de una caché de dispositivos a una de ficheros.
En el diseño original de UNIX se usaba una caché de bloques de dispositivos, a la que se accedía internamente mediante el identificador del dispositivo de bloques y el número de bloque dentro del mismo (algo como: get_block(id_disp, num_bloque)). En sistemas UNIX más modernos, se utiliza una caché de bloques de ficheros, a la que se accede usando el identificador del fichero y el número de bloque dentro del mismo (algo como: get_block(id_fich, num_bloque)). Analice las ventajas y desventajas del nuevo diseño frente al original, respondiendo a las siguientes preguntas:

a) Enumere qué capas de software presenta un sistema UNIX desde las llamadas al sistema de ficheros hasta los dispositivos. Sitúe los dos diseños de caché planteados en esa jerarquía.

b) Suponga un sistema que usa bloques de 4KB y cuya caché, sea del tipo que sea, está inicialmente vacía. En este sistema se abre un fichero, que está almacenado de forma contigua a partir del bloque 1000 del disco, y se leen sus primeros 100 bytes. A continuación, se invoca la llamada read(d, buf, 5000).
b1) Explique cómo se lleva a cabo esta última llamada en un sistema con el diseño de caché original, identificando qué pasos conlleva, qué estructuras de datos en memoria del sistema operativo se consultan (BCP, tabla intermedia, tabla de inodos, ...), y qué valores se extraen de las mismas. Asimismo, muestre qué accesos a la caché se generan (identificando cómo sería cada llamada get_block) y en qué paso se producen.
b2) Muestre qué diferencias existen si se utiliza el nuevo diseño de caché, especificando nuevamente qué accesos a la caché se generan (identificando cómo sería cada llamada get_block) y en qué paso se producen. Analice razonadamente qué diseño es más eficiente y en qué circunstancias.

c) Se pretende implementar un sistema de ficheros distribuido como NFS, donde los ficheros pueden estar almacenados en otra máquina pero, por eficiencia, se guardan en la máquina cliente copias en caché de los bloques accedidos de dichos ficheros. Explique razonadamente qué diseño de caché considera más adecuado para este sistema.

d) En un sistema UNIX se generan muchos ficheros temporales, cuya vida es significativamente más corta que el periodo de volcado de la caché, por lo que ni siquiera tienen que escribirse sus datos al dispositivo. Con objeto de minimizar todavía más la sobrecarga introducida por el tratamiento de este tipo de ficheros, se puede intentar diferir lo más posible el momento en el que se le asignan bloques libres del dispositivo a estos ficheros, para poder incluso evitarlo. Estudie esta cuestión para cada tipo de diseño de caché.

e) Los sistemas UNIX permiten acceder directamente a un dispositivo de bloques ofreciendo un fichero especial para cada dispositivo (por ejemplo, /dev/hda1). Los accesos al dispositivo usando ese fichero especial pasan también a través de la caché (en el caso del nuevo diseño, se usa como identificador de fichero el correspondiente al fichero especial). El sistema operativo permite acceder directamente al dispositivo usando el fichero especial mientras está montado. ¿En cuál de los diseños de caché esta doble posibilidad de acceso puede crear problemas de coherencia en la caché (concretamente, un problema de doble *caching*: mismo bloque almacenado dos veces)?

f) Se pretenden analizar aspectos adicionales de la coherencia de la caché vinculados con el borrado de información.
f1) Para ello, previamente y obviando inicialmente la existencia de la caché, indique qué operaciones sobre las estructuras del sistema de ficheros se producen cuando se ejecuta sobre un fichero la llamada ftruncate(0) (recuerde que ftruncate fija la longitud del fichero pudiéndolo truncar o expandir dependiendo de su tamaño actual). A continuación, se invoca ftruncate(8192) sobre el mismo fichero. ¿Qué operaciones se producen en esta segunda llamada?
f2) Centrándonos ya en los aspectos de coherencia de la caché, responda razonadamente si se pueden producir problemas de coherencia en alguno de los dos diseños de gestión de caché si no se invalidan en la caché los bloques afectados por la primera llamada a ftruncate.

Solución

a) El procesamiento de una llamada al sistema de ficheros en UNIX requiere una serie de etapas dentro del sistema operativo que culminan con el acceso a los sectores del dispositivo involucrado. Este procesamiento está organizado básicamente en dos niveles: el sistema de ficheros propiamente dicho, que se encarga de crear la abstracción de fichero, y la capa de manejadores de los distintos dispositivos de bloques presentes en el sistema, cada uno de los cuales esconde las características específicas del dispositivo ofreciendo una interfaz común que permite ver el dispositivo como un vector de bloques.
Cada una de estos dos niveles puede subdividirse a su vez en otros dos. En cuanto al sistema de ficheros, la coexistencia de múltiples tipos de sistemas de ficheros, aspecto característico de los sistemas UNIX, propicia la separación entre una capa superior, el sistema de ficheros virtual (VFS, *Virtual File System*), que agrupa toda la funcionalidad común del servicio de ficheros que es independiente del tipo de sistema de ficheros subyacente, y una capa inferior, que abarca todos los sistemas de ficheros específicos.
Por lo que se refiere al nivel de dispositivos de bloques, dado que hay una parte considerable de la funcionalidad que es común para todos los manejadores, tal como la gestión de la cola de peticiones y la planificación de los accesos al dispositivo, parece razonable crear una capa superior que aúne toda esta funcionalidad compartida por la mayoría de los manejadores de los dispositivos de bloques, a la que podemos denominar capa general de entrada/salida de bloques, y que se sitúe encima de los diversos manejadores específicos de los dispositivos de bloques.
Recapitulando, se pueden distinguir cuatro niveles:

1. Sistema de ficheros virtual, que recibe peticiones de acceso a un rango de bytes del fichero y determina qué bloques del fichero están involucrados, teniendo en cuenta para ello el tamaño de bloque usado por el sistema de ficheros. Nótese que, dado que este nivel es independiente del tipo de sistema de ficheros subyacente, no puede conocer en qué bloques del dispositivo se ubican los bloques del fichero. Este nivel usa *vnodos* como descriptores genéricos de los ficheros.
2. Sistema de ficheros específico, que recibe peticiones de acceso a bloques del fichero y las transforma en solicitudes de bloques del dispositivo usando para ello el *inodo*, en el caso de sistemas de ficheros basados en la filosofía UNIX, o una estructura equivalente.
3. Capa general de entrada/salida de bloques, que no realiza ninguna transformación en cuanto al nivel de abstracción de las peticiones, limitándose a agruparlas y ordenarlas para optimizar el acceso al dispositivo.
4. Manejador específico del dispositivo de bloques, que transforma las peticiones de acceso a bloques del dispositivo en el tipo de unidades que maneja el mismo, que son, generalmente, sectores.

En cuanto a la ubicación de los dos diseños de caché en la jerarquía de niveles de software que se acaba de plantear, ésta será la siguiente:

- El diseño original usa una caché de dispositivos, a la que se accede mediante un identificador de dispositivo y un número de bloque, por lo que ésta debe ubicarse entre el nivel de sistemas de ficheros específicos y la capa general de entrada/salida de bloques.
- El nuevo diseño utiliza una caché de ficheros, a la que se accede mediante un identificador de fichero y un número de bloque, por lo que debe situarse entre el sistema de ficheros virtual y el nivel de sistemas de ficheros específicos.

b1) A continuación, se detallan los pasos que se llevan a cabo para resolver la llamada `read(d, buf, 5000)` en un sistema que utiliza el diseño de caché original:

1. El sistema operativo consulta el BCP del proceso actual y accede a la posición *d* de la tabla de ficheros abiertos incluida en el BCP. En esa posición se almacena una referencia a una entrada de la tabla intermedia.
2. En la entrada de la tabla intermedia se obtiene la posición actual dentro del fichero, que será 100 puesto que ha habido una llamada previa que leyó 100 bytes. Por tanto, la llamada solicita un acceso en el rango de bytes [byte 100, byte 5100]. En la entrada también se almacena una referencia al *vnodo* que corresponde al fichero involucrado.
3. En el *vnodo* está almacenado el tamaño del bloque (4096 bytes en el ejemplo planteado), que permite calcular qué bloques del fichero están involucrados en la lectura: [byte 100, byte 5100] $\rightarrow$ [bloque 100/4096, bloque 5100/4096] $\rightarrow$ [bloque 0, bloque 1]. Asociado al *vnodo* está el *inodo* (o estructura equivalente) del sistema de ficheros específico. En este punto termina la labor del nivel de sistema de ficheros virtual, invocando la función de lectura específica del nivel de sistema de ficheros subyacente, solicitando los dos bloques del fichero requeridos ([bloque 0, bloque 1]).
4. El sistema de ficheros específico traduce la petición usando el *inodo* del fichero. Al tratarse de los dos primeros bloques del fichero, necesita consultar los dos primeros punteros directos del *inodo*. Dado que el

fichero está almacenado de forma contigua en el dispositivo a partir del bloque 1000, el resultado de la transformación es: [bloque 0 del fichero, bloque 1 del fichero] → [bloque 1000 del dispositivo, bloque 1001 del dispositivo].

5. En este punto se realizan dos consultas a la caché: get_block(*D*, 1000), que tiene acierto puesto que ese bloque se ha accedido en la lectura previa, y get_block(*D*, 1001), que falla y causa una petición de ese bloque a la capa general de entrada/salida de bloques, siendo *D* el identificador del dispositivo (puede corresponder a la unión del *major* y *minor* en UNIX), que está almacenado en el *vnodo*.

b2) Los tres primeros pasos son iguales, pero con este nuevo diseño de caché, justo al final del tercer paso, antes de invocar la rutina específica del sistema de ficheros subyacente, se realizan las dos consultas de la caché de ficheros: get_block(*F*, 0), que produce un acierto terminándose el procesamiento de ese bloque, y get_block(*F*, 1), que causa un fallo generándose una llamada a la función del sistema de ficheros subyacente solicitando ese bloque ([bloque 1 del fichero]), siendo *F* el identificador de fichero (puede corresponder a un identificador del sistema de ficheros donde está almacenado el fichero, junto con el número de *inodo* que tiene el fichero dentro de ese sistema de ficheros; ambos datos están almacenados en el *vnodo*).
En cuanto al cuarto paso, es igual que con el diseño original (aunque en este caso, sólo se ha solicitado un bloque), pero, una vez realizada la traducción usando el *inodo*, en vez de terminar con un acceso a la caché, realiza directamente una llamada a la capa general de entrada/salida de bloques solicitando el bloque [bloque 1001 del dispositivo].
Como puede apreciarse analizando las dos trazas, en el diseño original, antes de acceder a la caché hay que contactar con el nivel subyacente para que indexe el *inodo* y acceda, si es necesario, a los bloques indirectos para realizar la traducción. En el nuevo diseño no son necesarias todas estas operaciones. Por tanto, cuando se produce un acierto en la caché, el nuevo diseño es más eficiente.

c) En un sistema de ficheros distribuido, como NFS, en la máquina que actúa como cliente estará instalado debajo del VFS un nuevo tipo de sistema de ficheros que, a diferencia de los sistemas de ficheros de carácter local, cuando recibe una petición del sistema de ficheros virtual, no realiza ningún acceso a un dispositivo, sino que contacta con el servidor solicitándole los bloques del fichero requeridos. Por tanto, en este caso, el sistema de ficheros subyacente no realiza ninguna traducción de bloques del fichero a bloques del dispositivo, puesto que los datos no residen localmente, y el *inodo* gestionado por este tipo de sistemas de ficheros de red no contiene información de ubicación. Normalmente, basta con almacenar en el *inodo* un identificador del fichero remoto (algo como: dirección de la máquina remota, identificador del dispositivo y número de *inodo* dentro del dispositivo) que se usará cuando el cliente contacta con el servidor.
Este esquema de operación es difícil de encajar con el diseño de caché original de UNIX, puesto que en la máquina cliente no se llega al nivel de dispositivo durante el procesamiento del acceso a un fichero. Una posible solución es que este tipo de sistemas de ficheros de red no utilice la caché global de dispositivos del sistema operativo, sino una propia adaptada a sus características. Una solución alternativa es usar algún tipo de artificio para crear una especie de seudo-dispositivo asociado a los ficheros remotos, de forma que se pueda seguir utilizando la caché global de dispositivos para ellos.
El nuevo diseño de caché encaja bien en este modo de operación. El sistema de ficheros virtual hará la consulta en la caché de ficheros, como ocurre con cualquier fichero, y el nivel subyacente sólo tendrá que contactar con el servidor.

d) Cuando se produce una escritura en una zona del fichero que no tiene espacio asignado, ya sea porque se están añadiendo datos al final del fichero o debido a que se está escribiendo sobre un hueco, se busca un bloque libre en la caché y se copian en el mismo los datos.
Con el diseño original, en ese momento hay que asignarle un bloque libre del dispositivo, que conlleva una búsqueda en el mapa de bits de bloques libres y una actualización del puntero correspondiente del *inodo*, puesto que todo bloque en la caché tiene que estar identificado por el dispositivo al que pertenece y el número de bloque dentro del mismo.
En el nuevo diseño, cada bloque en la caché se identifica por el fichero al que pertenece y el número de bloque dentro del mismo. Por tanto, no sería necesario asignarle en ese instante un bloque libre del dispositivo. Esta asignación de espacio puede diferirse hasta el momento en el que vaya a escribirse el bloque al dispositivo, ya sea porque resulta expulsado o por el volcado periódico de los datos modificados. De esta forma, muchos ficheros temporales no sólo no requerirán escribir sus datos al dispositivo, como ya ocurría en el diseño original gracias a la política de escritura diferida, sino que tampoco necesitarán que se les asigne espacio del dispositivo.

e) Cuando se accede a un fichero especial, el procesamiento de la operación se simplifica puesto que el bloque *N* del fichero se corresponde con el bloque *N* del dispositivo, no siendo necesaria ninguna etapa de transformación.
Para analizar el problema de coherencia planteado en el enunciado, supongamos que el bloque *X* del fichero con identificador *F* se corresponde con el bloque *Y* del dispositivo con identificador *D*, y que en un determinado momento dos procesos están accediendo a ese mismo bloque de las dos formas, es decir, a través del fichero con identificador *F* y mediante el fichero especial de dispositivo, respectivamente.
En el diseño original, ambos accesos producirían la misma consulta de la caché (get_block(*D*, *Y*)), de forma directa en el caso del fichero especial, y después de una fase de transformación en el caso del fichero con identificador *F*. No existe, por tanto, ningún problema de coherencia.
Con el nuevo diseño, el acceso a través del fichero con identificador *F* causa la consulta de la caché get_block(*F*, *X*)), mientras que el acceso mediante el fichero especial genera la operación sobre la caché get_block(*D*, *Y*). Por tanto, si no se toma ninguna precaución adicional, acabarán existiendo dos bloques en la caché que se corresponden con el mismo bloque físico, con los consiguientes problemas de coherencia.

f1) La primera operación ftruncate, que especifica un tamaño igual a 0, consulta el *inodo* del fichero, al que ha llegado a través del descriptor de fichero consultando el BCP y la tabla intermedia, y libera todos los bloques a los que hace referencia el *inodo*, tanto los punteros directos como los indirectos. Todos los bloques liberados, ya sean de datos o de punteros, son marcados como tal en el mapa de bits de bloques. Asimismo, se actualiza el *inodo* poniendo el tamaño del fichero a cero y todos los punteros como nulos, así como se ajusta la fecha de modificación del fichero almacenada en el mismo.
La segunda operación ftruncate, que indica un tamaño igual a 8192, sólo tiene que acceder al *vnodo*, a través de la tabla de ficheros abiertos del BCP y de la tabla intermedia, y ajustar el nuevo tamaño y la fecha de modificación. No es necesario asignar espacio para los dos bloques, puesto que esa asignación sólo se requiere en el momento en el que se escriban datos en los mismos.

f2) En el caso del diseño de caché original, no es necesario invalidar los bloques de la caché que corresponden a los bloques truncados. Para entender por qué motivo no se producen problemas de coherencia aunque no se invaliden, hay que tener en cuenta que esos bloques no pueden volver accederse hasta que sean reasignados, puesto que no se les hace referencia desde ningún *inodo* (realmente, se pueden seguir accediendo a través del fichero especial que representa al dispositivo, pero esto no causa problemas de coherencia puesto que los bloques como tal existen aunque no pertenezcan a ningún fichero y un programa que los acceda utilizando el fichero especial pretende trabajar directamente con los bloques). Cuando alguno de estos bloques truncados pero presentes en la caché sea reasignado a otro fichero, el sistema operativo tendrá que actualizar su contenido conforme a los datos del nuevo fichero, pero esta actualización, como cualquier otra, se hará sobre la caché, con lo que el bloque almacenado en la misma quedará actualizado.
El nuevo diseño de caché sí presenta problemas de coherencia si no se invalidan en la caché los bloques truncados. En el ejemplo planteado, después de las dos operaciones ftruncate, un acceso al primer bloque del fichero puede encontrar todavía ese bloque en la caché a pesar de haber sido borrado en la primera operación ftruncate, produciéndose un problema de coherencia puesto que se leen los datos borrados.