

# Re: Shared Memory for Application/Communication decoupling

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.arch.embedded/2006-12/msg00695.html>

---

- *From:* Christian Walter <wolti@xxxxxx>
  - *Date:* Wed, 13 Dec 2006 14:05:36 +0100
- 

Steve at fivetrees wrote:

"Christian Walter" <wolti@xxxxxx> wrote in message  
[news:newscache\\$pym6aj\\$si1@xxxxxxxxxxxxxxxx](mailto:news:newscache$pym6aj$si1@xxxxxxxxxxxxxxxx)

Yes – the OS we had was custom and used a non standard fieldbus protocol in the past. The old protocol did not set any requirements on the timing of the device. If the device was busy it only answered to special commands which where handled in an interrupt.

Building a message in an interrupt is fine; any further processing should be scheduled. I'm sure that's what you mean.

Yes – Processing in the interrupt should never be done. Assembling a message is only possible if the data is readily available because you won't block in an interrupt. The big problems are that you have problems with consistency and in my opinion it should be done like I scratched below (at least if requirements allow and what I learned from working with protocol stacks).

- an interrupt basically receives or transmits and already assembled frame.
- if a frame is complete the interrupt posts an event to a queue which wakes up a processing tasks.
- this tasks does error checking and processes everything.

Normal communication took place in the main loop where all calculations where performed.

Some time ago it was decided that we need to support additional fieldbus protocols and the project was limited in time and therefore most of the existing code base had to be used. The new protocol had timing requirements and we also needed to support reading values at any time. Therefore I worked on the operating system (C + 68k assembler) and added a preemptive multitasker together with some simple IPC primitives. This parts are finished and therefore at least theoretically available for a new design.

## Re: Shared Memory for Application/Communication decoupling

We are now in the need to support more devices in the future and also have time and resources to rework and create bigger software components. In addition we want to move away from building our own OS ;).

FWIW, I often find that "building one's own OS" is an overstated task. A simple roundrobin is easy to maintain, and actually enhances synchronism. True preemptive multitasking has its place, but the asynchrony it brings carries a lot of overhead and baggage. Again, I suspect you know this.

I can't agree on that. I come from an academic background and what I learned quite fast in industry is that no one pays you for empty miles. Porting toolchains, writing already existing components, .... All this takes time and is never in a project timeframe and certainly not in the requirements from the marketing.

An OS together with task management, cooperative and or preemptive multitasker which is portable is not so easy. And again you won't find standard components like bootloaders, ... which work out of the box.

Maybe I could have told more about our already existing system but I think it is sometimes necessary to rethink old ideas. And I did not want to bias any of the opinions.

To put it another way, ensure that the comms task can only run when the data is stable and self-consistent. If this means building comms data packets on one pass through the loop, and buffering and transmission at a more leisurely pace either by interrupts or by polling, so be it. Similarly, for reception: when the application task runs, any comms-written data should be stable: it'll be able to react to (signalled) changes of data consistently, in one place. And you get your temporal decoupling [1] for free, or more specifically as a key part of the design.

It is not possible that the communication task runs only when the currently calculated data has become stable (which is calculated during automatic measurements) because the calculation takes too long (will also take a lot of time on a fast CPU). Therefore we introduced some buffering to communicate the values from the previous calculation. Of course I was a requirement that both tasks only work on consistent data.

Do you mean that there's fast data and slow data? Can you not synchronise the snapshots available to the comms task to the slower task? Or is that what you mean by "previous calculation"?

Yes. I think we both mean the same. Synchronizing a common view of data.

## Re: Shared Memory for Application/Communication decoupling

Updating is therefore always done in atomic blocks and at well known times.

This is good ;).

I'd say again: use a messaging system, not global data. Use methods (and hence the opportunity to signal a query or a change) to access internal data. At the very least, ensure that your comms task knows nothing at all about the application task's data structure and use, and vice versa. I repeat: there be dragons.

Right now they are linked as separate modules and do not know anything from each other. They only share a common header file with SHM keys.

Hmmm. That's still global data. <Shudder.>

Yes – Basically it boils down to global data. But its usage is quite different than using global variables between modules. In this case I am quite happy with the current implementation because it is quite clean and works.

But I see that there are drawbacks inherent in the design and at least the IPC primitives should be encapsulated. Therefore I will lead the development and design of the new system into a new direction.

But as most people suggested a Message–Passing solutions I think I will go away from the SHM approach and use a message passing system with three tasks where one task acts as a coordinator. This matches my requirements, is clear from the design and using a new OS I can use the message passing primitives from there.

Wahay! You'll reap the benefits, I promise you. Decoupling is a Good Thing. Not only does it immediately simplify things, it makes you think in another way, which usually results in a simpler, cleaner system. And very often more efficient too, since all the workarounds and kludges fall out. Wahay! ;)

I hope – At least there is more time now to discuss design options and I am familiar with our products now and therefore we can make better decisions. And finally I hope that my knowledge is improving over time ;)

Steve  
<http://www.fivetrees.com>

Re: Shared Memory for Application/Communication decoupling