Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

El sistema de ficheros virtual, page-cache y buffer-cache en Linux 2.4.10+

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

Creado el 05/11/2001 17:11 modificado el 05/11/2001 17:11

Continuando con el tema de page-cache y buffer-cache en Linux, en éste artículo doy una descripción, que aunque simple y general, un poco más detallada del funcionamiento del page-cache y buffer-cache en Linux. Como ya sabréis, a partir del 2.4.10 hubo dos cambios importantes, la nueva gestión de memoria virtual del kernel basado en una nueva implementación de Andrea Arcangeli y la unificación de las técnicas de buffer-cache y page-cache.

Introducción

Un *fichero* es una abstracción muy importante en programación. Los ficheros sirven para almacenar datos de forma permanente y ofrecen un pequeño conjunto de primitivas muy potentes (abrir, leer, avanzar puntero, cerrar, etc.). Los ficheros se organizan normalmente en estructuras de árbol, donde los nodos intermedios son directorios capaces de agrupar otro ficheros.

El sistema de ficheros es la forma en que el sistema operativo organiza, gestiona y mantiene la jerarquía de ficheros en los dispositivos de almacenamiento, normalmente discos duros. Cada sistema operativo soporta diferentes sistemas de ficheros. Para mantener la modularización del sistema operativo y proveer a las aplicaciones con una interfaz de programación (API) uniforme, los diferentes sistemas operativos implementan una capa superior de abstracción denominada *Sistema de Ficheros Virtual* (*VFS: Virtual File System*). Esta capa de software implementa las funcionalidades comunes de los diversos sistemas de ficheros implementados en la capa inferior.

Los sistemas de ficheros soportados por Linux se clasifican en tres categorías:

1. Basados en disco: discos duros, disquetes, CD-ROM. Estos sistemas son Ext2, ReiserFS, XFS, Ext3, UFS, ISO9660, etc.
2. Sistemas remotos (de red): NFS, Coda, y SMB.
3. Sistemas especiales: procfs, ramfs y devfs.

El modelo general de ficheros puede ser interpretado como orientado a objetos, donde los objetos son construcciones de software (estructura de datos y funciones y métodos asociados) de los siguientes tipos:

- **Super bloque**: mantiene información relacionada a los sistemas de ficheros montados. Está representado por un bloque de control de sistema almacenado en el disco (para sistemas basados en disco).
- **i-nodo**: mantiene información relacionada a un fichero individual. Cada i-nodo contiene la meta-información del fichero: propietario, grupo, fecha y hora de creación, modificación y último acceso, más un conjunto de punteros a los bloques del disco que almacenan los datos del fichero.
- **Fichero**: mantiene la información relacionada a la interacción de un fichero abierto y un proceso. Este objeto existe sólo cuando un proceso interactúa con el fichero.
- **Dentry**: enlaza una entrada de directorio (*pathname*) con su fichero correspondiente. Los objetos *dentry* recientemente usados son almacenados en una caché (*dentry cache*) para acelerar la translación desde un nombre de fichero al i-nodo correspondiente.

Todos los sistemas Unix modernos permiten que los datos del sistema de ficheros sean accedidos de dos formas distintas:

1. Asociación (*mapping*) de memoria con **mmap**: La llamada de sistema mmap() permite a los procesos de usuario acceder de forma directa a los datos del *page-cache* mediante asociación de memoria. El objetivo de mmap es permitir el acceso a los datos mediante direcciones del sistema de memoria virtual, por lo que los datos de ficheros pueden ser tratados como estructuras de memoria estándares. Los datos del fichero son leídos y copiados a la *page-cache* bajo demanda (*lazily*) cuando se generan fallos de página por intentos de acceso a páginas no residentes en RAM.
2. Llamadas de sistema de acceso directo al sistema de E/S de bloques, tales como **read** y **write**: La llamada de sistema read() lee los datos de los dispositivos de bloques a la caché del núcleo (para CDs y DVDs se puede evitar esta copia mediando el parámetro O_DIRECT del *ioctl*), luego se copian al espacio de direcciones del proceso. La llamada de sistema write() copia los datos en la dirección opuesta, desde la memoria del proceso a la caché del núcleo y eventualmente, en un *futuro próximo*, a los bloques correspondientes del disco. Estas interfaces son implementadas usando el *buffer-cache* o el *page-cach*e para almacenar temporalmente en la memoria del núcleo.

Page-cache y *buffer-cache* del Linux

En versiones anteriores del Linux (y en Unix en general), las operaciones de correspondencia de memoria (*mapping*) eran gestionadas por el sistema de memoria virtual (VM o MM), mientras que las llamadas de E/S por bloques fueron gestionadas independientemente por el subsistema de E/S. Por ejemplo, en Linux hasta la versión 2.2.x, el sistema de memoria virtual y el de E/S tenían cada uno sus propios mecanismos de caché para mejorar el rendimiento: *page-cache* y *buffer-cache* respectivamente.

Buffer-cache

El *buffer-cache* mantiene copias de bloques de disco individuales. Las entradas del caché están identificadas por el dispositivo y número de bloque. Cada *buffer* se refiere a cualquier bloque en el disco y consiste de una cabecera y un área de memoria *igual* al tamaño del bloque del dispositivo. Para minimizar la sobrecarga, los *buffers* se mantienen en una de varias listas enlazadas: sin usar (*unused*), libres (*free*), no modificadas (*clean*), modificadas (*dirty*), bloqueadas (*locked*), etc.

Por cada operación de lectura, el subsistema de *buffer-cache* debe buscar si el bloque en cuestión ya está copiado en la memoria. Para hacerlo de forma eficiente se mantienen tablas de dispersión para todos los *buffers* presentes en la caché. El *buffer-cache* es también usado para mejorar las operaciones de escritura, en vez de copiar todos los bytes inmediatamente al disco, el núcleo los almacena temporalmente en el *buffer-cache* e intenta agrupar varias operaciones para escribirlas al disco simultáneamente. Un *buffer* que está esperando por ser copiado al disco se denomina *modificado* (*sucio* o *dirty*).

Page-cache

A diferencia del anterior, el *page-cache* mantiene páginas completas de la memoria virtual (4 KB en la plataforma x86). Las páginas pertenecen a ficheros en el sistema de ficheros, de hecho las entradas en el *page-cache* están parcialmente indexadas por el número de i-nodo y su desplazamiento en el fichero. Pero una página es casi siempre más grande que un bloque de disco, y los bloques que conforman dicha página pueden estar almacenados de forma discontinua en el disco.

El *page-cache* es principalmente usado para satisfacer los requerimientos de interfaz del subsistema de memoria virtual, que usa páginas de tamaño fijo de 4KB, con el VFS, que usa bloques de tamaño variable u otro tipos de técnicas tales como *extents* en XFS y JFS.

Unificación del *page-cache* y *buffer-cache*

Los dos mecanismos anteriores operaron de forma semi-independiente uno del otro. El sistema operativo tenía que tener un especial cuidado para mantener sincronizados los dos caches y así prevenir que las aplicaciones reciban datos inválidos. No sólo eso, si el sistema estaba escaso de memoria, el sistema operativo tenía que tomar decisiones muy difíciles para liberar memoria del *buffer-cache* o del *page-cache*.

El *page-cache* tiende a ser más sencillo de administrar debido a que representa más directamente los conceptos usados en los niveles superiores del código del núcleo. El *buffer-cache* además tiene las limitaciones que los datos siempre deben ser asociados al espacio de direcciones del núcleo, lo que agrega límites artificiales a la cantidad de datos que pueden mantenerse en caché, ya que el hardware moderno puede tener fácilmente más RAM que el espacio de memoria del núcleo.

Con el tiempo, partes importantes del núcleo han pasado de usar el *buffer-cache* al *page-cache* por razones de simplicidad de codificación. Pero los bloques individuales del *page-cache* estaban todavía gestionados por el *buffer-cache*, lo que creaba confusiones entre el uso y programación de ambos niveles.

Esta falta de integración llevó a un rendimiento ineficiente y falta de flexibilidad. Para lograr un buen rendimiento es importante que los sistemas de memoria virtual y E/S estén bien integrados. La forma elegida en Linux para reducir las ineficiencias de las dobles copias es almacenar los datos sólo una vez (en el *page-cache)* y mantener punteros de cada elemento en el *buffer-cache*.

Las correspondencias temporales de memoria de las páginas en el *page-cache* para soportar read() y write() son raramente necesarias ya que Linux traduce permanentemente toda la memoria física al espacio de direcciones del núcleo. Linux además usa una técnica interesante, el número de bloque donde está una página en el disco está almacenada en forma de una lista en memoria con una estructura denominada *buffer_head*. Cuando una página modificada tiene que ser grabada en disco, las solicitudes de E/S pueden ser pasadas directamente al gestor del dispositivo, sin necesidad de hacer ningún tipo de operación adicional para determinar donde deben ser escritos los datos.

La unificación en el 2.4.10

Siguiendo los conceptos del "Unified I/O and Memory Caching Subsystem for NetBSD"[1], Linus Torvalds quiso cambiar radicalmente el comportamiento del *page-cache* en el núcleo de Linux. El 4 de mayo de 2001, escribió el siguiente mensaje[2] a la lista de desarrolladores (linux-kernel):

> *Quiero re-escribir block_read/write para que use el page cache, pero no porque impactará nada en esta discusión. Quiero hacerlo al principio del 2.5.x porque:*
>
> *- acelerará los accesos*
>
> *- reusará mejor el código existente y conceptualizará las cosas más claramente (i.e. convertiría un disco en sistema de ficheros _realmente_ simple con sólo un fichero enorme ;-)*
>
> *- hará que la gestión de memoria sea mucho mejor para cosas como el fsck - la presión de memoria está diseñada para trabajar en cosas como el page-cache.*
>
> *- será una cosa menos que use el buffer cache como una "caché" (quiero que la gente piense y use el buffer cache como una entidad de _E/S_, no como una caché).*
>
> *No cambiará el "caché al arranque" para nada (porque aún en el page cache, no hay nada en común entre la representación virtual de un _fichero_ (o metadata) y la representación virtual de un _disco_).*

Aunque estos cambios no estaban programados hasta el 2.5.x, Linus finalmente se decidió por integrar una cantidad considerable (más de 150) de parches de Andrea Arcangeli, más sus propios cambios, y liberó el 2.4.10, que finalmente unificaba el *page-cache* y *buffer-cache*. A raíz de estos cambios, se esperan importantes mejoras en el sistema de E/S, sobre todo cuando se ajusten mejor los parámetros del sistema de memoria virtual (al momento de escribir este artículo, la versión disponible del Linux es 2.4.13, que ya muestra importantes mejoras de rendimiento, sobre todo para sistemas con poca memoria, y un consumo bastante menor de RAM para el *page* y *buffer cache*).

Solo queda por decir: actualiza el kernel de tu servidor, hubo muchos cambios y cada vez funciona mejor. Para el 2.4.14 se esperan también soluciones a algunos problemas "límites" en cuanto a uso de memoria.

Ricardo Galli

Lista de enlaces de este artículo:

1. http://www.usenix.org/publications/library/proceedings/usenix2000/freenix/full_p
2. http://lwn.net/2001/0510/a/lt-blkdev-pc.php3

E-mail del autor: gallir _ARROBA_ uib.es