


```
>> + * 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 shutdown.
 */
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.
```