Subject: [PATCH] incorrect direct io error handling
Posted by Dmitriy Monakhov on Mon, 18 Dec 2006 13:22:44 GMT
View Forum Message <> Reply to Message

This patch is result of discussion started week ago here:
http://lkml.org/lkml/2006/12/11/66
changes from original patch:
 - Update wrong comments about i_mutex locking.
 - Add BUG_ON(!mutex_is_locked(..)) for non blkdev.
 - vmtruncate call only for non blockdev
LOG:
If generic_file_direct_write() has fail (ENOSPC condition) inside
__generic_file_aio_write_nolock() it may have instantiated
a few blocks outside i_size. And fsck will complain about wrong i_size
(ext2, ext3 and reiserfs interpret i_size and biggest block difference as error),
after fsck will fix error i_size will be increased to the biggest block,
but this blocks contain gurbage from previous write attempt, this is not
information leak, but its silence file data corruption. This issue affect
fs regardless the values of blocksize or pagesize.
We need truncate any block beyond i_size after write have failed , do in simular
generic_file_buffered_write() error path. If host is !S_ISBLK i_mutex always
held inside generic_file_aio_write_nolock() and we may safely call vmtruncate().
Some fs (XFS at least) may directly call generic_file_direct_write()with
i_mutex not held. There is no general scenario in this case. This fs have to
handle generic_file_direct_write() error by its own specific way (place).

Issue was found during OpenVZ kernel testing.

Exampe:
open("mnt2/FILE3", O_WRONLY|O_CREAT|O_DIRECT, 0666) = 3
write(3, "aaaaaa"..., 4096) = -1 ENOSPC (No space left on device)

stat mnt2/FILE3
File: `mnt2/FILE3'
Size: 0 Blocks: 4 IO Block: 4096 regular empty file
>>>>>>>>>>>>>>>>>>>>>>^^^^^^^^^^ file size is less than biggest block idx
Device: 700h/1792d Inode: 14 Links: 1
Access: (0644/-rw-r--r--) Uid: (0/ root) Gid: (0/ root)

fsck.ext2 -f -n mnt1/fs_img
Pass 1: Checking inodes, blocks, and sizes
Inode 14, i_size is 0, should be 2048. Fix? no

Signed-off-by: Dmitriy Monakhov <dmonakhov@openvz.org>
---

diff --git a/mm/filemap.c b/mm/filemap.c
index 8332c77..7c571dd 100644

```
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -2044,8 +2044,9 @@ generic_file_direct_write(struct kiocb *
  /*
   * Sync the fs metadata but not the minor inode changes and
   * of course not the data as we did direct DMA for the IO.
-  * i_mutex is held, which protects generic_osync_inode() from
-  * livelocking.  AIO O_DIRECT ops attempt to sync metadata here.
+  * i_mutex may not being held (XFS does this), if so some specific locking
+  * ordering must protect generic_osync_inode() from livelocking.
+  * AIO O_DIRECT ops attempt to sync metadata here.
   */
  if ((written >= 0 || written == -EIOCBQUEUED) &&
      ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2279,6 +2280,17 @@ __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;
   /*
@@ -2341,6 +2353,13 @@ 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);
@@ -2383,8 +2402,8 @@ 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. XFS does this.
+ * Returns -EIO if something went wrong during pagecache shootdown.
  */
 static ssize_t
 generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
```