NAME

buf - kernel buffer I/O scheme used in FreeBSD VM system

DESCRIPTION

The kernel implements a KVM abstraction of the buffer cache which allows it to map potentially disparate vm_page's into contiguous KVM for use by (mainly file system) devices and device I/O. This abstraction supports block sizes from DEV_BSIZE (usually 512) to upwards of several pages or more. It also supports a relatively primitive byte-granular valid range and dirty range currently hardcoded for use by NFS. The code implementing the VM Buffer abstraction is mostly concentrated in */usr/src/sys/kern/vfs_bio.c*.

One of the most important things to remember when dealing with buffer pointers (struct buf) is that the underlying pages are mapped directly from the buffer cache. No data copying occurs in the scheme proper, though some file systems such as UFS do have to copy a little when dealing with file fragments. The second most important thing to remember is that due to the underlying page mapping, the b_data base pointer in a buf is always *page* aligned, not *block* aligned. When you have a VM buffer representing some b_offset and b_size, the actual start of the buffer is (b_data + (b_offset & PAGE_MASK)) and not just b_data. Finally, the VM system's core buffer cache supports valid and dirty bits (m->valid, m->dirty) for pages in DEV_BSIZE chunks. Thus a platform with a hardware page size of 4096 bytes has 8 valid and 8 dirty bits. These bits are generally set and cleared in groups based on the device block size of the device backing the page. Complete page's worth are often referred to using the VM_PAGE_BITS_ALL bitmask (i.e., 0xFF if the hardware page size is 4096).

VM buffers also keep track of a byte-granular dirty range and valid range. This feature is normally only used by the NFS subsystem. I am not sure why it is used at all, actually, since we have DEV_BSIZE valid/dirty granularity within the VM buffer. If a buffer dirty operation creates a 'hole', the dirty range will extend to cover the hole. If a buffer validation operation creates a 'hole' the byte-granular valid range is left alone and will not take into account the new extension. Thus the whole byte-granular abstraction is considered a bad hack and it would be nice if we could get rid of it completely.

A VM buffer is capable of mapping the underlying VM cache pages into KVM in order to allow the kernel to directly manipulate the data associated with the (vnode,b_offset,b_size). The kernel typically unmaps VM buffers the moment they are no longer needed but often keeps the 'struct buf' structure instantiated and even bp->b_pages array instantiated despite having unmapped them from KVM. If a page making up a VM buffer is about to undergo I/O, the system typically unmaps it from KVM and replaces the page in the b_pages[] array with a place-marker called bogus_page. The place-marker forces any kernel subsystems referencing the associated struct buf to re-lookup the associated page. I believe the place-marker hack is used to allow sophisticated devices such as file system devices to remap underlying pages in order to deal with, for example, re-mapping a file fragment into a file block.

VM buffers are used to track I/O operations within the kernel. Unfortunately, the I/O implementation is also somewhat of a hack because the kernel wants to clear the dirty bit on the underlying pages the moment it queues the I/O to the VFS device, not when the physical I/O is actually initiated. This can create confusion within file system devices that use delayed-writes because you wind up with pages marked clean that are actually still dirty. If not treated carefully, these pages could be thrown away! Indeed, a number of serious bugs related to this hack were not fixed until the 2.2.8/3.0 release. The kernel uses an instantiated VM buffer (i.e., struct buf) to place-mark pages in this special state. The buffer is typically flagged B_DELWRI. When a device no longer needs a buffer it typically flags it as B_RELBUF. Due to the underlying pages being marked clean, the B_DELWRI|B_RELBUF combination must be interpreted to mean that the buffer is still actually dirty and must be written to its backing store before it can actually be released. In the case where B_DELWRI is not set, the underlying dirty pages are still properly marked as dirty and the buffer can be completely freed without losing that clean/dirty state information. (XXX do we have to check other flags in regards to this situation ???)

The kernel reserves a portion of its KVM space to hold VM Buffer's data maps. Even though this is virtual space (since the buffers are mapped from the buffer cache), we cannot make it arbitrarily large because instantiated VM Buffers (struct buf's) prevent their underlying pages in the buffer cache from being freed. This can complicate the life of the paging system.

HISTORY

The **buf** manual page was originally written by Matthew Dillon and first appeared in FreeBSD 3.1, December 1998.