Subject: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Dmitriy Monakhov on Mon, 12 Mar 2007 07:57:50 GMT
View Forum Message <> Reply to Message

I realy don't want to be annoying by sending this patcheset over and over
again, i just want the issue to be solved. If anyone think this solution
is realy cappy, please comment what exectly is bad. Thank you.

Changes:
  - patch was split in two patches.
  - comments added. I think now it is clearly describe things.
  - patch prepared against 2.6.20-mm3

How this patch tested:
  - fsstress test.
  - manual direct_io tests.

LOG:
 - Trim off blocks after generic_file_direct_write() has failed.
 - Update out of date comments about direct_io locking rules.

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

```
---
 mm/filemap.c |   32 ++++++++++++++++++++++++++++----
 1 files changed, 28 insertions(+), 4 deletions(-)

diff --git a/mm/filemap.c b/mm/filemap.c
index 0aadf5f..8959ae3 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -1925,8 +1925,9 @@ 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))) {
@@ -2240,6 +2241,29 @@ 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 __generic_file_aio_write_nolock has failed.
+  * This may happen because of:
+  * 1) Bad segment found (failed before actual write attempt)
+  * 2) Segments are good, but actual write operation failed
+  *    and may have instantiated a few blocks outside i_size.
+  *   a) in case of buffered write these blocks was already
+  *    trimmed by generic_file_buffered_write()
+  *   b) in case of O_DIRECT these blocks weren't trimmed yet.
+  *
+  * In case of (2b) these blocks have to be trimmed off again.
+  */
+ 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)) {
+   /*It is (2b) case, because segments are good*/
+   loff_t isize = i_size_read(inode);
+   if (pos + count > isize)
+    vmtruncate(inode, isize);
+  }
+ }
  mutex_unlock(&inode->i_mutex);

  if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2254,8 +2278,8 @@ 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,
--
1.5.0.1
```