

Re: a dozen cpu's on a chip

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2008-05/msg01595.html>

- *From:* Robert Baer <robertbaer@xxxxxxxxxxxxx>
 - *Date:* Tue, 13 May 2008 13:47:24 -0700
-

MooseFET wrote:

On May 13, 12:04 am, Martin Brown <||||newspam...@xxxxxxxxxxxxxxxxxxxxx> wrote:

MooseFET wrote:

On May 12, 8:12 am, John Larkin
<jjlar...@xx> wrote:
[...]

show off how tricky they can be. Something
without pointers. Something
that is impossible to crash.

No nontrivial language can ever be proven to be impossible to
crash.

Minimalist languages like Modula2 have been proved to be formally
correct

I would like to see this proof. I seriously doubt that there isn't an
error in it like the one pointed out by Godel in set theory. It may
also not be a proof at all. Many so called proofs are just arguments
that a proof could be done and not actual proofs. To be a proof all
the steps must be shown.

Re: a dozen cpu's on a chip

and compilers that implement the formal specification exist.

The static testing possible with that strict Pascal like grammar catches a very large proportion of common programming errors at (pre) compile time. It is still possible to write something that will crash, but you have to try a lot harder to do it.

I write a lot in Pascal. Although the strict type checking and more obvious syntax does prevent 90% of mistakes, it still leaves the 10%. Run time checking prevents some conditions from continuing to do damage but the program still bombs.

If you include hung systems in crashed ones, the problem becomes much harder (even though it started out as impossible).

Ada's high integrity subset for safety critical work is another one.

Did you know that there was a mistake in the spec for ADA that allowed "correct" compilers to produce two different results from the same source code. I have the details around here somewhere. It was something very nonobvious about what type an expression had. It takes some pages of coding to make the problem show.

I assume that the subset was made after this was discovered.

The full Ada implementation is too much like a race horse designed by committee with way too many bells and whistles hitching a ride.

ADA is 3 of everyones favorite languages.

I suspect LISP comes pretty close to being impossible to crash, but when you include practical implementations some of the OS impurities will provide ample opportunity to bring the thing to its knees.

Since LISP isn't compiled it is a lot harder to imagine it killing a system its self but the base interpreter can't be written in LISP so

Re: a dozen cpu's on a chip

the error would be there.

For my money any routine that attempts to read from uninitialised memory, or write to memory it doesn't own should be terminated with a page fault there and then (unless the location has been pre-defined as memory mapped I/O).

Ownership should be strictly enforced for this to work. On the various X86 machines the hardware does exist that could allow the OS to lock things down. It would make the call process a lot slower but in critical systems it may be worth it.

I think that hardware designers could do a lot to improve matters. Having a call-return stack, a parameter stack, return value area, and local variables area all independant and enforced in hardware would make it much harder to mess things up.

When a pass by reference is done, the reference could contain the permissions and size information so that the called routine would be restricted to only writing where the callee wants it to.

Regards,
Martin Brown
** Posted from <http://www.teranews.com> **

The business of having programs in the same address space as data would seem to be asking for trouble. Two independent memories, one exclusively for program and another exclusively for data would help; adding (programmable) protected areas for stacks and other program areas would also be of help. But, at the end of the day, the damnOS must be written to USE those protections, and in a logical, intelligent manner.
And the M\$ gooies fail in that regard...