Una de las estrategias de incremento más importantes en los dispositivos de entrada/salida es la lectura o escritura de bloques continuos de disco. Dicha estrategia afecta a cómo el sistema de archivos en un modelo UNIX debe realizar ciertas operaciones.

a. ¿Por qué la lectura de 2MB en varias operaciones de 4KB sobre la geometría del disco es menos eficiente que la lectura de esa misma cantidad de información en una misma región del disco? ¿Quién influye en esta diferencia de rendimiento (servidor de archivos, servidor de bloques o dispositivo de E/S)?

b. La posibilidad de aplicar mejoras a este nivel está relacionada con la ubicación de los bloques asociados a un fichero sobre el *layout* del disco. ¿Qué estructuras de metainformación usa el servidor de archivos para realizar la ubicación de nuevos bloques? ¿Cuál es la política de actualización de estas estructuras de memoria a disco?

   Considerando la estructura de un i-nodo típico (10 punteros directos, 1 indirectos, simple, doble y triple, direcciones de i-nodo y de bloque de 4 bytes), se proponen dos alternativas diferentes para intentar conseguir que las operaciones sobre el disco se hagan sobre regiones continuas de mayor espacio:

- Dimensionamiento de la unidad mínima de asignación (agrupación o bloque lógico) a un tamaño mayor (de 4KB a 512KB).
- Modificación de la política de asignación de bloques a disco, de forma que cuando se tiene que reservar un bloque en el dispositivo se intente siempre hacerlo pegado al bloque anterior.

c. Si tenemos 2048 ficheros de menos de 4KB, ¿cuánto espacio de disco se ocuparía en bloques de datos en cada alternativa?

d. ¿Cuántas veces se deberán consultar las estructuras de metainformación para la ubicación de bloques de datos si queremos escribir un fichero de 100MB? (Determínelo para cada alternativa de diseño).

   En la segunda de las alternativas de diseño se plantean problemas cuando hay varios procesos escribiendo de forma simultánea diferentes ficheros. No tenemos la garantía de que la asignación de bloques a un fichero sea contigua, debido a que se van intercalando las peticiones de ubicación de bloques y que no sabemos, *a priori*, el tamaño final del fichero.

   Para manejar el problema anterior, en ciertos sistemas de ficheros (ext4, XFS o ZFS), se aplican estrategias de ubicación retrasada (*delayed allocation*) que implica que a un bloque de datos no se le asigna su posición en disco hasta que no se vuelca de la cache de bloques a disco.

c. ¿En qué momento y bajo qué política se realiza el volcado de la cache de datos? ¿Cómo repercutiría la estrategia de *delayed allocation* en la mejor ubicación de los bloques de un mismo fichero.

d. Esta política se complementa con un mecanismo de solicitud de ubicación multibloque (*multiblock allocation*). ¿Qué ventajas de eficiencia tiene poder solicitar la ubicación de varios bloques a la vez, respecto de la solicitud de ubicación de ellos uno a uno? ¿Dónde podría estar ganándose tiempo?

Supongamos que se quiere hacer uso de la funcionalidad de *multiblock allocation* de otra forma, que sería sabiendo con antelación el tamaño del fichero (el caso de aplicaciones del tipo P2P). La opción alternativa a esa funcionalidad puede ser de dos tipos:

i. Que la aplicación que sabe el tamaño del fichero reserve todo su espacio (creando un fichero del tamaño correspondiente lleno de bytes a cero), y según se vayan generando los datos, se van sobreescribiendo las partes del mismo.

ii. Que no se reserve espacio efectivo alguno, se anota sólo que el fichero es de un cierto tamaño y en cuanto llegue el primer bloque de datos (en una posición intermedia) se hace la reserva de los bloques datos y de los bloques de punteros indirectos necesarios.

   Considerando la alternativa de bloques de datos de 4KB:

e. Supongamos de nuevo el caso del fichero de 100MB, queremos dimensionar el espacio para él con cada una de estas alternativas ¿Cuántos bloques habría que escribir en disco para la operación de reserva de espacio? (estudie el caso para cada opción *multiblock allocation*, i) e ii))

f. Si ahora tenemos que escribir 128KB correspondientes a la posición 43 x 2^20, ¿cuántos bloques de datos habría que escribir? (estudie el caso para cada opción *multiblock allocation*, i) e ii))

Solucion

a) La lectura de bloques de disco dispersos por diferentes partes del dispositivo es mucho menos eficiente que la lectura de un bloque del mismo tamaño de forma consecutiva. El motivo es la latencia debida al posicionamiento de las cabezas del disco en los sectores correspondientes a leer. Esa latencia se multiplica por el número de veces que hay que posicionarse en una región diferente del disco. Si todos los bloques son consecutivos esta latencia es única para toda la lectura.

Dentro de lo que son los módulos del sistema operativo para la gestión de archivos, es el dispositivo de E/S el que marca la diferencia del rendimiento.

b) El servidor de archivos utiliza los bimaps (de bloques e i-nodos) para determinar dónde hay estructuras (bloques e i-nodos, respectivamente) disponibles.

Todas la metainformación se actualiza entre memoria y disco en base a una política *write-through*. Otras alternativas presentan problemas de cara a eventuales pagaos no controlados de la máquina.

c) En un sistema de archivos de tipo UNIX estándar se almacena, para cada archivo 1 i-nodo, y si el archivo es de tamaño mayor que cero, como mínimo un bloque. El bloque es a todos los efectos la unidad mínima de asignación de espacio para archivos. Como los ficheros ocupan menos de 4KB todos ellos deben ocupar un bloque entero (es lo mínimo a asignar).

Así pues, para cada alternativa el espacio en bloques de datos ocupado es:

- 2048 ficheros x 1 bloque/fichero x 512KB/bloque = 1GB
- 2048 ficheros x 1 bloque/fichero x 4KB/bloque = 8MB

La modificación de la política de asignación no afecta, para nada, en el espacio ocupado por los ficheros.

d) Antes de nada resulta imprescindible saber cuántos bloques de datos ocupa el fichero de 100MB en cada alternativa (es decir para cada tamaño de bloque):

- 100MB / 512KB/bloque = 200 bloques
- 100MB / 4KB/bloque = 100 2^20 / 2^12 = 100 x 2^8 = 25.600 bloques

Hemos mencionado antes que las estructuras de metainformación asociadas a la búsqueda de bloques de datos libres son los bitmaps. El bitmap se consultará para cada bloque de datos que se quiera ubicar en disco.

Ahora bien, resulta imprescindible contabilizar no sólo los bloques de datos de información, sino los usados como

punteros intermedios. En el modelo de i-nodo que se presenta se habla de 10 punteros directos y 1 simple, doble y triple. Eso hace que en cualquiera de las dos alternativas (usan más de 10 bloques) estemos necesitando más o menos punteros indirectos. Para saber cuántos vamos a contabilizar cuántos punteros indirectos entran en un bloque de indirección de cada alterativa, es importante reseñar que los bloques de indirección son del mismo tamaño que los bloques de datos (512KB en un caso y 4KB en el otro):

- 512KB/bloque / 4bytes/dirección-bloque= 2^19/2^2 = 2^17direcciones-bloque/bloque > 128.000 direcciones-bloque/bloque
- 4KB/bloque / 4bytes/dirección-bloque= 2^12/2^2 = 2^10direcciones-bloque/bloque = 1024 direcciones-bloque/bloque

Eso nos indica que en el caso de la alternativa de 512KB de tamaño de bloque nos vale con un solo bloque de indirección: 200 bloques de fichero, 10 en punteros directos y 190 en un bloque indirecto simple. En resumen, la alternativa de bloque de 512KB necesita 201 bloques:

- 200 bloques de datos
- 1 bloque indirecto simple.

Para la alternativa de bloque de 4KB tendremos:

10 punteros directos: 10 bloques

1 bloque indirecto simple: 1024 bloques

X bloques en indirectos doble: 25600 – 1024 -10 = 24566 bloques

Para determinar cuántos bloques se necesitan por medio de indirección doble, dividimos las direcciones de bloques de datos que necesitamos por el número de direcciones por bloque de indirección: 24566direcciones-bloque / 1024direcciones-bloque/bloque = 23,9902 bloques → 24 bloques de indirección.

De este cálculo se deriva que se necesitan 24 bloques de indirección en el segundo nivel, referenciados por el bloque indirecto doble apuntado por el puntero correspondiente del i-nodo. En resumen, la alternativa de bloque de 4KB necesita 25600 + 1 + (1+24)=25626:

- 25600 bloques de datos
- 1 bloque indirecto simple (direcciona los 1024 bloques siguientes).
- 1 bloque indirecto doble que apunta a 24 bloques indirectos de segundo nivel (direccionan en total los 24566 bloques restantes)

Es importante resaltar que las consultas de metainformación se centran en proveer del número de bloques necesarios para almacenar los datos. En el enunciado se comenta que la política de asignación intentará reservar bloques consecutivos para un mismo fichero. Esa funcionalidad sólo se puede conseguir dándole a la función de reserva de bloque (la que consulta y actualiza el bitmap) un indicio del bloque preferido, algo del estilo “el último bloque que he usado es el 288, intenta darme el 289”. Si no se explicita que haya una reubicación de los datos cuando no se puede disponer de dicho bloque libre, no sería necesario contabilizar los acceso derivados de dicha reubicación de bloques.

e) La cache de datos en un sistema UNIX estándar se realiza por medio de una política de actualización retrasada o *delayed-write*. Esta política implica actualizar los datos modificados en memoria a disco cada 30 segundos.

La estrategia *delayed-allocation* pospone la reserva de espacio en disco (la ubicación del bloque) al momento en el que dicho bloque se escribe. El funcionamiento normal es que si un fichero escribe un bloque nuevo, dicho bloque se escribe en la cache (inicialmente) pero, a la vez, se reserva (usando el bitmap de bloques) dónde irá en el disco cuando se escriba. En esta nueva estrategia ese segundo paso se difiere indicando, de alguna manera en la cache que dicho bloque está *dirty* (no sincronizado con el disco) y *unallocated* (sin asignación de espacio en el disco). Al hacerlo de esta forma, en el momento en el cual el demonio o hilo de ejecución de núcleo se active y comience a sincronizar los bloques de datos se le asignará espacio en el dispositivo físico.

Esta nueva estrategia, per se, no implica una mejora en rendimiento si se siguen aplicando las mismas políticas en otros aspectos. Analicemos el efecto:

- Si tenemos en mente la intención de mejorar el rendimiento esto se puede conseguir haciendo que a la hora de asignar bloques a un fichero se intenten ubicar de forma contigua el mayor número de ellos posible.
- Si consideramos la forma en la que trabaja el hilo de sincronización de la cache, este hilo de núcleo **recorre secuencialmente** los bloques de la cache, busca los que están sucios (*dirty*) y los copia en disco (con la estrategia *delayed-allocation* además debe buscar dónde ubicar el bloque).
- Con el recorrido secuencial, como tal, no se mejora la ubicación de bloques en disco. Ya que el *slot* de la cache que usan los bloques no tiene relación con el fichero al que pertenecen (se van usando dependiendo de dónde se liberen bloques por parte de la política de reemplazo).
- Una mejora posible sería forzar al hilo de ejecución a seguir el orden de los ficheros abiertos (el vector de v-nodos, por ejemplo) y de ahí los bloques de cache que éstos usan. De esta forma se pasará a escribir en disco los bloques de la cache **fichero por fichero**, permitiendo con la estrategia de *delayed-allocation* ubicar en ese momento los bloques de forma consecutiva.

f) La existencia de mecanismos para realizar *multiblock-allocation* no implica que el sistema tenga mejor o peor capacidad para ubicar los bloques de forma consecutiva ya que el mismo algoritmo de ajuste se podría realizar dentro de la función que implementa ese mecanismo como externamente. La principal ventaja de tener una función para reservar varios bloques de golpe tiene que ver con la sincronización de los metadatos. Como hemos comentado antes los metadastos se sincronizan en disco por medio de una política *write-through*. Si tenemos que reservar 4 bloques uno a uno haremos que para cada bit modificado en el bitmap se tenga que escribir en disco. Si con una sola función es posible reservar 4 bloques, modifcando 4 bits del mapa y de una sola tacada actualizar esa información en disco una sola vez se ganaría mucho rendimiento reducinedo las escrituras en disco.

La implementación de una algoritmo más o menos sofistiado dentro del mecanismo de *multiblock-allocation*, aunque es posible, por lo general contradice lo que son los principios generales de diseño de un sistema operativo, separando políticas de mecanismos en la medida de lo posible. En cualquier caso, dicho algoritmo no sería más que la implementación de una función que tomando el mapabits como dato de entrada y el número de bloques requeridos, devuelve las posiciones a reservar. Dicha función sería utilizable dentro del mecanismo de *multiblock-allocation* o fuera.

g) Vamos a realizar las siguientes suposiciones de partida sobre el funcionamiento de cada alternativa:

- Usando *multiblock-allocation*: Se reservan los bloques que se van a usar en el disco, es decir que el i-nodo y los bloques indirectos apunta a números de bloque reservados, pero dichos bloques no se escriben (se marcarían como “a rellenar con 0's”).
- La alternativa i) no sólo reserva los bloques de disco sino que además los escribe, rellenándolos a 0's.
- La alternativa ii) marca que el fichero tiene un tamaño dado pero no reserva bloques, por lo tanto los punteros directos del i-nodo no apuntan a nada de momento y no se crean bloques de indirección.

  Si recordamos los datos de este fichro para el caso de bloques de 4KB tenemos:

- 25600 bloques de datos
- 1 bloque indirecto simple
- 1 bloque indirecto doble
- 24 bloques de indirección de segundo nivel (referenciados por el el indirecto doble).

Para estos cálculos no vamos a contar las modificaciones en el directorio anterior, puesto que serán iguales para todas las alternativas.

Para cada caso, la reserva de espacio será:

- Usando *multiblock-allocation*:
  - Se reservan los 25600 bloques de datos y los 26 de indirección en el bitmap: Esto idealmente puede suponer la escritura de un 1 sólo bloque de bitmap (cambiando 25600 bits de golpe). (Un bloque de bitmap tiene 4KB/bloque x 8 bits/byte = 2^12 x 2^3 = 2^15 >32.000 bits/bloque
  - Se solicita un nuevo i-nodo (bitmap de i-nodos)
  - Se escribe el i-nodo.
  - Se escribe el bloque indirecto.
  - Se escribe el bloque indirecto simple y los 24 del siguiente nivel.
  - TOTAL: 29 escrituras
- Rellenando a 0's (alternativa i)):
  - Se deben solicitar los 25600 + 26 bloques al bitmap. Si no hay multiblock-allocation esto son 25626 escrituras write-through de la cache.
  - Se solicita un nuevo i-nodo (bitmap de i-nodos)
  - Se escribe el i-nodo
  - Se escribe el bloque indirecto.
  - Se escribe el bloque indirecto simple y los 24 del siguiente nivel.

    - Se escriben los 25600 bloques de datos
    - TOTAL: 25625 + 1 + 1 + 1 + (1+24) + 25600 = 2x25626 +1 escrituras
- Modificando sólo el tamaño (alternativa ii)):
    - Se solicita un nuevo i-nodo (bitmap de i-nodos)
    - Se escribe el i-nodo
    - TOTAL: 2 escrituras

h) Si se van a escribir 128KB, debemos calcular cuántos bloques de disco son eso: 128KB / 4KB/bloque = 32 bloques

Un segundo aspecto a considerar es dónde se encuentran esos bloques. Se nos dice que se trata de la posición 43 x 2^20 (después de los 43 primeros megabytes). Eso implica que es una posición que es redireccionada por los punteros indirectos dobles. Con estos dos datos procedemos a estudiar el comportamiento de cada alternativa.

- Usando *multiblock-allocation*: Sólo se tendrán que escribir los bloques de datos a modificar, estos ya están reservados y sólo hay que modificar su contenido, por lo tanto: 32 escrituras.
- Alternativa i): Los bloques de datos estan reservados y además rellenos a 0's sólo habría que sobreescribir dichos bloques, así que también: 32 escrituras
- Alternativa ii): Además de los 32 bloques de datos es necesario reservar el bloque indirecto doble y el bloque de segundo nivel donde se referencian los 32 bloques de datos. Además necesitaremos reservar los bloques en el bitmap, que si no hay *multiblock-allocation* son 32+2 escrituras en el bitmap de bloques, cada una de las cuales hace write-through y se combierten escrituras directas a disco. El cómputo total es: 2x (32+2) escrituas.

CONSIDERACIONES: En la alternativa ii) se ha apliacado el comportomaiento típico de estas operaciones que implica sólo reservar los bloques intermedios que se han escrito, dejándose el resto de bloques (anteriores y posteriores) sin reservar.