

Re: remove zero_page (was Re: -mm merge plans for 2.6.24)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-10/msg02574.html>

- *From:* Nick Piggin <nickpiggin@xxxxxxxxxxxxx>
 - *Date:* Tue, 9 Oct 2007 01:17:47 +1000
-

On Thursday 04 October 2007 01:21, Linus Torvalds wrote:

On Wed, 3 Oct 2007, Nick Piggin wrote:

I don't know if Linus actually disliked the patch itself, or disliked my (maybe confusingly worded) rationale?

Yes. I'd happily accept the patch, but I'd want it clarified and made obvious what the problem was – and it wasn't the zero page itself, it was a regression in the VM that made it less palatable.

OK, revised changelog at the end of this mail...

I also thought that there were potentially better solutions, namely to simply avoid the VM regression, but I also acknowledge that they may not be worth it – I just want them to be on the table.

In short: the real cost of the zero page was the reference counting on the page that we do these days. For example, I really do believe that the problem could fairly easily be fixed by simply not considering zero_page to be a "vm_normal_page()". We already *do* have support for pages not getting ref-counted (since we need it for other things), and I think that zero_page very naturally falls into exactly that situation.

So in many ways, I would think that turning zero-page into a nonrefcounted page (the same way we really do have to do for other things anyway) would be the much more *direct* solution, and in many ways the obvious one.

That was my first approach. It isn't completely trivial, but vm_normal_page() does play a part (but we end up needing a vm_normal_page() variant -- IIRC vm_normal_or_zero_page()). But taken as a whole, non-refcounted zero_page is obviously a lot more work than no zero page at all :)

Re: remove zero_page (was Re: -mm merge plans for 2.6.24)

HOWEVER – if people think that it's easier to remove zero_page, and want to do it for other reasons, *AND* can hopefully even back up the claim that it never matters with numbers (ie that the extra pagefaults just make the whole zero-page thing pointless), then I'd certainly accept the patch.

I have done some tests which indicate a couple of very basic common tools don't do much zero-page activity (ie. kbuild). And also combined with some logical arguments to say that a "sane" app wouldn't be using zero_page much. (basically -- if the app cares about memory or cache footprint and is using many pages of zeroes, then it should have a more compressed representation of zeroes anyway).

However there is a window for some "insane" code to regress without the zero_page. I'm not arguing that we don't care about those, however I have no way to guarantee they don't exist. I hope we wouldn't get a potentially useless complexity like this stuck in the VM forever just because we don't _know_ whether it's useful to anybody...

How about something like this?

From: Nick Piggin <npiggin@xxxxxxx>

The commit b5810039a54e5babf428e9a1e89fc1940fabff11 contains the note

A last caveat: the ZERO_PAGE is now refcounted and managed with rmap (and thus mapcounted and count towards shared rss). These writes to the struct page could cause excessive cacheline bouncing on big systems. There are a number of ways this could be addressed if it is an issue.

And indeed this cacheline bouncing has shown up on large SGI systems. There was a situation where an Altix system was essentially livelocked tearing down ZERO_PAGE pagetables when an HPC app aborted during startup. This situation can be avoided in userspace, but it does highlight the potential scalability problem with refcounting ZERO_PAGE, and corner cases where it can really hurt (we don't want the system to livelock!).

There are several broad ways to fix this problem:

1. add back some special casing to avoid refcounting ZERO_PAGE
2. per-node or per-cpu ZERO_PAGES
3. remove the ZERO_PAGE completely

I will argue for 3. The others should also fix the problem, but they result in more complex code than does 3, with little or no real benefit that I can see. Why?

Re: remove zero_page (was Re: -mm merge plans for 2.6.24)

Re: remove zero_page (was Re: -mm merge plans for 2.6.24)

Inserting a ZERO_PAGE for anonymous read faults appears to be a false optimisation: if an application is performance critical, it would not be doing many read faults of new memory, or at least it could be expected to write to that memory soon afterwards. If cache or memory use is critical, it should not be working with a significant number of ZERO_PAGES anyway (a more compact representation of zeroes should be used).

As a sanity check -- measuring on my desktop system, there are never many mappings to the ZERO_PAGE (eg. 2 or 3), thus memory usage here should not increase much without it.

When running a make -j4 kernel compile on my dual core system, there are about 1,000 mappings to the ZERO_PAGE created per second, but about 1,000 ZERO_PAGE COW faults per second (less than 1 ZERO_PAGE mapping per second is torn down without being COWed). So removing ZERO_PAGE will save 1,000 page faults per second, and 2,000 bounces of the ZERO_PAGE struct page cacheline per second when running kbuild, while saving less than 1 page clearing operation per second (even 1 page clear is far cheaper than a thousand cacheline bounces between CPUs).

Of course, neither the logical argument nor these checks give anything like a guarantee of no regressions. However, I think this is a reasonable opportunity to remove the ZERO_PAGE from the pagefault path.

The /dev/zero ZERO_PAGE usage and TLB tricks also get nuked. I don't see much use to them except complexity and useless benchmarks. All other users of ZERO_PAGE are converted just to use ZERO_PAGE(0) for simplicity. We can look at replacing them all and ripping out ZERO_PAGE completely if/when this patch gets in.

-

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/>