Diseño de Sistemas Operativos
Febrero de 2002
Problema de Sistema de Ficheros

ENUNCIADO

Los dispositivos "*Disk On Chip*" (DOC) son físicamente un chip (sin partes móviles) que presentan un interfaz equivalente al de un pequeño disco (de 8 Megabytes). El almacenamiento interno de los datos se realiza sobre memoria no volátil de tecnología *flash*.

La geometría interna de un DOC distingue dos unidades distintas:

- El **sector** (de 0,5 Kilobytes). Un DOC puede ser considerado como un vector de sectores.
- La **región** (de 32 Kilobytes). Un DOC puede ser considerado como un vector de regiones.

Las operaciones que se pueden realizar sobre un DOC son:

- **Borrado** de una **región**. Borra de un golpe todos los sectores de la región. Los sectores no pueden ser borrados independientemente. Esta es una operación lenta, que degrada el medio. Una región que sea borrada 1 millón de veces empezará a fallar.
- **Lectura** de un **sector**. Un sector puede ser leído cualquier número de veces.
- **Escritura** de un **sector**. Sólo se puede escribir sobre un sector borrado. Tal sector no podrá rescribirse hasta que se borre la región que lo contiene.
- **Estado** de un **sector**. Informa sobre si está borrado o ya ha sido escrito.

Un Sistema de Ficheros para un DOC, ha de tener en cuenta las siguientes consideraciones de diseño:

- Dado que los DOC son pequeños, se debe aprovechar al máximo su capacidad de almacenamiento.
- Dado que el medio se degrada y para maximizar su vida útil, es necesario minimizar el número de borrados, así como repartir estos lo más uniformemente posible sobre el conjunto de regiones.
- Dado que no se puede rescribir *in-situ* y para proteger la integridad del Sistema de Ficheros, hay que evitar que la estructura de información escrita en el DOC pase por estados inconsistentes.

De cara a adaptar un Sistema de Ficheros tipo UNIX (UFS) convencional a un DOC, conteste razonablemente a las siguientes cuestiones:

1) Con los datos mencionados, calcule: número de regiones del DOC, número de sectores por región, número de sectores en el DOC, y número de bits por sector.

2) ¿Cuál sería el tamaño de bloque adecuado para este sistema? Razone pros y contras para los casos extremos: 1 bloque = 1 sector ó 1 bloque = 1 región.

3) Justifique la conveniencia de usar la cache de bloques para un dispositivo de tipo DOC. ¿Cuáles serían las políticas más adecuadas?

4) Indique qué estructuras de datos de un UFS convencional están dispuestas en posiciones fijas del dispositivo, y dónde se indica dicha posición, es decir, qué otra estructura de datos la referencia.

5) Suponiendo un UFS convencional, indique qué accesos de lectura y/o escritura son necesarios para crear un subdirectorio del directorio raíz.

6) Dadas las particularidades del DOC y la lógica del Sistema de Ficheros, ¿en qué estados puede encontrarse un bloque? ¿Cómo determinaría el estado de un bloque?

7) Describa en qué circunstancias podría ser borrada una región sin que se comprometa la consistencia del Sistema de Ficheros.

8) Enumere las características de un UFS convencional que hacen que no sean aptos para este tipo de dispositivos. Para cada una, esboce una solución.

SOLUCIÓN

1)
Datos:

1 DOC	=	8 Mbytes	=	2^23 bytes				
1 Sector	=	0,5 Kbytes	=	2^9 bytes	=	2^10 / 2	=	512 bytes
1 Región	=	32 Kbytes	=	2^15 bytes				

Cálculos:

Los dispositivos "Disk On Chip" (DOC) son físicamente un chip (sin partes móviles) que presentan un interfaz equivalente al de

Regiones / DOC	=	2^23 / 2^15	=	2^(23-15)	=	2^8	=	256
Sectores / DOC	=	2^23 / 2^9	=	2^(23-9)	=	2^14	=	16 K
Sectores / Región	=	2^15 / 2^9	=	2^(15-9)	=	2^6	=	64
Bits / Sector	=	2^9 * 2^3	=	2^(9+3)	=	2^12	=	4 K

2)
Consideraciones iniciales:

- Sobre la eficiencia de las transferencias de datos.
  El tamaño del bloque **no** afecta a la eficiencia. Las operaciones de lectura y escritura se realizan sobre sectores individuales. Dado que el dispositivo **no** tiene partes móviles, el tiempo de acceso al medio es uniforme, esto es, se tardaría **exactamente** lo mismo en leer n sectores dispersos que consecutivos.
- Sobre la re-escritura y el borrado.
  Dada la naturaleza del DOC, sólo se puede escribir sobre sectores que estén "borrados". Esto hace imposible la re-escritura *in-situ* de un sector. Es preciso encontrar sectores borrados dónde poder escribir. Esto obliga a localizar Regiones (conjuntos de sectores) que puedan ser borradas. Esto *puede* condicionar la complejidad del Sistema de Ficheros.

Considerando 1 Bloque = 1 Sector:
Pros:

- Minimiza la fragmentación interna (espacio asignado no aprovechado) que siempre se estima como ½ bloque por fichero.
- Supone el mayor número posible de bloques: 16 K.

Contras:

- Un mayor número de bloques implica mayor tamaño para las estructuras de metadatos. Por ejemplo, harían falta 4 bloques (de 512 bytes) para contener el mapa de bits de bloques de 16 Kbits.
- Como se indicó anteriormente, para escribir un sector se precisa disponer de uno en estado borrado ya que es imposible la re-escritura *in-situ*. Esto obliga al Sistema de Ficheros ha realizar una gestión complicada (pero imprescindible) de la disposición de la información sobre el disco.
- Aumenta la dispersión de los datos. La información lógicamente próxima se dispersa. Pero dado que el dispositivo ofrece tiempos uniformes de acceso al medio este punto resulta **irrelevante**.

Considerando 1 Bloque = 1 Región. Es el caso diametralmente opuesto al anterior.
Pros:

- Se reduce la dispersión de datos, pero ya hemos dicho que esta es **irrelevante**.
- Al coincidir las unidades de transferencia y borrado, el Sistema de Ficheros podría simular las re-escrituras con la secuencia: borrado + escritura. Así se hace trivial la adaptación de cualquier SF a este tipo de dispositivos. Es necesario hacer notar que esta (aparente) ventaja sólo se dará para tamaños de bloque potencia de la región y nunca para tamaños menores que esta.

Contras:

- No obstante, hacer un borrado + escritura por cada re-escritura deseada, maximiza el número total de borrados. Además, estás serían *in-situ* (en posiciones fijas), lo cuál maximiza la degradación de la región y como consecuencia, la vida útil del DOC.
- Aumenta la fragmentación interna. Se desperdician por termino medio 16 Kbytes por fichero, y como se ve a continuación, mucho más en el caso de bloque de metadatos.
- Supone un total de 256 bloques disponibles. Esta cifra es **ridícula** para un dispositivo de 8 Mbytes.
- El número de bloques es muy reducido, pero esto **no reduce, sino que desperdicia** el espacio necesario para almacenar as estructuras de metadatos. Por ejemplo, haría falta 1 bloque (de 32 Kbytes) para contener el mapa de bits de bloques de 256 bits (= 32 bytes). ¡1024 veces más!

Por lo tanto, en ningún caso se justifica la opción 1 boque = 1 región. Todo apunta a escoger el tamaño de bloque menor posible, un sector. Ahora bien, ¿es ésta la solución óptima?

De cara a aprovechar al máximo la capacidad de almacenamiento del DOC, hemos de reducir al mínimo el espacio necesario para los metadatos, información imprescindible para su correcta gestión. Luego la pregunta es:

¿Cuánto espacio ocuparán los metadatos?
Para responder numéricamente haremos las siguientes suposiciones:

- Todas las estructuras ocuparán un número entero de bloques.
- Las estructuras que consideraremos y su tamaño son:
  - (BB) Bloque de Boot. 1 bloque.
  - (SB) Superbloque: 1 bloque.
  - (MBB) Mapa de Bits de Bloques. Con 1 bit por bloque del dispositivo.

Los dispositivos "Disk On Chip" (DOC) son físicamente un chip (sin partes móviles) que presentan un interfaz equivalente al de

  - (MBI) Mapa de Bits de Inodos. Con 1 bit por inodo en el sistema de ficheros (normalmente tantos como bloques).
  - (VI) Vector de Inodos. Con inodos de 32 bytes, tamaño razonable dado el tamaño del dispositivo.
- Calcularemos la proporción ocupada por metadatos para los tamaños de bloque: 1, 2 y 4 sectores.

Para 1 Bloque = 1 Sector.

Bloques / DOC	=	2^14						
Bloques / Bitmap	=	2^14 / (2^9 * 2^3)	=	2^(14-9-3)	=	2^2	=	4
Bloques / V. Inodos	=	2^14 / (2^9 / 2^5)	=	2^(14-9+5)	=	2^10	=	1024
Bloques / Metadatos	=	1 + 1 + 4 + 4 + 2^10	=	x	=	1034	~=	2^10
Los Metadatos son	=	2^10 / 2^14	=	2^(10-14)	=	1/2^4	=	1/16

Para 1 Bloque = 2 Sectores = 1 Kbyte

Bloques / DOC	=	2^13						
Bloques / Bitmap	=	2^13 / (2^10 * 2^3)	=	2^(13-10-3)	=	2^0	=	1
Bloques / V. Inodos	=	2^13 / (2^10 / 2^5)	=	2^(13-10+5)	=	2^8	=	256
Bloques / Metadatos	=	1 + 1 + 1 + 1 + 2^8	=	x	=	260	~=	2^8
Los Metadatos son	=	2^8 / 2^13	=	2^(8-13)	=	1/2^5	=	1/32

Para 1 Bloque = 4 Sectores = 2 Kbytes.

Bloques / DOC	=	2^12						
Bloques / Bitmap	=	2^12 / (2^11 * 2^3)	=	2^(12-11-3)	=	1/2^2	~=	1
Bloques / V. Inodos	=	2^12 / (2^11 / 2^5)	=	2^(12-11+5)	=	2^6	=	64
Bloques / Metadatos	=	1 + 1 + 1 + 1 + 2^6	=	x	=	70	~=	2^6
Los Metadatos son	=	2^6 / 2^12	=	2^(6-12)	=	1/2^6	=	1/64

Como se puede apreciar, a medida que el tamaño del bloque se dobla, el número de ellos se reduce a la mitad, y el espacio destinado a inodos también. Dado que esta es la parte más voluminosa de los metadatos, estos también se reducen aproximadamente a la mirad.
Así mismo se puede observar que a partir del tamaño de 2 Kbytes empezamos a tener que utilizar todo un bloque para almacenar un bitmap que sólo ocupa una parte de él.

La elección más prudente estaría alrededor de estas cantidades, pero para decidir habría que conocer un dato que no se da, el tamaño medio de los ficheros. Ante la ausencia de éste cabría optar por 1 Bloque = 1 Kbyte.

3)
Cuando hablamos de la "Cache de Bloques" **no** nos estamos refiriendo a una memoria cache (asociativa), sino a la estructura de datos, gestionada por el Servidor de Ficheros y construida sobre memoria principal (convencional), que permite aumentar las prestaciones del sistema de Entrada / Salida.

De cara a las operaciones de lectura, y dado que la velocidad de acceso al DOC es alta (aunque **no** la de la memoria principal), podría pensarse que resulta poco útil. No obstante siempre resultará eficaz.

La principal utilidad de la Cache de bloques está en las operaciones de escritura. Dada la característica de no re-escribible *in-situ* del DOC, cada escritura consume un bloque en estado “borrado”. Tarde o temprano, estos se agotan y habrá que empezar a borrar regiones **no** en uso para obtener espacio.

Es por lo tanto, imprescindible reducir al máximo las operaciones de escritura (y consecuentemente los borrados) sobre el DOC. Para ello utilizaremos la Cache de bloques (cuanto mayor mejor) con una política de escritura *write-back* o *delayed-write*, de manera que se amortigüen al máximo el número de escrituras reales sobre el DOC.

4) La estructura de un UFS convencional es:

[BB][SB][MBB___][MBI___][VI____][Bloque de datos_____]

- (BB) Bloque de Boot.
  - 1 bloque.
  - Puede contener código ejecutable para el arranque del sistema.
  - Su posición es fija, el primer bloque de cada dispositivo o partición.
  - Su posición y tamaño no se indican en ningún sitio. Se sabe que está ahí.

Los dispositivos "Disk On Chip" (DOC) son físicamente un chip (sin partes móviles) que presentan un interfaz equivalente al de

- (SB) Superbloque:
  - 1 bloque.
  - Contiene la identificación del tipo de UFS así como las referencias que indican dónde se encuentran el resto de los campos del UFS (MBB, MBI, VI, Datos). En cierta manera es la raíz de la estructura de datos que el Sistema de Ficheros es.
  - Se encuentra a continuación del BB. Al ser de vital importancia, también puede encontrarse replicado en otras posiciones preestablecidas.
  - Su posición y tamaño no se indican en ningún sitio. Se sabe que está ahí.
- (MBB) Mapa de Bits de Bloques.
  - Es un vector de bits con 1 bit por bloque del dispositivo. Ocupa un número entero de bloques.
  - Indexando en él con un número de bloque accederemos primero al bloque que lo contiene y luego al bit correspondiente y podemos saber si tal bloque está libre (0) u ocupado (1).
  - Normalmente se haya **ubicado de forma contigua**, justo a continuación del SB.
  - Quien indica su posición y tamaño es el SB.
- (MBI) Mapa de Bits de Inodos.
  - Es un vector de bits con 1 bit por inodo del sistema de ficheros. Ocupa un número entero de bloques.
  - Indexando en él con un número de inodo accederemos primero al bloque que lo contiene y luego al bit correspondiente y podemos saber si tal inodo está libre (0) u ocupado (1).
  - Normalmente se haya **ubicado de forma contigua**, justo a continuación del MBB.
  - Quien indica su posición y tamaño es el SB.
- (VI) Vector de Inodos.
  - Es un vector de inodos. Es decir, aquí se hayan dispuestos, uno detrás de otros todos los inodos del sistema. Ocupa un número entero de bloques.
  - El inodo se diseña para que ocupe una potencia de 2 bytes, de manera que haya una potencia de 2 de inodos por bloque.
  - Indexando en él con un número de inodo accederemos, primero al bloque que lo contiene, y luego al inodo en sí.
  - Normalmente se haya **ubicado de forma contigua**, justo a continuación del MBI.
  - Quien indica su posición y tamaño es el SB.
- (BBDD) Bloques de Datos.
  - El resto del dispositivo se utiliza para almacenamiento de datos de ficheros y/o directorios (también bloques de indirección de inodos).

Así pues, como hemos visto, las dos primeras estructuras (BB y SB) se hayan dispuestas en posición fija conocida de forma implícita. Las tres siguientes (MBB, MBI y VI) se encuentran ubicadas de forma contigua en una posición y tamaño indicadas en el Superbloque.

Ahora bien, estas **no** son las únicas referencias (direcciones de otras estructuras) que contiene un UFS. Cada vez que se indica un número de bloque o un número de inodo (aunque esta última referencia siempre es relativa y se usa para indexar el Vector de inodos), se está refiriendo a otra estructura dispuesta en posición fija.

Así, la estructura lógica de la información contenida en el Sistema de Ficheros está establecida como un árbol con su raíz en el inodo raíz (indicado como tal en el SB), y de las entradas de éste directorio, al resto de directorios y ficheros del sistema. Cada entrada de directorio indica a un inodo concreto y cada inodo, a través de sus campos de localización, indica las posiciones fijas de los bloques que ocupa dicho fichero o directorio.

5)
Tomando como punto de partida que el UFS está montado y su Superbloque está en memoria, los accesos necesarios para crear un subdirectorio del directorio raíz será:

1. **1** acceso de lectura del inodo del directorio raíz (normalmente el número 2). También es posible que la operación de montado lo haya traído a memoria y bloqueado allí.
2. **n** accesos de lectura del contenido **completo** del directorio raíz, accediendo a través de los campos directos, indirectos, etc. del inodo, buscando una entrada de directorio libre, así como comprobando que la que queremos crear no existe ya. Si existiese se aborta la operación.
3. **n** accesos de lectura del mapa de bits de inodos, buscando uno libre para el nuevo subdirectorio. Si no se encuentra uno libre, la operación se aborta.
4. **1** acceso de escritura del bloque del mapa de bits de inodos donde hemos marcado el bit correspondiente como ocupado.
5. **n** acceso de lecturas del mapa de bits de bloques, buscando uno libre para el nuevo subdirectorio. Si no se encuentra uno libre, la operación se aborta.
6. **1** acceso de escritura del bloque del mapa de bits de bloques donde hemos marcado el bit correspondiente como ocupado.

Los dispositivos "Disk On Chip" (DOC) son físicamente un chip (sin partes móviles) que presentan un interfaz equivalente al de

7. **1** acceso de escritura del bloque ubicado, una vez introducidas en él las entradas de directorio “.” y “..” apuntando a sí mismo y al raíz respectivamente.
8. **1** acceso de escritura del inodo ubicado (del bloque que lo contiene), una vez asignado a él el nuevo bloque de datos con sus entradas de directorio.
9. **1** acceso de escritura del bloque del directorio raíz, una vez introducida en él la nueva entrada apuntando al nuevo subdirectorio.
10. **1** acceso de escritura del inodo raíz (del bloque que lo contiene) una vez actualizado en él el campo número de enlaces, así como el tiempo de última modificación.
11. **1** posible acceso de escritura del SB, si en él se refleja información sobre el número de bloque o inodos disponibles en cada momento, una vez actualizado en él dicha información.

6)
Se pregunta por los posibles estados de un bloque. Es preciso hacer notar, que el bloque es la unidad de acceso (y de ubicación, ya que no lo hemos distinguido del concepto de zona) del Sistema de Ficheros, y por tanto, todos los sectores que lo compongan tendrán siempre el mismo estado. Este estado será el del bloque.

Dadas las particularidades del DOC, y basándonos en la operación de “Estado de un sector” un bloque puede estar:

- Borrado
- Escrito

Según la lógica habitual del Sistema de Ficheros, y consultando el correspondiente bit del MBB, un bloque puede estar:

- Libre, bit = 0.
- Ocupado, bit = 1.

De la combinación de ambas visiones surge el siguiente diagrama de los estados en los que puede estar un bloque:

Los estados resultantes son 4:

1. Virgen = Borrado + Libre. Es un bloque que puede ser reservado (bit = 1) o vuelto a borrar (aunque ya lo está).
2. Reservado = Borrado + Ocupado. Es un estado transitorio previo a la escritura del mismo.
3. En Uso = Escrito + Ocupado. Hay en él información útil y actual. Cuando dejemos de usarlo los liberaremos (bit = 0).
4. Gastado = Escrito + Libre. Es un estado transitorio previo al borrado de la región que lo contiene.

Las transiciones entre estados suceden con las acciones indicadas:

- Borrado, de la región que contiene el bloque.
- Escritura del bloque.
- bit = 1/0. Asignación al bit del mapa de bits de bloques del Sistema de Ficheros.

Las transiciones no indicadas en el diagrama de estados darían lugar a errores (Ej. Re-escritura en el estado Gastado) o a inconsistencias en el Sistema de Ficheros (Ej. Borrar un bloque En Uso o Reservado).

7)
Del diagrama anterior se deduce claramente que sólo se *debe* borrar una región tal que todos sus bloques estén en los estados Gastado o Virgen, y

que estos estados se distinguen de los otros dos posibles porque en el mapa de bits los bloques están marcados como libres.

8)
De todas las consideraciones hechas hasta el momento, podemos resumir en los principales inconvenientes que dificultan la adaptación de un Sistema de Ficheros UFS a un dispositivo tipo DOC en:

- El UFS mantiene múltiples estructuras de control ubicadas en posiciones fijas y de forma contigua.
- En el DOC es imposible de re-escritura *in-situ* de un bloque.
- Con el DOC se necesita borrar regiones (de un tamaño potencia del bloque) para obtener espacio ubicable.

Para plantear una solución factible, hay que darse cuenta de que re-ubicar un bloque de un fichero no plantea problemas. Dado que la disposición física de los bloques de los ficheros nunca trasciende más allá del propio Sistema de Ficheros, esto es, para referirnos a ellos siempre utilizamos su numeración lógica, si deseamos reubicar (disponer en otra posición física) un bloque de un fichero, no tendríamos más que indicar la nueva posición en el correspondiente campo de su inodo.

El problema está las estructuras de datos de Mapas de bits y en el Vector de inodos, que se hayan dispuestas en posición fija y ubicadas de forma contigua. Para poder reubicar estas estructuras y que su disposición física no sea siempre la misma, hay que introducir un nivel más de indirección en el camino de acceso a las mismas, al estilo de lo que inodo supone para los bloques de los ficheros.

Para el caso del Vector de Inodos, se utilizaría una especie de Super-Inodo, que referenciado desde el Superbloque nos permitiría el acceso a los inodos, así como su ubicación en cualquier posición física, esto es, en cualquier bloque. De manera equivalente podría hacerse para cada uno de los Mapas de bits, si es que estos efectivamente ocupan un número elevado de bloques. Si ocupasen un número reducido de bloques, podríamos indicar en el mismo Superbloque la dirección de cada uno de los bloques ocupados por estos Mapas de Bits.

De esta manera, desplazamos el problema a la disposición fija que normalmente ocupa el propio Superbloque. La disposición física de éste podría escogerse, de forma circular, de entre un conjunto de posiciones prefijadas acompañándolo de un de un número de secuencia que al montar el Sistema de Ficheros nos permita localizar la última versión del mismo.

Respecto al problema de obtener espacio ubicable borrando regiones. La solución pasa por realizar esta tarea de forma asíncrona. Normalmente usaríamos un proceso de fondo (*daemon* o *thread* del *kernel*) que trabajaría como un recolector de espacio. Leyendo los ficheros que ocupan bloques en uso, para luego volver a escribirlos en espacio vacío, y así conseguir regiones completas susceptibles de ser borradas.

-----Fin.