
Subject: Re: [PATCH] incorrect error handling inside generic_file_direct_write
Posted by [Andrew Morton](#) on Tue, 12 Dec 2006 10:40:27 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, 12 Dec 2006 16:18:32 +0300
Dmitriy Monakhov <dmonakhov@sw.ru> wrote:

```
> >> but according to filemaps locking rules: mm/filemap.c:77
> >> ..
> >> * ->i_mutex (generic_file_buffered_write)
> >> * ->mmap_sem (fault_in_pages_readable->do_page_fault)
> >> ..
> >> I'm confused a little bit, where is the truth?
> >
> > xfs_write() calls generic_file_direct_write() without taking i_mutex for
> > O_DIRECT writes.
> Yes, but my question is about __generic_file_aio_write_nolock().
> As i understand _nolock suffix means that i_mutex was already locked
> by caller, am i right ?
```

Nope. It just means that __generic_file_aio_write_nolock() doesn't take the lock. We don't assume or require that the caller took it. For example the raw driver calls generic_file_aio_write_nolock() without taking i_mutex. Raw isn't relevant to the problem (although ocfs2 might be). But we cannot assume that all callers have taken i_mutex, I think.

I guess we can make that a rule (document it, add BUG_ON(!mutex_is_locked(..)) if it isn't a blockdev) if needs be. After really checking that this matches reality for all callers.

It's important, too - if we have an unprotected i_size_write() then the seqlock can get out of sync due to a race and then i_size_read() locks up the kernel.
