Re: Disobeying jet engines − why?

Source: http://sci.tech−archive.net/Archive/sci.electronics.design/2008−01/msg05068.html

- *From*: John Larkin <jjlarkin@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
- *Date*: Thu, 31 Jan 2008 11:19:06 −0800

On Thu, 31 Jan 2008 11:13:47 +0000, Martin Brown
<|||newspam|||@nezumi.demon.co.uk> wrote:

> In message <fnq85j$7et$1@xxxxxxxx>, Jan Panteltje
> <pNaonStpealmtje@xxxxxxxxx> writes
>
> > On a sunny day (Wed, 30 Jan 2008 07:08:01 −0800) it happened John Larkin
> > <jjlarkin@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in
> > <4f41q3hq25ra5ld18dfa5jvlcnkge8klnc@xxxxxxx>:
> >
> > > Sometimes the right thing to do is to buy the
> > > correct development tool
> > > for the job. Anyone who attempts to write a
> > > database from scratch for a
> > > PC wants their head examining (and that was
> > > true almost from the early
> > > days of CPM). Same with anyone who
> > > attempts to debug embedded code in a
> > > hard RT environment without using the right
> > > tools.
> >
> > Umm, I should have my head examined. Twice.
> >
> > John
> >
> > Hey, John, I agree.
> > Did not want to comment at first, and that is especially about debug.
> > If you know your code, then you need no debugger! HIGH LEVEL
> > LANGUAGES
> > LIKE HEX or ASM need no debugger (I ain't kidding).
>
> And what if it is someone else's code (which is frequently the case on
> large projects) and almost always the case when contracting.

> Most I will do in huge programs in C (C++ is not really a language but a disability, Stroussup did not know how to program, and that is why he created C++),
> is add some printf() statements.

What a dinosaur. You might at least use the ASSERT, VERIFY and TRACE macros so thoughtfully provided in modern implementations.

> Honestly last time I used a debugger was in 1983? They tried to sell me all sorts of stuff, ICE, hell, you should be able to understand what is going on from what happens.

So you come back to a piece of kit that isn't working and have to binary chop to find the failing line of code each time carefully reproducing whatever situation caused it to fail. Or alternatively waiting hours or days for the failure situation to happen again by chance.

The relatively simple post mortem debug tools we had in the mid 80's would log the failing address, failure code, stack top and registers. From that and the link map you could go to the exact failing line of code and knowing how it failed usually work out why.

If you mean "failing" as "crashing", true. But our bare−metal assembly stuff never crashes once we get any initial stupidities out, in the first hour or so of testing. After that, "failure" becomes defined as not functioning as desired, and a post−mortem tool won't even recognize that when it happens, so can't checkpoint the event.

The best way to fix a functional problem is to note the behavior and re−read the code. The problem will usually be there in black−and−white. Next best is to step through actual code execution, with a background debugger.

If code is truly crashing intermittently, something in the design philosophy is badly wrong. That's isn't and can't be cured by debugging, because all you'll find are the once−a−day or once−a−week bugs, not the once−a−month ones. Such bugs should simply never exist by design.

> This method worked to allow the last remaining bugs in the shipped systems to be practically eliminated as the number deployed increased and failures became rarer and rarer. You can even get MS compilers to do this trick now.

Finding "practically" all the bugs isn't good enough, and failures should happen never, not "rarely." Our goal is to ship the first production unit with zero bugs, and we most always do. If your philosophy is to ship with some number of bugs, you will.

John

.