Introducción a los Sistemas Operativos

Cache de Disco

I.S.O.

- Versión: Noviembre de 2025
- Palabras Claves: Unix, Buffer Cache, Cache, Disco, Bloque, Archivos

Las diapositivas han sido extraídas del libro “THE DESIGN OF THE UNIX OPERATING SYSTEM” de Maurice J. Bach

Recordar

- Servicios del Subsistema de E/S
  - Planificación
  - Buffering
  - Caching
  - Spooling
  - ...

Disk Cache

- Buffers en memoria principal para almacenamiento temporario de bloques de disco.

- Objetivo: MINIMIZAR LA FRECUENCIA DE ACCESO AL DISCO

Algunas observaciones

- Cuando un proceso quiere acceder a un bloque de la cache hay dos alternativas:
  - Se copia el bloque al espacio de direcciones del usuario no permitiría compartir el bloque
  - Se trabaja como memoria compartida permite acceso a varios procesos
    - Esta área de memoria debe ser limitada, con lo cual debe existir un algoritmo de reemplazo

Estrategia de reemplazo

- Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado.
- Es una lista de bloques, donde el último es el más recientemente usado (LRU, Least Recently Used)
- Cuando un bloque se referencia o entra en la cache queda al final de la lista
- No se mueven los bloques en la memoria: se asocian punteros.
- Otra alternativa: Least Frecuently Used. Se reemplaza el que tenga menor número de referencias

Introducción a los Sistemas Operativos

Buffer Cache
Unix System V

Objetivo y estructura

- Minimizar la frecuencia de acceso a disco
- Es una estructura formada por buffers en Mem. Principal
- El kernel asigna un espacio en la memoria principal durante la inicialización para esta estructura.
- Un Buffer-Cache tiene dos partes:
  - Headers: Contienen información para modelar: la ubicación del bloque en RAM, número del bloque, estado, relaciones entre headers, etc
  - El buffer en sí: el lugar en RAM (referenciado por el header) donde se almacena realmente el bloque del dispositivo llevado a memoria

Buffer Cache en el Kernel

- El módulo de buffer cache es independiente del sistema de archivos y de los dispositivos de hardware
- Es un servicio del SO

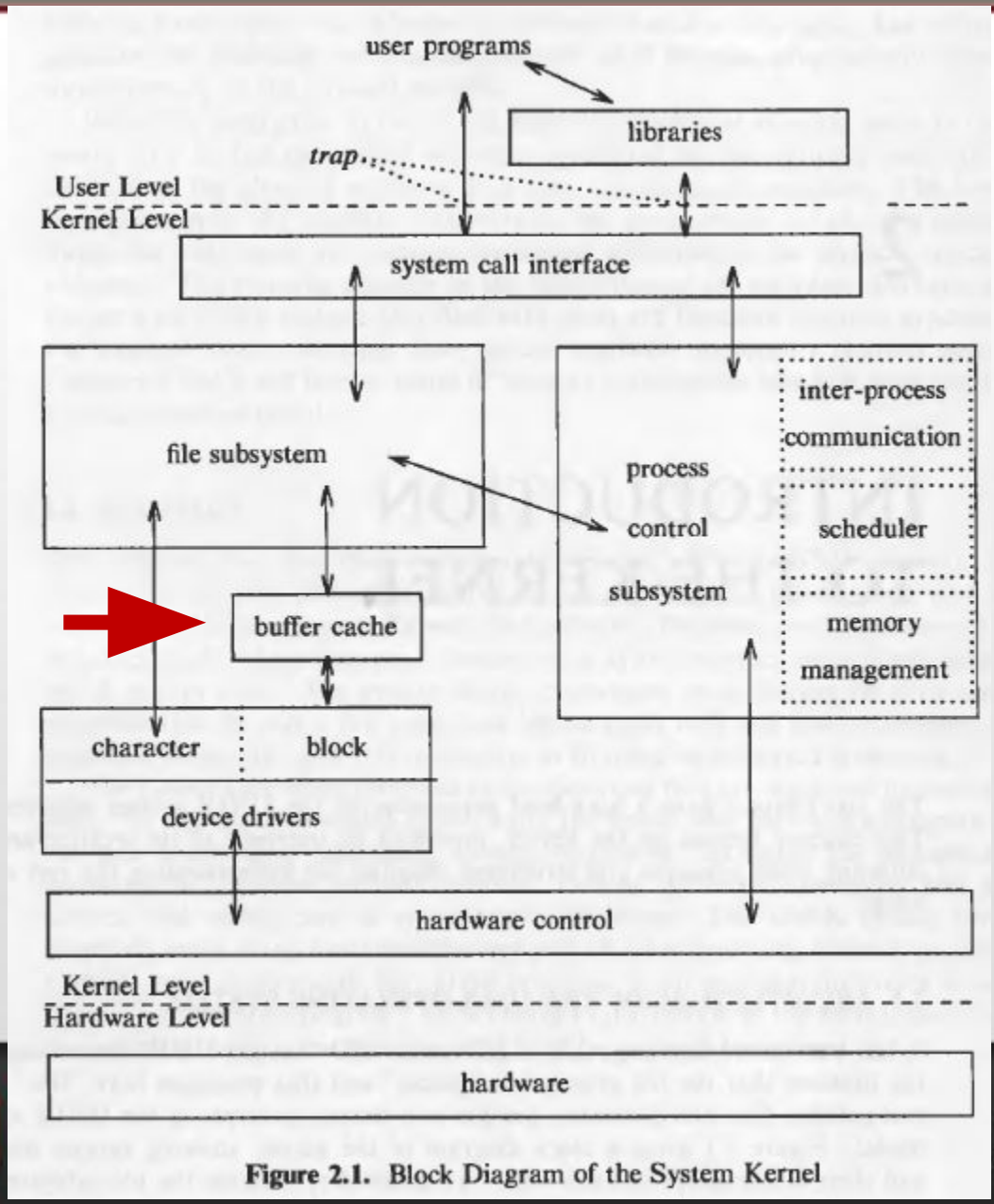


Figure 2.1. Block Diagram of the System Kernel

El header

- Identifica el nro. de dispositivo y nro. de bloque
- Estado
- Punteros a:
  - ✔ **2 punteros para la hash queue** (más adelante vemos para que se usan)
  - ✔ **2 punteros para la free list** (más adelante vemos para que se usan)
  - ✔ **1 puntero al bloque en memoria**

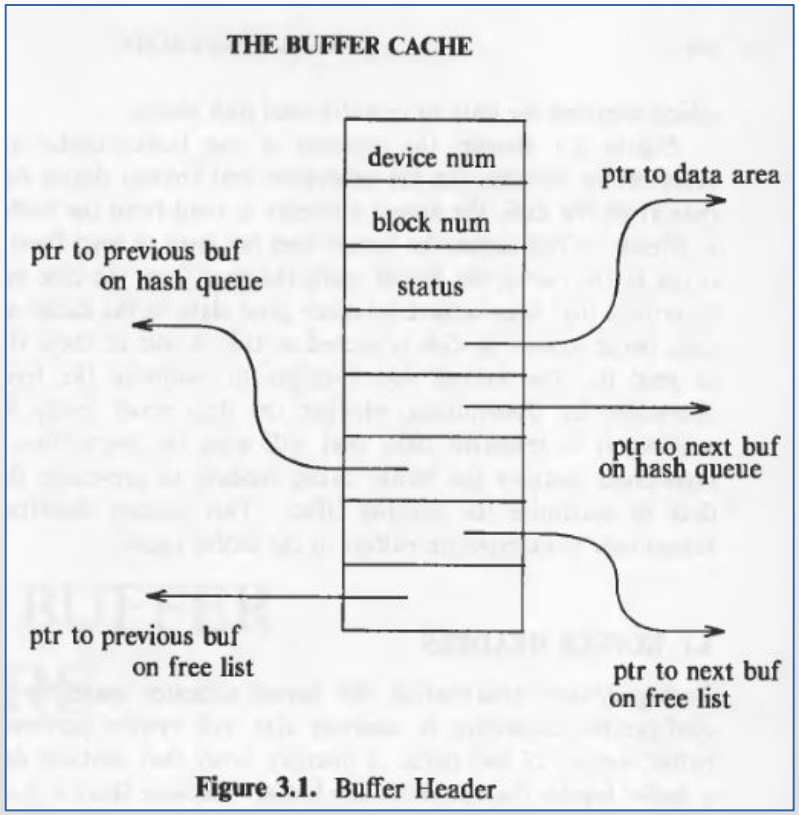


Figure 3.1. Buffer Header

Estados de los buffers

- Free o disponible
- Busy o no disponible (en uso por algún proceso)
- Se está escribiendo o leyendo del disco.
- Delayed Write (DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original en disco.

Free List

- Organiza los buffers disponibles para ser utilizados para cargar nuevos bloque de disco.
- No necesariamente los buffers están vacios (el proceso puede haber terminado, liberado el bloque pero sigue en estado delayed write)
- Se ordena según LRU (least recent used)

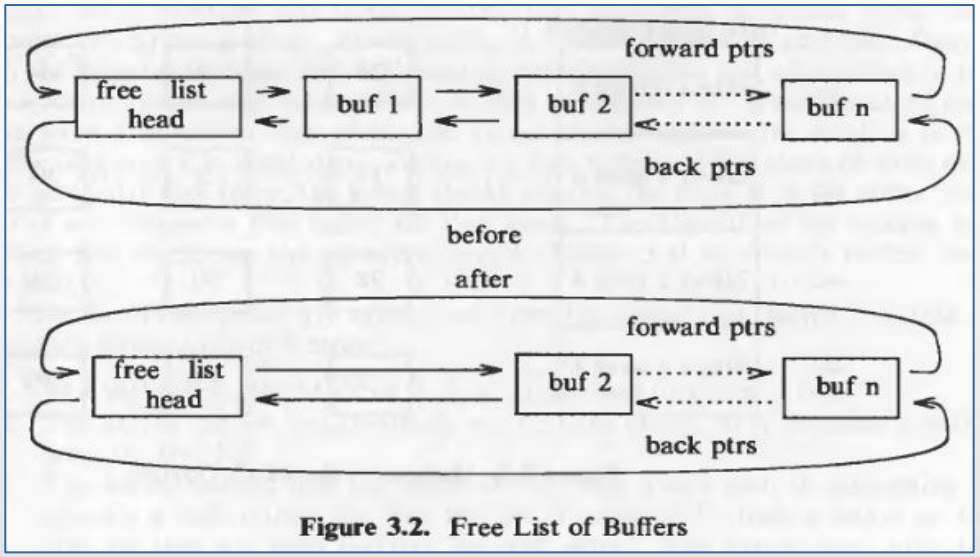


Figure 3.2. Free List of Buffers

Hash Queues

- Son colas para optimizar la búsqueda de un buffer en particular
- Los headers de los buffers se organizan según una función de hash usando (dispositivo,#bloque)
- Al número de bloque (dispositivo/bloque) se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean más eficientes
- Se busca que la función de hash provea alta dispersión para lograr que las colas de bloques no sean tan extensas

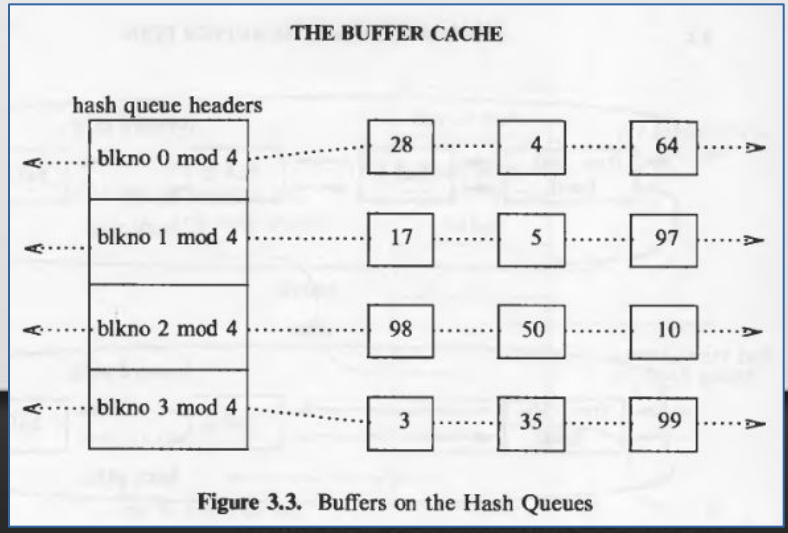


Figure 3.3. Buffers on the Hash Queues

Hash Queues

- Para agrupar los bloques se utilizan los punteros que anteriormente habíamos visto que se almacenaban en el header

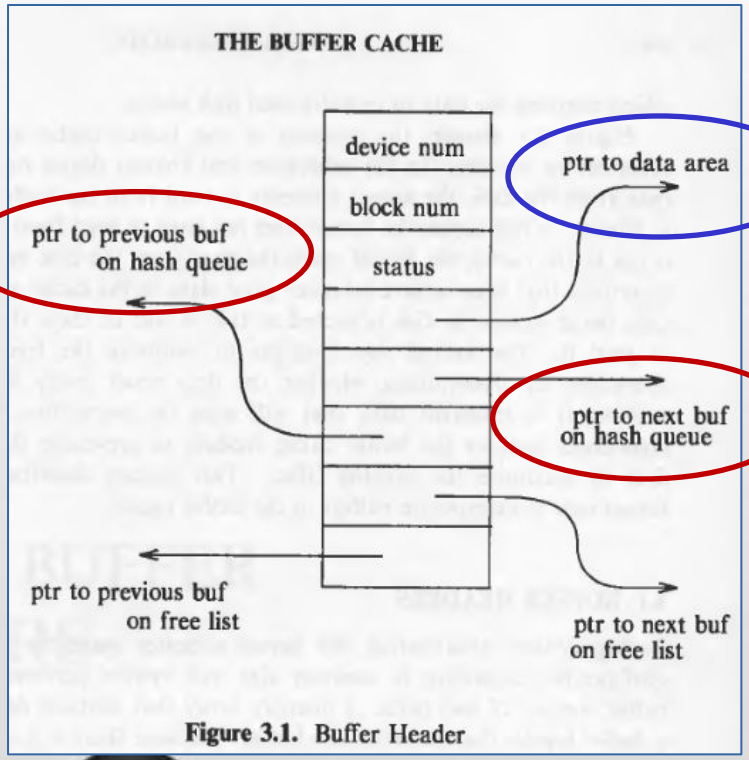


Figure 3.1. Buffer Header

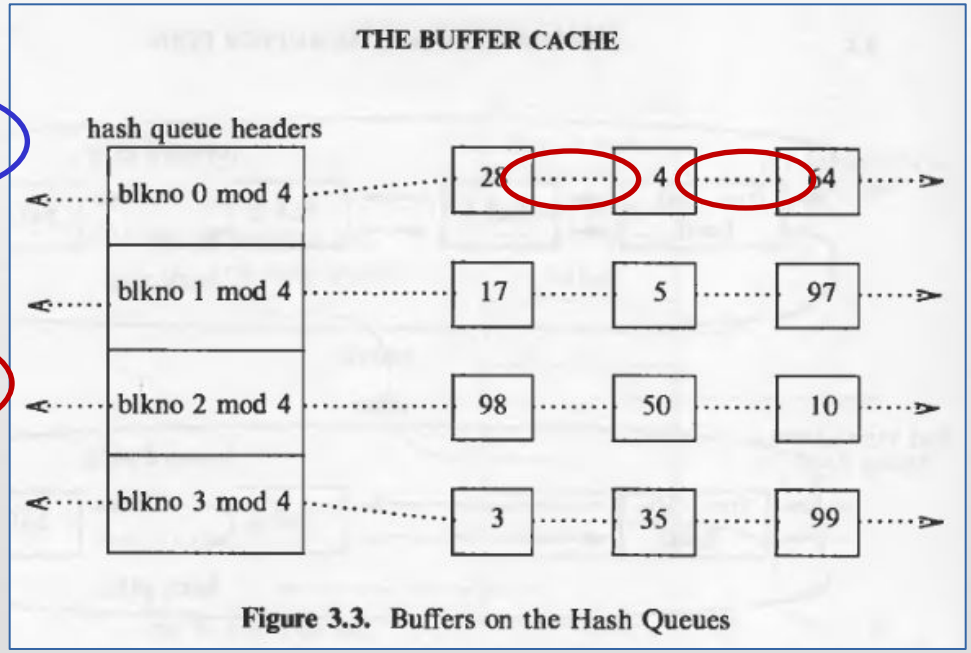


Figure 3.3. Buffers on the Hash Queues

Free List

- Sigue el mismo esquema de la Hash queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado
- El header de un buffer siempre está en la Hash Queue
- Si el proceso que lo referenciaba terminó, va a estar en la Hash Queue y en la Free List

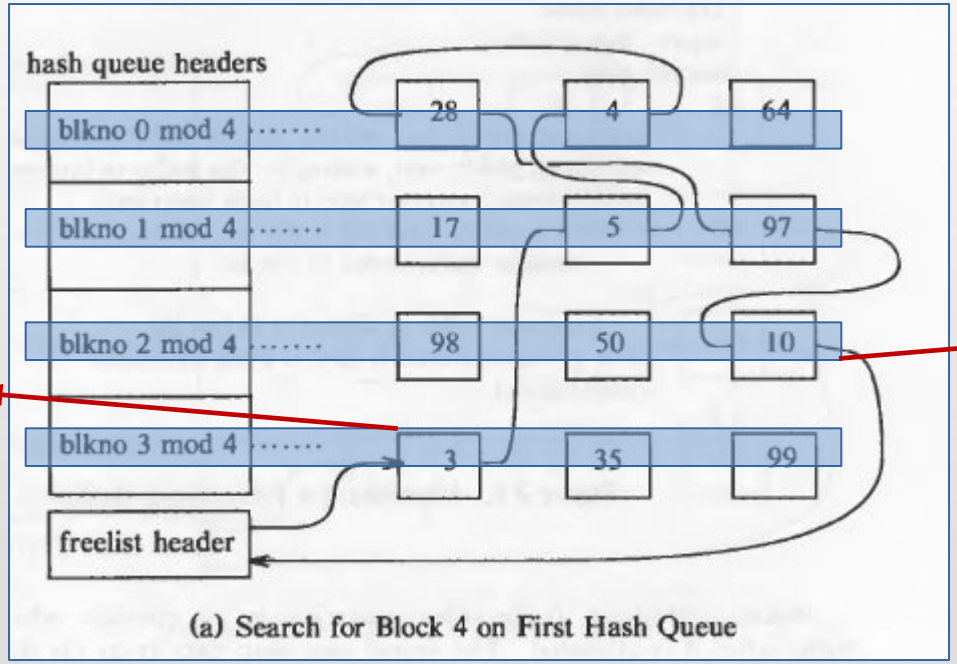
Funcionamiento del buffer cache

- Cuando un proceso quiere acceder a un archivo, se utiliza su i-nodo para localizar los bloques de datos donde se encuentra éste.
- El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S.
- Se pueden dar 5 escenarios:
  1) El kernel encuentra el bloque en la hash queue y el buffer está libre (en la free list).
  2) El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
  3) Idem 2, pero el bloque libre esta marcado como DW.
  4) El kernel no encuentra el bloque en la hash queue y la free list está vacía.
  5) El kernel encuentra el bloque en la hash queue pero está BUSY.

Búsqueda/recuperación de un buffer: 1er escenario

- **Ejemplo: busco el bloque 4:** El kernel encuentra el bloque en la hash queue y el buffer está libre (en la free list).
  - El kernel encuentra el bloque en la hash queue
  - Está disponible (está en la free list)

Header del buffer libre menos recientemente usado

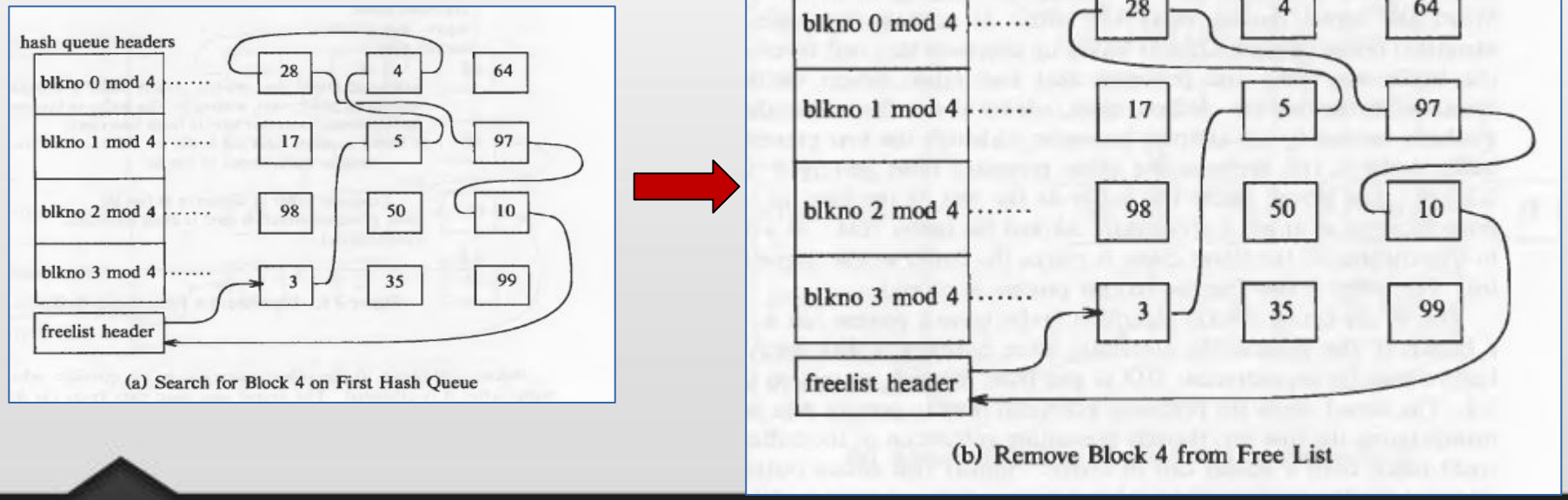


(a) Search for Block 4 on First Hash Queue

Header del buffer libre más recientemente usado

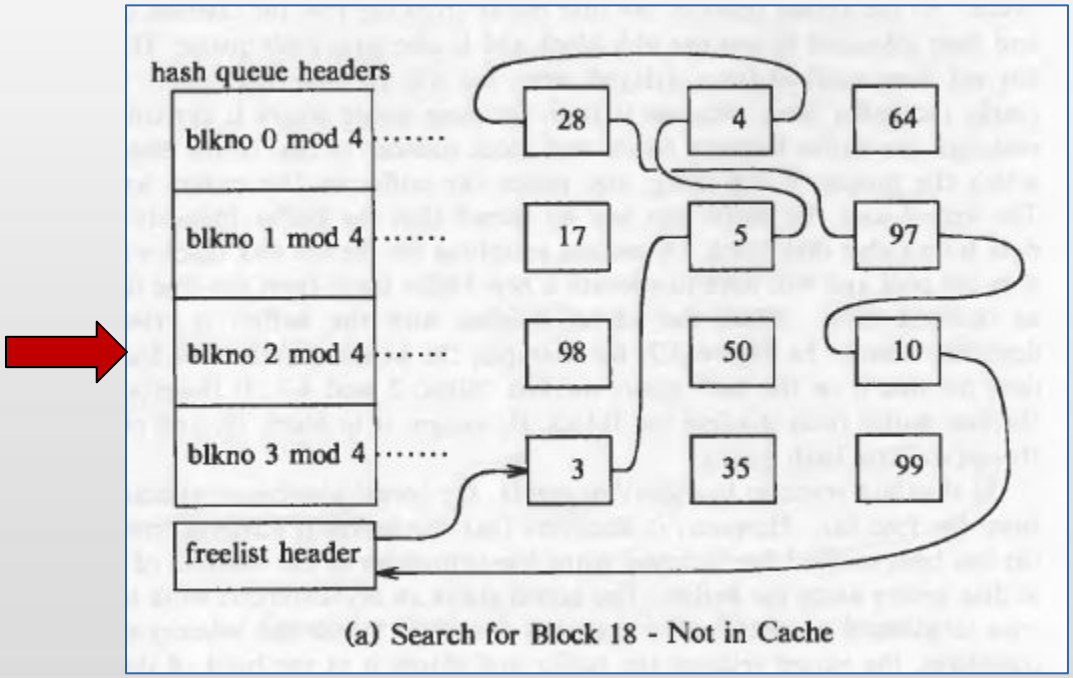
Búsqueda /recuperación de un buffer: 1er escenario (cont.)

- Se remueve el buffer 4 de la free list
- Pasa el buffer 4 a estado BUSY (ocupado)
- El proceso usa el bloque 4
- Se deben reacomodar los punteros de la FreeList



Búsqueda/recuperación de un buffer: 2do escenario

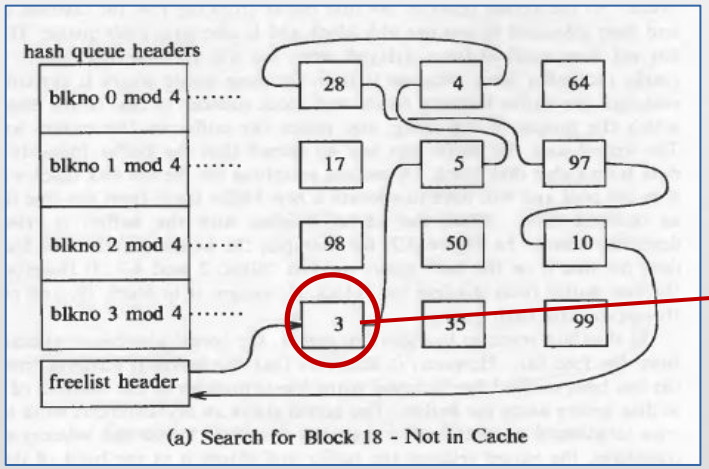
- **Ejemplo: busco el bloque 18:** El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
  - El bloque buscado no está en la hash queue
  - Se debe buscar un bloque libre



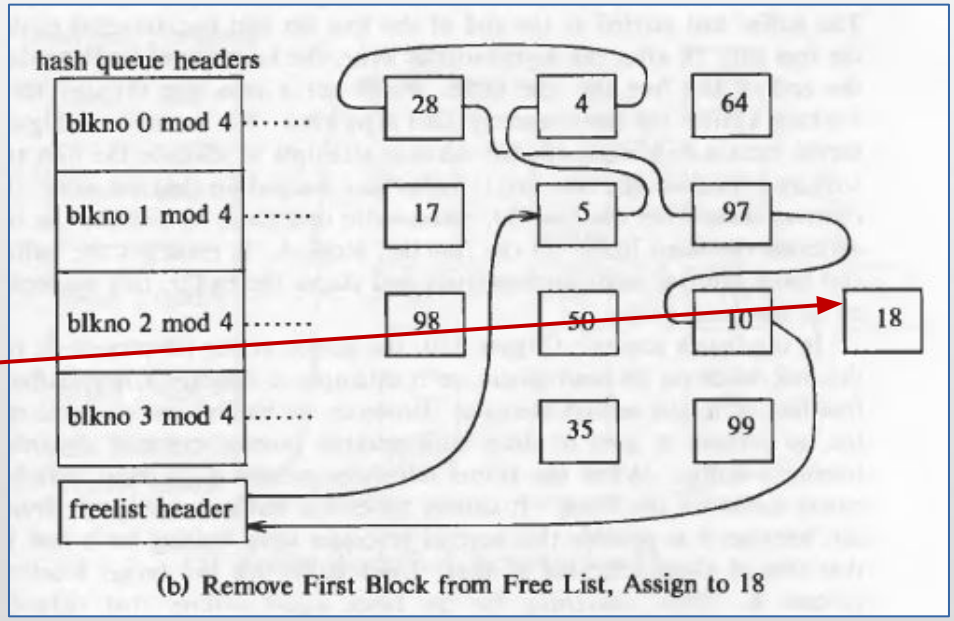
(a) Search for Block 18 - Not in Cache

Búsqueda/recuperación de un buffer: 2do escenario (cont.)

- Se toma un buffer de la free list (el 3)
- Siempre se usa el primero
- Se lee del disco el bloque deseado en el buffer obtenido
- Se ubica el header en la hash queue correspondiente (solo se cambian punteros, **NO se intercambian ubicaciones de memoria)**



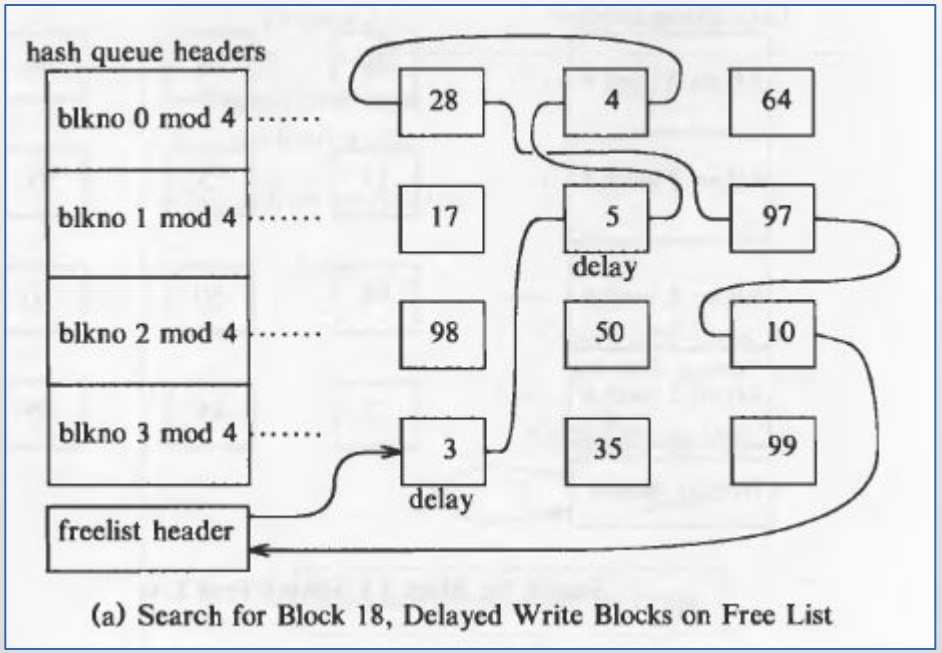
(a) Search for Block 18 - Not in Cache



(b) Remove First Block from Free List, Assign to 18

Búsqueda/recuperación de un buffer: 3er escenario

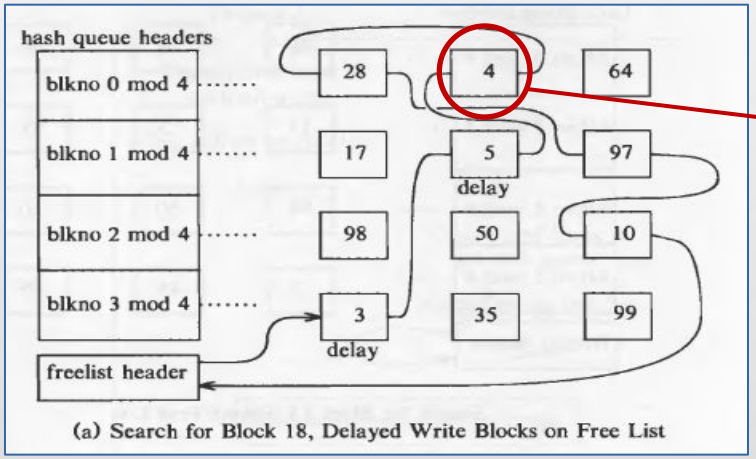
- **Ejemplo: busca el bloque 18:** El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre, marcado domo Delayed Write
  - El Kernel no encuentra el bloque buscado en la hash queue
  - Debe tomar el 1ro de la free list, pero está marcado DW
  - El kernel debe mandar a escribir a disco al bloque 3 y tomar el siguiente buffer de la free list



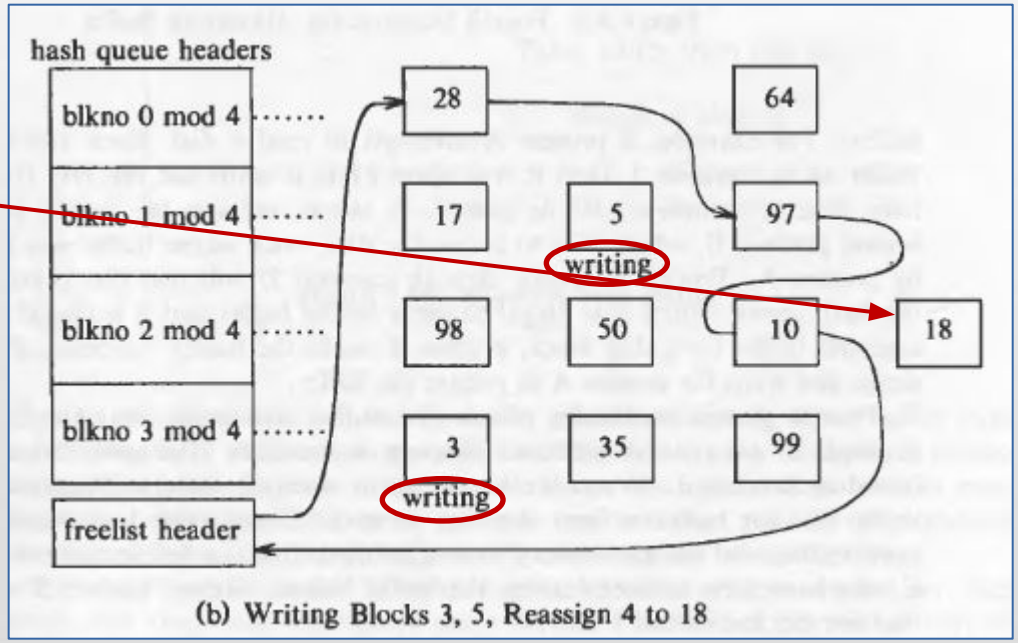
(a) Search for Block 18, Delayed Write Blocks on Free List

Búsqueda/recuperación de un buffer: 3er escenario

- Si también está DW, sigue con el mismo proceso hasta encontrar uno que no esté marcado como DW.
- Mientras los DW se escriben en disco, se asigna el siguiente buffer free al proceso
- Una vez escritos a disco los bloques DW, estos son ubicados al principio de la FreeList



(a) Search for Block 18, Delayed Write Blocks on Free List

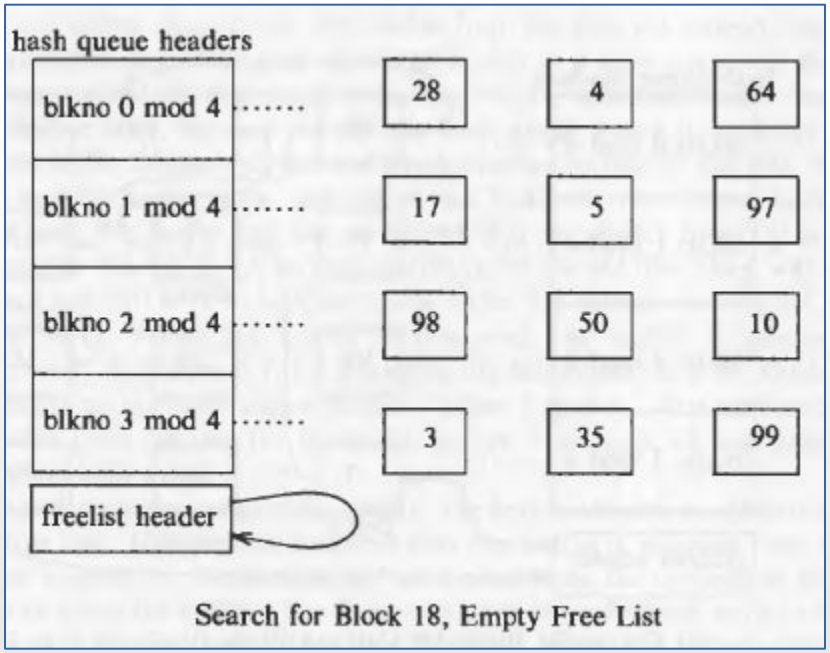


(b) Writing Blocks 3, 5, Reassign 4 to 18

Búsqueda/recuperación de un buffer: 4to escenario

- El kernel no encuentra el bloque en la hash queue y la free list está vacía.
- El proceso queda bloqueado en espera a que se "libere" algún buffer
- Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue (algún proceso pudo haberlo pedido mientras éste dormía)

Búsqueda/recuperación de un buffer: 4to escenario



Search for Block 18, Empty Free List

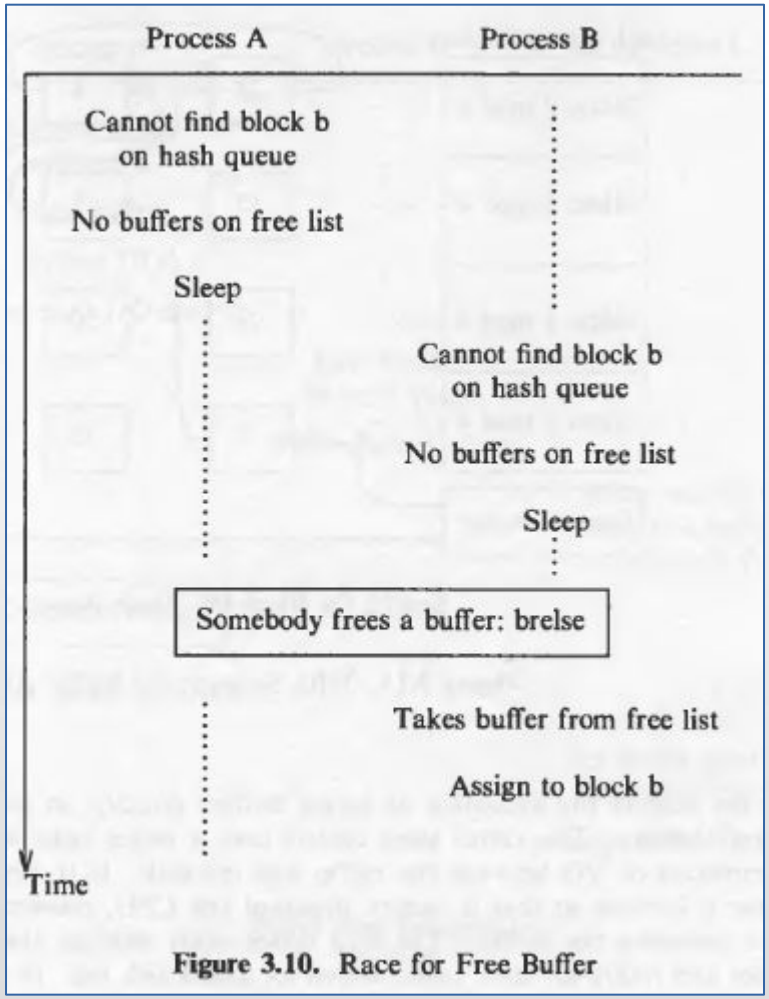
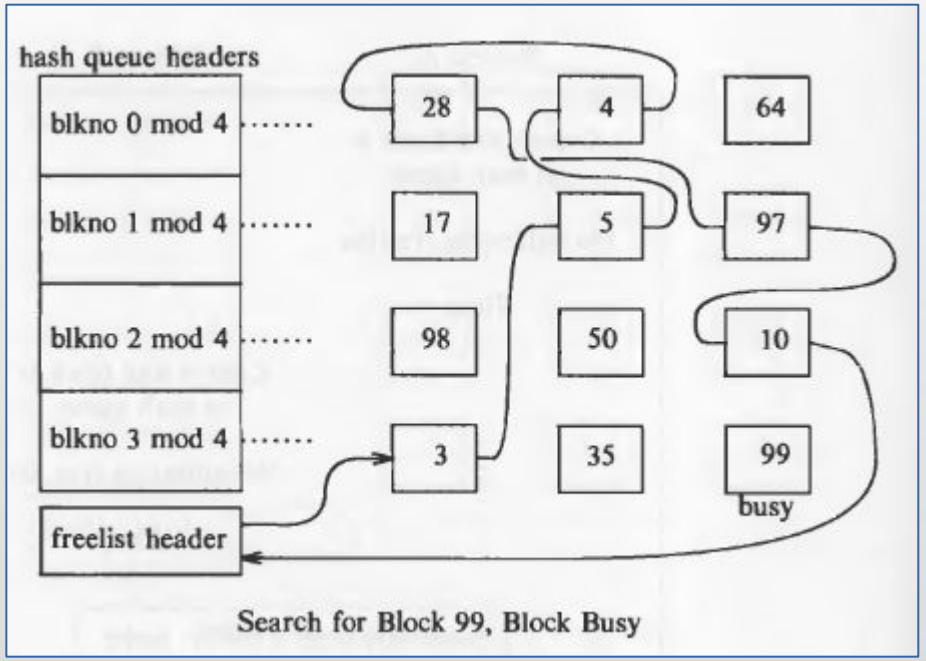


Figure 3.10. Race for Free Buffer

Búsqueda/recuperación de un buffer: 5to escenario

- **Ejemplo: busca el bloque 99:** El kernel encuentra el bloque en la hash queue pero está BUSY.
  - El kernel busca un bloque y el buffer que lo contiene está marcado como busy
  - El proceso se bloquea a la espera de que el buffer se desbloquee



Search for Block 99, Block Busy

Búsqueda/recuperación de un buffer: 5to escenario

- Eventualmente el proceso que tenia el buffer 99 lo libera
  - Se despiertan todos los procesos en espera de algún buffer
  - El proceso que buscaba el buffer 99 debe buscarlo nuevamente en la hashqueue y en la freelist

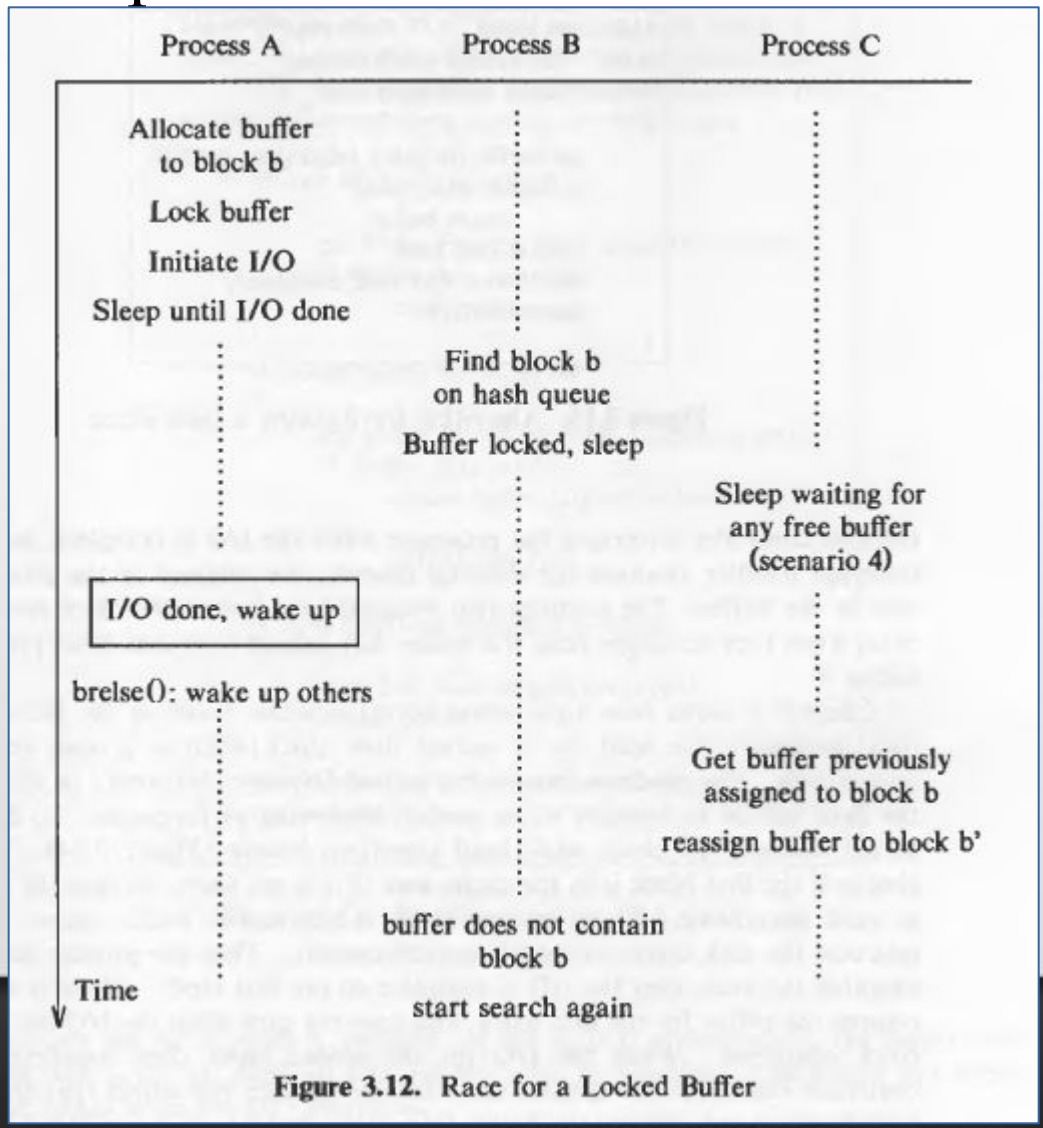


Figure 3.12. Race for a Locked Buffer

Algoritmo de asignación

- Escenarios:
  1) El kernel encuentra el bloque en la hash queue y el buffer está libre.
  2) El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
  3) Idem 2, pero el bloque libre esta marcado como DW.
  4) El kernel no encuentra el bloque en la hash queue y la free list está vacía.
  5) El kernel encuentra el bloque en la hash queue pero está BUSY.

Algoritmo de asignación

```
algorithm getblk
input:   file system number
         block number
output: locked buffer that can now be used for block
{
      while (buffer not found)
      {
            if (block in hash queue)
            {
                  if (buffer busy)       /* scenario 5 */
                  {
                        sleep (event buffer becomes free);
                        continue;       /* back to while loop */
                  }
                  mark buffer busy;       /* scenario 1 */
                  remove buffer from free list;
                  return buffer;
            }
            else      /* block not on hash queue */
            {
                  if (there are no buffers on free list)       /* scenario 4 */
                  {
                        sleep (event any buffer becomes free);
                        continue;       /* back to while loop */
                  }
                  remove buffer from free list;
                  if (buffer marked for delayed write) {      /* scenario 3 */
                        asynchronous write buffer to disk;
                        continue;       /* back to while loop */
                  }
                  /* scenario 2 -- found a free buffer */
                  remove buffer from old hash queue;
                  put buffer onto new hash queue;
                  return buffer;
            }
      }
}
```

Figure 3.4. Algorithm for Buffer Allocation