

When Good Code Goes Bad - Workshop on Debugging December 2006

Additional Ideas

Transcript

I do want to cover one problem that can be a little confusing.

This is actually code that I got off the web. It's code that is correct. It works properly. It's a random number generator. I've verified that it works properly. And it produces ten seemingly reasonable random numbers. Then I put it on MPP2, and it produces ten random numbers that are not at all reasonable. So a common question here is, what's wrong with the machine? Why did the machine break?

Well, we can look at the code, and this is the code that produces the values. This is sort of an example of how not to write code. But it's also a common approach.

All this code does is it requests a random number, makes it real; gets the largest integer, turns that into a real; does the division, stores the results in an array; and then prints out the array.

The reason this code isn't so good, particularly for debugging, is that we have to pull this function out to see whether or not it's making sense. We can't really tell what the values are when they're essentially being consumed by the computation.

So, I'm going to change this ever so slightly. I commented out the original line and I got a new value. I'm printing out both the largest integer and the value passed by this function. We can take a look at the results, and we can see something that's a little startling. This value is the largest integer. Now the largest four byte minus one integer is about two billion. That number's a little bit bigger than two billion. That's more about the size of an eight byte minus one bit integer.

MPP2 has a concept of I4 and I8, I4 being a four-byte integer, I8 being an eight-byte integer. The default on MPP2 is an eight-byte integer. Now, the way I compiled this code initially was I forced the selection of an I4 module, ran it and compiled it. That treated the integer as four bytes. Then I took the code, recompiled it, and ran it on MPP2 with an eight-byte integer module. And that's where we run into the disaster.

So, it really wasn't a problem with the machine. This is definitely a problem with an assumption made about what the machine's doing. It's important to be attentive to these sorts of details. It's easy enough to check whether the assumptions are correct. When you run into a problem like this, and you're not sure what to do, by all means ask a consultant.

One additional thing to talk about is comments. Basically a comment isn't necessarily the life history of the person who wrote the code, and it's not necessarily a statement about what every line of code is supposed to do. When you write a set of comments, it should tell you what the programmer had in

mind, what they were hoping to accomplish by their code. If they failed to accomplish that, then we at least know what went wrong.

It's also useful to know something about the assumptions of the code. Four-byte integers versus eight-byte integers. It turns out that the people who wrote the code that I downloaded commented their code well enough for me to know that it would fail the way I'd hoped.

So having code appropriately and reasonably commented will resolve a lot of the issues that we run into.