

Re: [bug] ata subsystem related crash with latest -git

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-10/msg06001.html>

- *From:* Linus Torvalds <torvalds@xxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 17 Oct 2007 13:10:59 -0700 (PDT)
-

On Wed, 17 Oct 2007, Jens Axboe wrote:

Well, hang on – where does it end up doing `sg_next()` on the LAST sg entry?

They pretty much **all** do, as far as I can tell.

For example, to look at the very first one:

```
#define for_each_sg(sglist, sg, nr, __i) \  
for (__i = 0, sg = (sglist); __i < (nr); __i++, sg = sg_next(sg))
```

let's say that "nr" is 1 (and that's also the allocation size), and look at what that causes.

Right. It causes us to do "`sg = sg_next(sg)`" once. Which will do what? It will increment `sg` (so that it now points past the single-entry array!) and then it will dereference that invalid entry to see if it's a chain entry!

And no, "1" is not the special case. The special case is **any** time when you iterate over as many `sg` entries as you allocated. You **always** have to leave the last one unused in order to avoid this "access past the end" problem.

The alternative is to rewrite the loop, but it's going to be ugly as hell, and you need to do that for **every*single*user** of `sg_next()`. You can re-write the above loop as something like

```
define for_each_sg(sglist, sg, nr, __i) \  
for (__i = 0, sg = NULL ; \  
__i < (nr) && sg = (sg ? sglist : sg_next(sg) ; \  
__i++))
```

Re: [bug] ata subsystem related crash with latest -git

but the important part here is that you must do that "sg_next()" *after* you have broken out of the loop, and you must basically do it one less time than you go through the loop.

IOW, any loop that leaves "sg" pointing to past the array is inevitably buggy, because it will have accessed that last past-the-end entry as part of trying to decide whether it should perhaps follow a link.

I'd argue that this is a bug, like it was in ll_rw_blk.c. I still agree that I should make the interface more robust, I just don't see where libata ends up doing the sg_next() on the last entry.

See above. I think the exact sequence may be:

ata_qc_issue()
(implicitly inlined) ata_sg_setup()
(explicitly inlined) dma_map_sg()
(macro) for_each_sg()

but I didn't see if there are other possible chains that get you to one of those invalid sg loops.

And no, it's *not* just for_each_sg(). Pretty much any "natural" loop over the SG list will cause it, because that's how you write loops in C: you almost always end up pointing to one past the last entry after the loop.

Linus

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>