Subject: [PATCH]mm incorrect direct io error handling (v4)
Posted by Dmitriy Monakhov on Thu, 15 Feb 2007 10:47:32 GMT
View Forum Message <> Reply to Message

Incorrect direct io error handling (v4)
 Changes from v3:
 - Patch preparead gainst 2.6.20-rc6-mm3. It seems now it's not conflict
   with Nick's stuff.
 - fix generic segment check function.

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. Initialy i've proposed do it in
__generic_file_aio_write_nolock() with explicit guarantee i_mutex always held,
but not everybody was agree with it. So we may safely call vmtruncate() inside
generic_file_aio_write(), here i_mutex already locked.

TEST_CASE:
open("/mnt/test/BIG_FILE", O_WRONLY|O_CREAT|O_DIRECT, 0666) = 3
write(3, "aaaaaaaaaaaaaaa"..., 104857600) = -1 ENOSPC (No space left on device)

#stat /mnt/test/BIG_FILE
  File: `/mnt/test/BIG_FILE'
  Size: 0   Blocks: 110896   IO Block: 1024   regular empty file
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<^^^^^^^^file size is less than biggest block idx

Device: fe07h/65031d   Inode: 14   Links: 1
Access: (0644/-rw-r--r--) Uid: (0/   root)   Gid: (0/   root)
Access: 2007-01-24 20:03:38.000000000 +0300
Modify: 2007-01-24 20:03:38.000000000 +0300
Change: 2007-01-24 20:03:39.000000000 +0300

#fsck.ext3 -f /dev/VG/test
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Inode 14, i_size is 0, should be 56556544. Fix<y>? yes
Pass 2: Checking directory structure

Changes not dirrectly connected with issue:
 - move common segment checks to separate helper function.

Patch pass ltp and fsstres test.

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

```
diff --git a/mm/filemap.c b/mm/filemap.c
index 529eb9e..5f189e0 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -1170,6 +1170,31 @@ success:
  return size;
 }

+int generic_segment_checks(const struct iovec *iov, unsigned long *nr_segs,
+  size_t *count, unsigned long access_flags)
+{
+ unsigned long   seg;
+ for (seg = 0; seg < *nr_segs; seg++) {
+  const struct iovec *iv = &iov[seg];
+
+  /*
+   * If any segment has a negative length, or the cumulative
+   * length ever wraps negative then return -EINVAL.
+   */
+  *count += iv->iov_len;
+  if (unlikely((ssize_t)(*count|iv->iov_len) < 0))
+   return -EINVAL;
+  if (access_ok(access_flags, iv->iov_base, iv->iov_len))
+   continue;
+  if (seg == 0)
+   return -EFAULT;
+  *nr_segs = seg;
+  *count -= iv->iov_len; /* This segment is no good */
+  break;
+ }
+ return 0;
+}
+
 /**
  * generic_file_aio_read - generic filesystem read routine
  * @iocb: kernel I/O control block
@@ -1191,24 +1216,9 @@ generic_file_aio_read(struct kiocb *iocb, const struct iovec *iov,
  loff_t *ppos = &iocb->ki_pos;

  count = 0;
- for (seg = 0; seg < nr_segs; seg++) {
-  const struct iovec *iv = &iov[seg];
-
```

```
-  /*
-   * If any segment has a negative length, or the cumulative
-   * length ever wraps negative then return -EINVAL.
-   */
-  count += iv->iov_len;
-  if (unlikely((ssize_t)(count|iv->iov_len) < 0))
-   return -EINVAL;
-  if (access_ok(VERIFY_WRITE, iv->iov_base, iv->iov_len))
-   continue;
-  if (seg == 0)
-   return -EFAULT;
-  nr_segs = seg;
-  count -= iv->iov_len; /* This segment is no good */
-  break;
- }
+ retval = generic_segment_checks(iov, &nr_segs, &count, VERIFY_WRITE);
+ if (retval)
+  return retval;

  /* coalesce the iovecs and go direct-to-BIO for O_DIRECT */
  if (filp->f_flags & O_DIRECT) {
@@ -2103,8 +2113,9 @@ generic_file_direct_write(struct kiocb *iocb, const struct iovec *iov,
  /*
   * 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, 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))) {
@@ -2293,30 +2304,14 @@ __generic_file_aio_write_nolock(struct kiocb *iocb, const struct
iovec *iov,
  size_t ocount;  /* original count */
  size_t count;  /* after file limit checks */
  struct inode  *inode = mapping->host;
- unsigned long seg;
  loff_t  pos;
  ssize_t  written;
  ssize_t  err;

  ocount = 0;
- for (seg = 0; seg < nr_segs; seg++) {
-  const struct iovec *iv = &iov[seg];
-
-  /*
```

```
-   * If any segment has a negative length, or the cumulative
-   * length ever wraps negative then return -EINVAL.
-   */
-  ocount += iv->iov_len;
-  if (unlikely((ssize_t)(ocount|iv->iov_len) < 0))
-   return -EINVAL;
-  if (access_ok(VERIFY_READ, iv->iov_base, iv->iov_len))
-   continue;
-  if (seg == 0)
-   return -EFAULT;
-  nr_segs = seg;
-  ocount -= iv->iov_len; /* This segment is no good */
-  break;
- }
+ err = generic_segment_checks(iov, &nr_segs, &ocount, VERIFY_READ);
+ if (err)
+  return err;
  if (unlikely(aio_restarted())) {
   /* nothing to transfer, may just need to sync data */
   return ocount;
@@ -2442,6 +2437,20 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
  mutex_lock(&inode->i_mutex);
  ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
    &iocb->ki_pos);
+ if (unlikely( ret < 0 && file->f_flags & O_DIRECT)) {
+  unsigned long nr_segs_avail = nr_segs;
+  size_t count = 0;
+  if (!generic_segment_checks(iov, &nr_segs_avail, &count,
+    VERIFY_READ)) {
+   loff_t isize = i_size_read(inode);
+   /*
+    * direct_IO() may have instantiated a few blocks
+    * outside i_size.  Trim these off again.
+    */
+   if (pos + count > isize)
+    vmtruncate(inode, isize);
+  }
+ }
  mutex_unlock(&inode->i_mutex);

  if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2458,8 +2467,8 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
 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.
  */
 static ssize_t
 generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
```