

Re: [RFC] MMIO accessors & barriers documentation

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-09/msg02981.html>

- *From:* Benjamin Herrenschmidt <benh@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 12 Sep 2006 10:46:03 +1000
-

wmb() is often used to make sure a memory store is visible to a busmastering PCI device... before the code proceeds with some more transactions in the memory space shared by the host and PCI device.

Yes and that's a different issue. It's purely memory-to-memory barriers and we already have these well defined.

The problem `_is_` with MMIO :) There, you have some ordering issues happening with some processors that we need to handle, hence the whole discussion. See below my discussion of your example

`prepare_to_read_dma_memory()` is the operation that an ethernet driver's RX code wants. And this is `_completely_` unrelated to MMIO. It just wants to make sure that the device and host are looking at the same data. Often this involves polling a DMA descriptor (or index, stored inside DMA-able memory) looking for changes.

Why would you need a barrier other than a compiler barrier() for that ?

All you need for such operations that do not involve MMIOs are the standard `wmb()`, `rmb()` and `mb()` with their usual semantics and polling for something to change isn't something that requires any of these. Only a compiler barrier (or an ugly volatile maybe). Though having a subsequent read from memory that must be done after that change happened is indeed the job of `rmb()`.

This has nothing to do with MMIO and is not what I'm describing in the document. MMIO has it's own issues especially when it comes to MMIO vs. memory coherency. I though I described them well enough, looks like not.

Re: [RFC] MMIO accessors & barriers documentation

`flush_my_writes_to_dma_memory()` is the operation that an ethernet driver's TX code wants, to precede either an MMIO "poke" or any other non-MMIO operation where the driver needs to be certain that the write is visible to the PCI device, should the PCI device desire to read that area of memory.

That's the problem. You need –different– type of barriers whether the subsequent operation to "poke" the device is an MMIO or an update in memory. Again, the whole problem is that on some out of order architectures, non-cacheable storage is on a completely different domain than cacheable storage and ordering between them requires specific barriers unless you want to ditch performances.

Thus in your 2 above examples, we have:

1– Descriptor update followed by MMIO poke. That needs ordering rule #2 in my list (memory W + MMIO W), which is today not provided by the PowerPC `writel()`, but should be according to the discussions we had and which would be provided by the barrier `memory_to_io_wb()` in my list if you chose to use relaxed ordering `__writel()` version instead for performances.

2– Descriptor update followed by update of an index in memory (so no MMIO involved). This is a standard memory ordering issue and thus a simple `wmb()` is needed there.

Currently, the PowerPC `writel()`, as I just said, doesn't provide ordering for your example #1, but the PowerPC `wmb()` does provide the semantics of both memory/memory coherency and memory/MMIO coherency (thus making it more expensive than necessary in the memory/memory case).

My goal here, is to:

– remove the problem for people who don't understand the issues by making `writel()` etc... fully ordered vs. memory. for the cases that matter to drivers. Thus the –only– case that drivers writers would have to care about if using those accessors is the memory–memory case in your second example.

– provide relaxed `__writel` etc... for people who –do– understand those issues and want to improve performances of the hot path of their driver. In order to make this actually optimal and safe, I need to precisely define in what way it is relaxed, what precise ordering semantics are provided, and provide specific barriers for each of these.

That's what I documented. If you think my document is not clear enough, I would be happy to have your input on how to make it clearer. Maybe some introduction explaining the difference above ? (re–using your examples).

Re: [RFC] MMIO accessors & barriers documentation

There are still a few questions that I listed about what we want to provide. The main one is the ordering of MMIO vs. spin_unlock. Do we want to provide that in the default writel or do we accept that we still require a barrier in that case even when using "ordered" versions of the accessors because the performance cost would be too high.

So far, I tend to prefer being fully ordered (and thus not require the barrier) but I wanted some feedback there. So far, everybody have carefully avoided to voice an firm opinion on that one though :)

Ben.

—

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/>