Subject: Re: [PATCH] incorrect error handling inside generic_file_direct_write
Posted by Christoph Hellwig on Tue, 02 Jan 2007 11:17:46 GMT
View Forum Message <> Reply to Message

On Fri, Dec 15, 2006 at 10:53:18AM -0800, Chen, Kenneth W wrote:
> Christoph Hellwig wrote on Friday, December 15, 2006 2:44 AM
> > So we're doing the sync_page_range once in __generic_file_aio_write
> > with i_mutex held.
> >
> >
> > > mutex_lock(&inode->i_mutex);
> > > - ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
> > > - &iocb->ki_pos);
> > > + ret = __generic_file_aio_write(iocb, iov, nr_segs, pos);
> > > mutex_unlock(&inode->i_mutex);
> > >
> > > if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
> >
> > And then another time after it's unlocked, this seems wrong.
>
>
> I didn't invent that mess though.
>
> I should've ask the question first: in 2.6.20-rc1, generic_file_aio_write
> will call sync_page_range twice, once from __generic_file_aio_write_nolock
> and once within the function itself. Is it redundant? Can we delete the
> one in the top level function? Like the following?

Really? I'm looking at -rc3 now as -rc1 is rather old and it's definitly
not the case there. I also can't remember ever doing this - when I
started the generic read/write path untangling I had exactly the same
situation that's now in -rc3:

 - generic_file_aio_write_nolock calls sync_page_range_nolock
 - generic_file_aio_write calls sync_page_range
 - __generic_file_aio_write_nolock doesn't call any sync_page_range variant