
Subject: [PATCH 1/2] mm: move common segment checks to separate helper function (v6)

Posted by [Dmitriy Monakhov](#) on Mon, 12 Mar 2007 07:57:36 GMT

[View Forum Message](#) <> [Reply to Message](#)

I really don't want to be annoying by sending this patchset over and over again. If anyone thinks this patch is really cappy, please comment what exactly is bad. Thank you.

Changes:

- patch was split in two patches.
- comments added. I think now it clearly describes things.
- make `generic_segment_checks()` inline
- patch prepared against 2.6.20-mm3

How is tested:

- LTP test, and other `readv/writev` op tests.

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

mm/filemap.c | 76 ++++++-----
1 files changed, 39 insertions(+), 37 deletions(-)

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

index 8e1849a..0aadf5f 100644

--- a/mm/filemap.c

+++ b/mm/filemap.c

@@ -1159,6 +1159,39 @@ 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.

+ */

+inline 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
@@ -1180,24 +1213,9 @@ 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) {
@@ -2094,30 +2112,14 @@ __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;

    count = ocount;
    pos = *ppos;
--
1.5.0.1

```
