Subject: Re: [PATCH] incorrect error handling inside generic_file_direct_write Posted by Dmitriy Monakhov on Tue, 12 Dec 2006 20:14:31 GMT

View Forum Message <> Reply to Message

```
Andrew Morton <akpm@osdl.org> writes:
```

```
> On Tue, 12 Dec 2006 16:18:32 +0300
> Dmitriy Monakhov <dmonakhov@sw.ru> wrote:
>> >> but according to filemaps locking rules: mm/filemap.c:77
>> >> ..
>> > * ->i mutex (generic file buffered write)
>> > * ->mmap_sem (fault_in_pages_readable->do_page_fault)
>> >> ...
>> >> I'm confused a litle bit, where is the truth?
>> >
>> > xfs write() calls generic file direct write() without taking i mutex for
>> > O DIRECT writes.
>> Yes, but my quastion is about generic file aio write nolock().
>> As i understand nolock sufix means that i mutex was already locked
>> by caller, am i right?
> Nope. It just means that __generic_file_aio_write_nolock() doesn't take
> the lock. We don't assume or require that the caller took it. For example
> the raw driver calls generic_file_aio_write_nolock() without taking
> i mutex. Raw isn't relevant to the problem (although ocfs2 might be). But
> we cannot assume that all callers have taken i_mutex, I think.
> I guess we can make that a rule (document it, add
> BUG_ON(!mutex_is_locked(..)) if it isn't a blockdev) if needs be. After
> really checking that this matches reality for all callers.
I've checked generic_file_aio_write_nolock() callers for non blockdev.
Only ocfs2 call it explicitly, and do it under i_mutex.
So we need to do:
1) Change wrong comments.
2) Add BUG ON(!mutex is locked(..)) for non blkdev.
3) Invoke vmtruncate only for non blkdev.
Signed-off-by: Dmitriy Monakhov <dmonakhov@openvz.org>
diff --git a/mm/filemap.c b/mm/filemap.c
index 7b84dc8..540ef5e 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@ @ -2046,8 +2046,8 @ @ generic_file_direct_write(struct kiocb *
 /*
 * Sync the fs metadata but not the minor inode changes and
```

```
- * i mutex is held, which protects generic osync inode() from
- * livelocking.
+ * i_mutex may not being held, if so some specific locking
+ * ordering must protect generic_osync_inode() from livelocking.
 if (written >= 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
 int err = generic_osync_inode(inode, mapping, OSYNC_METADATA);
@ @ -2282,6 +2282,17 @ @ generic file aio write nolock(struct k
 written = generic_file_direct_write(iocb, iov, &nr_segs, pos,
     ppos, count, ocount);
  * If host is not S_ISBLK generic_file_direct_write() may
  * have instantiated a few blocks outside i_size files
 * Trim these off again.
+ if (unlikely(written < 0) && !S ISBLK(inode->i mode)) {
+ loff t isize = i size read(inode);
+ if (pos + count > isize)
+ vmtruncate(inode, isize);
+ }
 if (written < 0 || written == count)
  goto out;
@ @ -2344,6 +2355,13 @ @ ssize_t generic_file_aio_write_nolock(st
 ssize t ret;
 BUG ON(iocb->ki pos!= pos);
+ * generic file buffered write() may be called inside
+ * __generic_file_aio_write_nolock() even in case of
+ * O_DIRECT for non S_ISBLK files. So i_mutex must be held.
+ */
+ if (!S ISBLK(inode->i mode))
+ BUG_ON(!mutex_is_locked(&inode->i_mutex));
 ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
  &iocb->ki pos);
@ @ -2386,8 +2404,8 @ @ ssize t generic file aio write(struct ki
EXPORT_SYMBOL(generic_file_aio_write);
/*
- * Called under i_mutex for writes to S_ISREG files. Returns -EIO if something
- * went wrong during pagecache shootdown.
+ * May be called without i mutex for writes to S ISREG files.
+ * Returns -EIO if something went wrong during pagecache shootdown.
```

* of course not the data as we did direct DMA for the IO.

```
*/
static ssize_t
generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
```