Why Use a Commercial Kernel?

by
Ralph Moore
***smx* Architect**

Reasons to NOT Develop An In-House Kernel

"Our application is too simple for a kernel"

I hear this statement at least twice a week. It tells me very little about the project, but it does tell me that the speaker has no vision of using a commercial kernel in his project. This statement is comparable to "I don't need a power saw because I only have to cut a few boards." Clearly, that speaker is aware only of the speed advantage of a power saw. He is not aware that the power saw makes much more precise cuts and thus his boards will fit together better and his project will come out better.

A commercial kernel is similar to a power saw. Many people think multitasking is the only benefit of a commercial kernel. They have not taken the effort to look deeply into the features it offers and they have not made an effort to envision how these features would benefit their project.

"It will only take 1-2 weeks."

People who think they can invent things quickly often know little about what they plan to invent. It is common sense that if the average kernel costs over $10,000 and the average programmer costs $7500 per month, then an acceptable kernel cannot be designed, coded, and debugged in 1-2 weeks.

Don't pay expert wages for apprentice work

Would you pay a race car mechanic's wages to change tires? Of course not. Yet, your staff is undoubtedly expert in the requirements of your industry and you are paying them accordingly. So, does it make sense to assign them to do something they know little or nothing about? Absolutely not! Such work should be subcontracted out – and buying a commercial kernel is the best way to do so. (In fact, you should be looking at the cost of a kernel in exactly the same way as you look at the cost of an outside consultant, because you are buying outside expertise in both cases. Embedded products are not sold in sufficient volumes to achieve the mass market prices of WindowsXP or Word.)

“The learning curve is too long”

Other than for Linux or Windows, I am surprised that anyone has the chutzpah to use this argument against buying a commercial kernel. It is obviously much harder to invent than to learn. Most kernels are designed to be easy to learn and use (maybe because those of us in the kernel business run into this objection so often!)

What’s not in the kernel ends up in the application

Lack of vision is the cause of this. Instead of making maximum use of proven commercial code, your crew is reinventing the wheel with fresh, buggy code. Is this the best use of their time?

An in-house kernel is never done

As your crew learns what they didn’t know about kernels, the one to two week estimate lengthens, thus robbing time from development of your product. This is the downside of not hiring outside experts in a craft that is not central to your business.

There is never time to document an in-house kernel

Naturally! You want your crew doing more important things – like finishing your project. Hence, comments in the kernel code are out of date or non-existent and there is no manual. Nor has the kernel been kept up to date with advancements in tools and processors. (Nor is there much hope of it being compatible with any other software package that you might want to add to your product.)

The inevitable crew change

A couple of years down the road you are left with a quirky, undocumented kernel, which is the heart of your product. The crew who designed it has moved on to greener pastures. (Too bad about yours, but at least they will never make that mistake again!) The rep crew is faced with a really steep learning curve (and cursing the creators of this monster). Thank God for component obsolescence! It justifies chucking the whole design and starting over with a commercial kernel. (You learned something too!)

Fiasco Prevention

Now let’s talk about doing it right. What does a commercial kernel offer, that makes business sense?

No wasted time deciding what to do, doing it, etc.

The kernel is already done. Just learn it and get on with the project!

Thorough documentation

Most commercial kernels come with a User's Guide, a Reference Manual, and well-documented code. Not only that, these are accurate and up to date. Most kernel vendors also offer training.

Proven code

A commercial kernel is not only debugged, it is *proven*. It has been used by hundreds of users in a multiplicity of ways. Moreover, it continues to be used and tested and latent bugs continue to be found and fixed. A tough kernel bug can take weeks to find and fix.

Board support and tool integration

A good kernel comes with a BSP and integration with a good tool suite. These, alone, are often worth the cost, in programming time, of the kernel. (BSP means "Board Support Package". It includes startup code and device drivers for most on-chip peripherals as well as for a specific development board.)

Structured programming

Support for multitasking is not the most important benefit of a commercial kernel. Support for structured, object-oriented programming is. A commercial kernel allows a programmer to implement his application in the form of pre-defined objects such as tasks, messages, semaphores, etc. There are firm rules for creating and using these objects. A good kernel detects misuse of objects and informs the programmer immediately, thus saving tons of debug time. More importantly, the code ends up with a structure that makes it easier to implement, to understand, to debug, and to change in the future – even by a different programmer.

Universality

Most commercial kernels do standard things in standard ways. The result is greater compatibility with other embedded software products. Good commercial kernels offer a wide selection of already integrated products such as file managers, networking stacks, graphical user interfaces, etc. You may need one of these in the future.

Technical Support

A few good words can save the kingdom…. or at least hours or days of frustrating, wasted effort. Who do your programmers talk to if you decided to do it in-house?

Conclusion

The idea to develop an in-house kernel or RTOS may be defensible technically (in a few cases), but it seldom is a good business decision.

Ralph@smxinfo.com