Subject: Re: [PATCH] incorrect error handling inside generic_file_direct_write Posted by Dmitriy Monakhov on Tue, 12 Dec 2006 06:22:48 GMT

View Forum Message <> Reply to Message

Andrew Morton <akpm@osdl.org> writes:

```
> On Mon, 11 Dec 2006 16:34:27 +0300
> Dmitriy Monakhov <dmonakhov@openvz.org> wrote:
>> OpenVZ team has discovered error inside generic file direct write()
>> If generic_file_direct_IO() has fail (ENOSPC condition) 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.
>> We need truncate any block beyond i size after write have failed, do in simular
>> generic file buffered write() error path.
>>
>> Exampe:
>> open("mnt2/FILE3", O_WRONLY|O_CREAT|O_DIRECT, 0666) = 3
>> write(3, "aaaaaaa"..., 4096) = -1 ENOSPC (No space left on device)
>>
>> stat mnt2/FILE3
>> File: `mnt2/FILE3'
                                 IO Block: 4096 regular empty file
>> Size: 0
                  Blocks: 4
>> >>>>>>>>>>> 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 7b84dc8..bf7cf6c 100644
>> --- a/mm/filemap.c
>> +++ b/mm/filemap.c
>> @ @ -2041,6 +2041,14 @ @ generic_file_direct_write(struct kiocb *
     mark inode dirty(inode):
>>
>>
    }
    *ppos = end;
>> + } else if (written < 0) {
>> + loff t isize = i size read(inode);
>> + /*
```

```
>> + * generic_file_direct_IO() may have instantiated a few blocks
>> + * outside i size. Trim these off again.
>> + */
>> + if (pos + count > isize)
>> + vmtruncate(inode, isize);
>> }
>>
> XFS (at least) can call generic file direct write() with i mutex not held.
How could it be?
from mm/filemap.c:2046 generic file direct write() comment right after
place where i want to add vmtruncate()
 * 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.
 */
> And vmtruncate() expects i_mutex to be held.
generic file direct IO must called under i mutex too
from mm/filemap.c:2388
 * Called under 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,
This means XFS generic file direct write() call generic file direct IO() without
i mutex held too?
>
> I guess a suitable solution would be to push this problem back up to the
> callers: let them decide whether to run vmtruncate() and if so, to ensure
> that i mutex is held.
> The existence of generic file aio write nolock() makes that rather messy
> though.
```