
Subject: [PATCH] mm: be sure to trim blocks after direct_io has failed
Posted by [Dmitriy Monakhov](#) on Wed, 28 Feb 2007 08:55:08 GMT
[View Forum Message](#) <> [Reply to Message](#)

This is updated version of patch aimed to fix direct_io error handling issue
i've previously sent 2 weeks ago. If you don't like anything in this patch
please let me know.

Changes:

- comments added. I think now it is clearly describe things.
- patch prepared against 2.6.20-mm2

How this patch tested:

- LTP test, and other readv/writev op tests.
- fsstress test.
- manual direct_io tests.

Log:

- Move common segment checks to separate helper function.
- 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- Update out of date comments about direct_io locking rules.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 | 107 ++++++-----
1 files changed, 66 insertions(+), 41 deletions(-)

diff --git a/mm/filemap.c b/mm/filemap.c

index a9284c2..c81184c 100644

--- a/mm/filemap.c

+++ b/mm/filemap.c

@@ -1147,6 +1147,38 @@ success:

return size;

}

+/

+ * Performs necessary checks before doing a write

+ *

+ * Adjust number of segments and amount of bytes to write.

+ * Returns appropriate error code that caller should return or

+ * zero in case that write should be allowed.

+ */

+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
+ @@ -1168,24 +1200,9 @@ 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) {
@@ -2080,8 +2097,9 @@ 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))) {
@@ -2271,30 +2289,14 @@ __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;
@@ -2420,6 +2422,29 @@ 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))) {
@@ -2436,8 +2461,8 @@ 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 shutdown.
+ * May be called without i_mutex for writes to S_ISREG files.
+ * Returns -EIO if something went wrong during pagecache shutdown.
*/
static ssize_t
generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
--
1.5.0.1

```
