

Re: a dozen cpu's on a chip

Source: <http://sci.tech--archive.net/Archive/sci.electronics.design/2008-05/msg01668.html>

- *From:* MooseFET <kensmith@xxxxxxxxxx>
 - *Date:* Wed, 14 May 2008 07:46:47 -0700 (PDT)
-

On May 13, 10:19 am, Martin Brown <|||newspam...@xxxxxxxxxxxxxxxxxxxxxx>
wrote:

MooseFET wrote:

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

MooseFET wrote:

On May 12, 8:12 am, John Larkin
<jjlar...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
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 imposible 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.

It is the output of ISO WG13 so you have to buy the (very) expensive ISO
standard if you want to play with it.

Re: a dozen cpu's on a chip

I am far too cheap to pay for it just to go looking for the mistake but I am going to assert without out proof that there certainly must be one. I assert this on the basis that humans don't do proofs and that computer checked proofs are only as good as the computer that checked it which its self must have been proven and so on "turtles all the way down".

However, there are several compilers still available which implement ISO M2. Personally I don't like their choice of syntactic sugar, but I can see why they did things. I subscribed to another M2 dialect. By the time the full spec was published most industrial people had moved to other languages for purely commercial reasons.

I had heard it was "practical" reasons. They hed to get product out the door and thus had to pick a language and go with it.

The specification is in VDM-SL which can be checked by software tools.

Sly Fox Chicken Coop Gaurd Inc" at your service. The checking tools would have had to have been checked.

It isn't very easy to read though. A small piece is online as an example and is a bit out of date but gives the flavour.

<http://freepages.modula2.org/downloads/process2.pdf>

It doesn't state what happens if the extra add of the increment would overflow. Assume an 8 bit unsigned V

for V:= 1 to 245 by 70

you get loops for V=1, 71, 141, 211 and then what? Does it throw and overflow or not?

[...]

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

Halting problems are notoriously difficult.

Re: a dozen cpu's on a chip

Re: a dozen cpu's on a chip

Yes as in impossible with a tool that checks in a finite time. There is always some bizare sort of looping that will hang the checking tool or be missed.

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

No. Although I was never a great fan of Ada.

something very nonobvious about what type an expression had. It takes some pages of coding to make the problem show.

I think it was John Barnes at Praxis that did the high integrity stuff.

BTW: The ADA folks said "there shall be no subsets" early on.

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

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 the error would be there.

Re: a dozen cpu's on a chip

It is a common misconception that Lisp is always interpreted. I was involved long ago in the development of a Common Lisp Compiler. There are incremental Lisp compilers which may look from the outside like interpreters but generate fast native code. The whole compiler and its libraries was written in Lisp and bootstrapped with a Lisp interpreter onto new hardware. Only the deepest layer of OS interface was done in assembler it ran on the Mac and later on the PC.

Interesting. "Lots of Inconvenient Single Parenthesis" has come a long way.

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.

OS/2 was pretty good in this respect. Step out of line and your process gets swatted before it can do any harm.

Linux seems to be fairly good on that subject too, but that is not what I meant. I was referring to protection on the level of the routine call.

Subroutines could be "pure" in that they only ever reference their parameters and only modify their defined return areas.

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 independent and enforced in hardware would make it much harder to mess things up.

Limiting address range each process is allowed to access will do.

Re: a dozen cpu's on a chip

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.

You do have to be careful you don't create something that is unweildly. x86 has an array BOUND instruction but I can't recall ever seeing it used in anger.

Doing it at the hardware level isn't all that hard. The reference only needs to contain the starting address and the length in special hardware. Accesses would always use the reference and an offset into it. The hardware would have to have an adder and comparison circuit special to the addressing process.

The Modula2 solution was to have pragmas that allowed development code to insert preambles and postambles to defend against stack overflow, numeric faults, page faults etc and a kernel to return a traceback or postmortem dump that would identify the failing code. The traceback still worked in production code but the testing was disabled.

Borland pascal would do full range and overflow and stack checking. It did initialized object tests on virtual method dispatches.

The first PC compilers with these capabilities were around in the mid 1980's distributed by Logitech (now better know for its mouse).

Regards,

Martin Brown

** Posted from <http://www.teranews.com>**