

Re: [rfc] optimise unlock_page

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-05/msg07589.html>

- *From:* Hugh Dickins <hugh@xxxxxxxxxxxx>
 - *Date:* Wed, 16 May 2007 20:28:36 +0100 (BST)
-

On Wed, 16 May 2007, Nick Piggin wrote:

On Wed, May 16, 2007 at 06:54:15PM +0100, Hugh Dickins wrote:

On Sun, 13 May 2007, Nick Piggin wrote:

Well I think so, but not completely sure.

That's not quite enough to convince me!

I did ask Linus, and he was very sure it works.

Good, that's very encouraging.

Not so much from a high level view (although it does put more constraints on the flags layout), but from a CPU level... the way we intermix different sized loads and stores can run into store forwarding issues[*] which might be expensive as well. Not to mention that we can't do the non-atomic unlock on all architectures.

Ah yes, that's easier to envisage than an actual correctness problem.

The other option of moving the bit into ->mapping hopefully avoids all the issues, and would probably be a little faster again on the P4, at the expense of being a more intrusive (but it doesn't look too bad, at first glance)...

Hmm, I'm so happy with PG_swapcache in there, that I'm reluctant to cede it to your PG_locked, though I can't deny your use should take precedence. Perhaps we could enforce 8-byte alignment of struct address_space and struct anon_vma to make both bits available

Re: [rfc] optimise unlock_page

(along with the anon bit).

But I think you may not be appreciating how intrusive PG_locked will be. There are many references to page->mapping (often ->host) throughout fs/ : when we keep anon/swap flags in page->mapping, we know the filesystems will never see those bits set in their pages, so no page_mapping-like conversion is needed; just a few places in common code need to adapt.

And given our deprecation discipline for in-kernel interfaces, wouldn't we have to wait a similar period before making page->mapping unavailable to out-of-tree filesystems?

Please seek out those guarantees. Like you, I can't really see how it would go wrong (how could moving in the unlocked char mess with the flag bits in the rest of the long? how could atomically modifying the long have a chance of undoing that move?), but it feels like it might take us into errata territory.

I think we can just rely on the cache coherency protocol taking care of it for us, on x86. movb would not affect other data other than the dest. A non-atomic op _could_ of course undo the movb, but it could likewise undo any other store to the word or byte. An atomic op on the flags does not modify the movb byte so the movb before/after possibilities should look exactly the same regardless of the atomic operations happening.

Yes, I've gone through that same thought process (my questions were intended as rhetorical exclamations of inconceivability, rather than actual queries). But if you do go that way, I'd still like you to check with Intel and AMD for errata. See include/asm-i386/spinlock.h for the CONFIG_X86_OOSTORE || CONFIG_X86_PPRO_FENCE __raw_spin_unlock using xchgb: doesn't that hint that exceptions may be needed?

Hugh

-

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/>

Re: [rfc] optimise unlock_page