Subject: Re: [PATCH] incorrect error handling inside generic_file_direct_write
Posted by Andrew Morton on Mon, 11 Dec 2006 20:40:52 GMT
View Forum Message <> Reply to Message

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, "aaaaaa"..., 4096) = -1 ENOSPC (No space left on device)
>
> stat mnt2/FILE3
> File: `mnt2/FILE3'
> Size: 0   Blocks: 4   IO Block: 4096   regular empty file
> >>>>>>>>>>>>>>>>>>>>>^^^^^^^^^^ file size is less than biggest 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.
And vmtruncate() expects i_mutex to be held.

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.
