

Debugging goes back to the future

It is not possible to run a computer backwards. So when debugging a program, whenever a point of interest has passed, it is gone. The only way to go back is to restart the program from the beginning. This makes debugging very inefficient, since the programmer is investing a lot of effort each time for small amounts of incremental information.

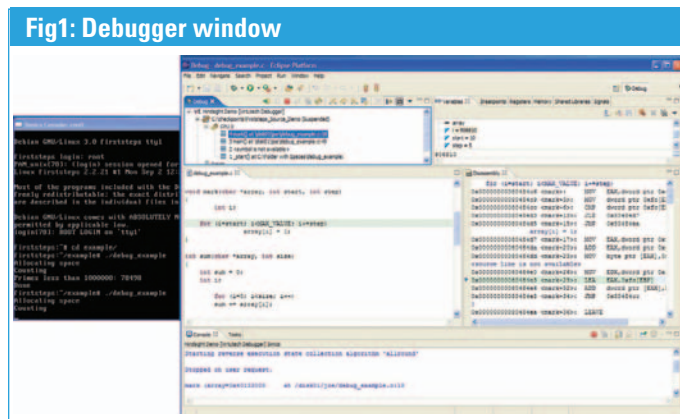
Virtutech has developed Simics Hindsight to make it possible to execute a program in reverse and step backwards through the code and is said to be the first general-purpose development tool for reversible execution of arbitrary software running on arbitrary systems.

When running software on a computer, once a particular instruction has been performed, it is impossible to go backwards through the code. If it turns out a bug resulted from an error earlier in the program (which is frequently the case), there's no going back. Every programmer has had the experience of stepping over a procedure call without looking into the details, only to discover later that the error originated within that procedure. Or wondering "how did that pointer become zero?" In each case it is necessary to restart the program, and run it to just before problem occurred. That might take a few seconds, or several hours. Or it simply isn't possible – in a parallel or networked system, recreating a particular event can be almost arbitrarily time consuming.

Simics Hindsight retains the whole timeline, allowing the programmer to go back-and-forth across any subset of the software to zero in on the problem. With each sweep back-and-forth, new breakpoints, watchpoints, and other instrumentation can be incrementally added until the problem is fully understood.

Since computers themselves do not actually run backwards, it is not possible to provide reversible execution with code running directly on hardware.

Instead, a virtual model of the hardware is built that is accurate enough such that the software



under development cannot tell the difference. The actual binary code runs unchanged, even for real-time operating systems, device drivers, firmware, network protocol stacks, etc. In addition to having this level of fidelity, the model also runs fast enough that software engineers prefer using it as part of their edit-compile-debug loop. This basic capability has been available in Virtutech's system software development platform, Simics, for several years. Simics Hindsight adds the ability to run the model of that same system backwards.

Classical debugging commands all have an analogous reverse command. For example, all debuggers include a *continue* command that runs the program (forwards) until either a breakpoint is reached or the program terminates.

Simics Hindsight has a corresponding *reverse continue* command which runs the program backwards until either a breakpoint is reached or the program starts. Simics Hindsight can even *unboot* an operating system, running the code backwards until it reaches the initial hardware reset or power-on state.

Simics has extremely high performance when executing code forwards. Reversible execution would be an interesting academic curiosity if it only executed in reverse at glacially slow speeds, but Simics Hindsight executes code in reverse almost as fast as it executes code forwards.

Without Simics Hindsight, a programmer has to struggle

to stop the program just before something bad happens, such as detecting corruption of a data structure. It can be extremely difficult to define a breakpoint if a record is added wrongly after the first million correctly added records. The breakpoint would have to be defined to ignore the first million times through the code and stop on just the correct occurrence. With Simics Hindsight, it is possible to execute the code until the problem is detected, which is typically obvious, and then run in reverse until the root cause is determined, which is almost certainly associated with the record just added.

Simics Hindsight is used to debug a program in much the same way as a normal debugger. The program under test is launched in debug mode, the programmer sets breakpoints at interesting places if necessary and the program starts. At some point, either a breakpoint is reached, an assertion in the code fails indicating trouble, or the underlying hardware detects a problem such as division by zero. Now, instead of having to rerun the program with additional breakpoints, Simics Hindsight permits new breakpoints to be added and the program run backwards. If the problem seems to have occurred just before it was detected, the code can be reverse-single-stepped.

Debugging using Simics Hindsight can be performed in a much more direct way than with traditional approaches. Here is an example of a tricky problem. A kernel mode device

driver for a network interface contains a bug. The bug is in the code that handles the packets that are received with incorrect checksums. As part of the processing, a zero is written to the wrong location.

Since the driver is running in kernel mode, in this case it overwrites a system data structure. Packets with bad checksums are rare, so the system usually runs correctly under normal operation. Even when bad packets are received, the errant zero might have no effect if it is written to an area that is not in use.

But if the zero is written to an active data structure, the system may eventually crash when the overwritten value is accessed and a zero acquired instead.

Suddenly the system halts having attempted to access address zero.

A cursory examination of where the system halted makes it clear that an important data structure has been overwritten. In principle, the problem could be almost anywhere in the kernel, in an interrupt routine, in a device driver or any other code for which access to kernel memory is allowed.

With a traditional debugger, it is almost impossible to debug this sort of kernel problem. Even with a kernel debugger, the only thing to be done is to put a watchpoint on the location and re-boot the system. But this will completely alter the timing of the system and it is unknown when the next bad checksum packet will arrive. The symptoms this time around will be completely different. If bad checksum packets occur sufficiently rarely, the system will just appear to be unstable, crashing occasionally for an unknown reason.

With Simics Hindsight it is possible to set a watchpoint on the errant zero. Execution is then reversed until the watchpoint is hit. When execution stops at the watchpoint, it stops in the network device driver bad checksum handling routine. With that bit of information it is probably trivial to spot the error in the code and fix it.