THE PAGE CACHE AND PAGE WRITEBACK

El kernel de Linux implementa una caché de disco llamada caché de páginas. El objetivo de esta caché es minimizar la E/S del disco almacenando datos en la memoria física que de otro modo requerirían acceso al disco.

Dos factores se combinan para hacer de la caché de disco un componente crítico. En primer lugar, el acceso al disco es varios órdenes de magnitud más lento que el acceso a la memoria. En segundo lugar, los datos a los que se accede una vez serán, con gran probabilidad, accedidos de nuevo en un futuro próximo. Este principio se denomina localidad temporal.

- **ENFOQUES DEL CACHEO**

La caché de páginas consiste en páginas físicas en la RAM, cuyo contenido corresponde a bloques físicos en un disco. El tamaño es dinámico; puede crecer para consumir cualquier memoria libre y reducirse para aliviar la presión de la memoria. Cada vez que el kernel comienza una operación de lectura, primero comprueba si los datos necesarios están en la caché de páginas. Si lo está, el kernel puede renunciar a acceder al disco y leer los datos directamente de la RAM. A esto se le llama cache hit. Si los datos no están en la caché, lo que se llama un fallo de caché, el núcleo debe programar operaciones de E/S en bloque para leer los datos del disco.

- **WRITE CACHING**

Las cachés pueden implementar una de tres estrategias diferentes.

En la primera estrategia, denominada no-write, la caché simplemente no almacena en caché las operaciones de escritura.

En la segunda estrategia, una operación de escritura actualiza automáticamente tanto la caché en memoria como el archivo en disco. Este enfoque se llama write-through porque las operaciones de escritura pasan inmediatamente por la caché al disco. Este enfoque tiene la ventaja de mantener la caché coherente sin necesidad de invalidarla.

La tercera estrategia, se llama write-back. En una write-back cache, los procesos realizan operaciones de escritura directamente en la caché de páginas. El almacenamiento de respaldo no se actualiza inmediata o directamente. En su lugar, las páginas escritas en la caché de páginas se marcan como dirty y se añaden a una lista dirty. Periódicamente, las páginas de la lista dirtyse vuelven a escribir en el disco en un proceso llamado writeback, lo que hace que la copia en el disco se alinee con la caché en memoria. Las páginas se marcan entonces como no sucias.

- **CACHE EVICTION**

La última pieza de la caché es el proceso por el cual los datos se eliminan de la caché, ya sea para hacer espacio para las entradas de la caché más relevantes o para reducir la caché para hacer disponible más memoria RAM para otros usos. Este proceso, y la estrategia que decide qué eliminar, se llama cache eviction. Esto funciona seleccionando las páginas limpias y simplemente reemplazándolas con algo más. Si no hay suficientes páginas limpias en la caché, el kernel fuerza un writeback para que haya más páginas limpias disponibles. La parte difícil es decidir qué desalojar. La estrategia de desalojo ideal consiste en desalojar las páginas con menos probabilidades de ser utilizadas en el futuro.

- **Least Recently Used O Menos utilizado recientemente**

La página de caché menos usada recientemente es reemplazada.

- **The Two-List Strategy O Estrategia de dos listas**

Versión modificada de LRU. En lugar de mantener una lista, la lista LRU, Linux mantiene dos listas: la lista activa y la lista inactiva. Las páginas de la lista activa se consideran "hot" y no están disponibles para su desalojo. Las páginas de la lista inactiva están disponibles para cache eviction. Las páginas se colocan en la lista activa sólo cuando se accede a ellas mientras ya residen en la lista inactiva. Ambas listas se mantienen de forma pseudo-LRU: Los elementos se añaden a la cola y se eliminan de la cabeza, como en una cola.

- **CACHE DE PAGINAS DE LINUX**

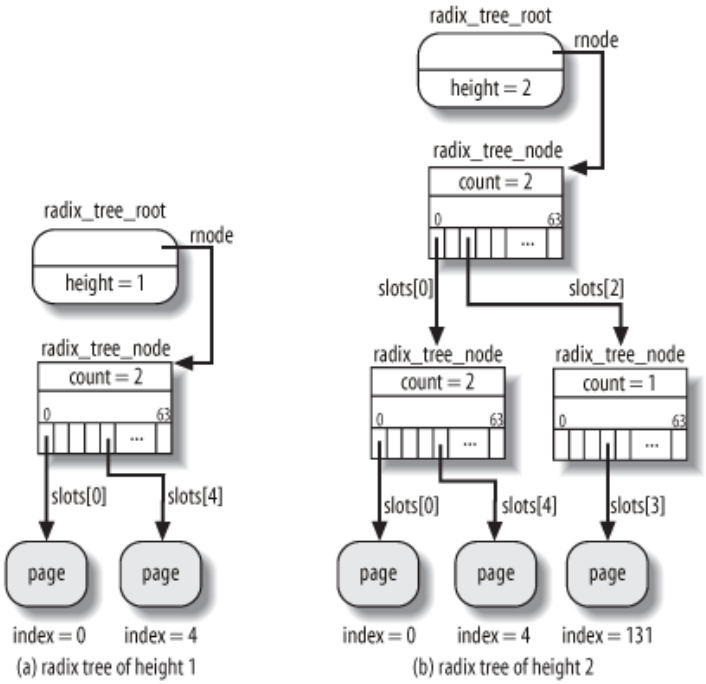
Las páginas se originan a partir de lecturas y escrituras de ficheros regulares del sistema de ficheros, ficheros de dispositivos de bloque y ficheros mapeados en memoria. De este modo, la caché de páginas contiene trozos de ficheros a los que se ha accedido recientemente. Durante una operación de E/S de páginas, el kernel comprueba si los datos residen en la caché de páginas. Si están, el kernel puede devolver rápidamente la página solicitada desde la memoria en lugar de leer los datos del disco.

- **THE address_space OBJECT**

Aunque la caché de páginas de Linux podría funcionar ampliando la estructura de los inodos para soportar las operaciones de E/S de páginas, esta opción limitaría la caché de páginas a los archivos. Para mantener una caché de páginas genérica, la caché de páginas de Linux utiliza un nuevo objeto para gestionar las entradas en la caché y las operaciones de E/S de páginas. Ese objeto es la estructura address_space. Al igual que

el archivo puede tener muchas direcciones virtuales, pero existir sólo una vez en la memoria física.

- **RADIX TREE**



Dado que el kernel debe comprobar la existencia de una página en la caché de páginas antes de iniciar cualquier E/S de páginas, dicha comprobación debe ser rápida. De lo contrario, la sobrecarga de buscar y comprobar la caché de páginas podría anular cualquier beneficio que la caché pudiera proporcionar.

La caché de páginas se busca a través del objeto address_space más un valor de offset. Cada address_space tiene un único radix tree almacenado como page_tree. Un radix tree es un tipo de árbol binario. El árbol radix permite una búsqueda rápida de la página deseada, dado sólo el offset del archivo.

- **ANTIGUA TABLA HASH**

Cuatro problemas:

- Un único bloqueo global protegía el hash. La contención de bloqueos era bastante alta y el rendimiento se veía afectado por ello.

- El hash era más grande de lo necesario porque contenía todas las páginas de la caché de páginas.

- El rendimiento cuando la búsqueda del hash fallaba era más lento de lo deseado, particularmente porque era necesario recorrer las cadenas de un valor hash dado.

- **BUFFER CACHE**

Los búfers actúan como descriptores que asignan páginas en la memoria a bloques de disco; por lo tanto, la caché de páginas también reduce el acceso al disco durante las operaciones de E/S de bloques, ya que almacena en caché los bloques de disco y almacena en búfer las operaciones de E/S de bloques hasta más tarde. Este almacenamiento en caché suele denominarse caché de búfer.

Las operaciones de E/S en bloque manipulan un solo bloque de disco a la vez. Una operación común de E/S en bloque es la lectura y escritura de inodos. A través de los búferes, los bloques de disco se asignan a sus páginas asociadas en memoria y se almacenan en la caché de páginas.

- **FLUSHER THREADS**

Las operaciones de escritura se aplazan en la caché de páginas. Cuando los datos de la caché de páginas son más nuevos que los del almacén de respaldo, los llamamos dirty. Las páginas dirty que se acumulan en la memoria necesitan eventualmente ser escritas de nuevo en el disco. La escritura de páginas sucias se produce en tres situaciones:

- Cuando la memoria libre se reduce por debajo de un umbral específico, el kernel escribe los datos sucios de vuelta al disco para liberar memoria, ya que sólo la memoria limpia está disponible para el desalojo.

- Cuando los datos sucios crecen por encima de un umbral específico, los datos suficientemente viejos se escriben de nuevo en el disco para asegurar que los datos sucios no permanezcan sucios indefinidamente.

- Cuando un proceso de usuario invoca las llamadas al sistema sync() y fsync(), el kernel realiza writeback bajo demanda.

Los flusher threads realizan las tres tareas.

En primer lugar, necesitan devolver los datos sucios al disco cuando la cantidad de memoria libre en el sistema se reduce por debajo de un nivel especificado. El objetivo de esta write-back en segundo plano es recuperar la memoria consumida por las páginas sucias cuando la memoria física disponible es baja. Cuando la memoria libre cae por debajo de este umbral, el kernel despierta uno o más flusher threads para comenzar a escribir en las páginas sucias. La función continúa escribiendo datos hasta que se cumplan dos condiciones:

- Se ha escrito el número mínimo de páginas especificado.

- La cantidad de memoria libre está por encima del umbral.

- **MODO LAPTOP**

Es una estrategia especial de writeback de páginas destinada a optimizar la vida de la batería minimizando la actividad del disco duro y permitiendo que los discos duros permanezcan girados el mayor tiempo posible.

Realiza un único cambio en el comportamiento de escritura de páginas. Además de realizar writeback de las páginas sucias cuando se hacen demasiado viejas, los flusher threads también se hacen cargo de cualquier otra E/S física del disco, descargando todos los búferes sucios al disco.

- **BDFLUSH, KUPDATED Y PDFLUSH**

El hilo del kernel bdflush realiza la escritura en segundo plano de las páginas sucias cuando la memoria disponible es baja. Se mantiene un conjunto de umbrale, y bdflush se despierta cada vez que la memoria libre cae por debajo de esos umbrales.

Debido a que bdflush descarga los búferes sólo cuando la memoria es baja o el número de búferes es demasiado grande, se introdujo el hilo kupdated para reescribir periódicamente las páginas sucias.

Los hilos pdflush se comportan de forma similar a los flusher threads. La principal diferencia es que el número de hilos pdflush es dinámico dependiendo de la carga de E/S del sistema. Los hilos pdflush son globales para todos los discos del sistema.

- **EVITAR LA CONGESTIÓN CON MÚLTIPLES HILOS**

Se resuelve este problema permitiendo la existencia de múltiples hilos de descarga. Cada hilo descarga individualmente las páginas sucias al disco, permitiendo que diferentes flusher threads se concentren en diferentes colas de dispositivos. Con los hilos pdflush, el número de hilos era dinámico, y cada hilo intentaba mantenerse ocupado cogiendo datos de la lista sucia por superbloque y escribiéndolos de nuevo en el disco. El enfoque de pdflush evita que un solo disco ocupado haga pasar hambre a otros discos.