http://catless.ncl.ac.uk/Risks/19.49.html

What really happened on Mars Rover Pathfinder

Mike Jones <mbj@MICROSOFT.com>
Sunday, December 07, 1997 6:47 PM

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing" -- bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

This week at the IEEE Real-Time Systems Symposium I heard a fascinating keynote address by David Wilner, Chief Technical Officer of Wind River Systems. Wind River makes VxWorks, the real-time embedded systems kernel that was used in the Mars Pathfinder mission. In his talk, he explained in detail the actual software problems that caused the total system resets of the Pathfinder spacecraft, how they were diagnosed, and how they were solved. I wanted to share his story with each of you.

VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.

Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft. A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).

The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue. The spacecraft also contained a communications task that ran with medium priority.

Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the(high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

This scenario is a classic case of priority inversion.

HOW WAS THIS DEBUGGED?

VxWorks can be run in a mode where it records a total trace of all interesting system events, including context switches, uses of synchronization objects, and interrupts. After the failure, JPL engineers spent hours and hours running the system on the exact spacecraft replica in their lab with tracing turned on, attempting to replicate the precise conditions under which they believed that the reset occurred. Early in the morning, after all but one engineer had gone home, the engineer finally reproduced a system reset on the replica. Analysis of the trace revealed the priority inversion.

HOW WAS THE PROBLEM CORRECTED?

When created, a VxWorks mutex object accepts a boolean parameter that indicates whether priority inheritance should be performed by the mutex. The mutex in question had been initialized with the parameter off; had it been on, the low-priority meteorological thread would have inherited the priority of the high-priority data bus thread blocked on it while it held the mutex, causing it be scheduled with higher priority than the medium-priority communications task, thus preventing the priority inversion.
Once diagnosed, it was clear to the JPL engineers that using priority inheritance would prevent the resets they were seeing.

VxWorks contains a C language interpreter intended to allow developers to type in C expressions and functions to be executed on the fly during system debugging. The JPL engineers fortuitously decided to launch the spacecraft with this feature still enabled. By coding convention, the initialization parameter for the mutex in question (and those for two others which could have caused the same problem) were stored in global variables, whose addresses were in symbol tables also included in the launch software, and available to the C interpreter. A short C program was uploaded to the spacecraft, which when interpreted, changed the values of these variables from FALSE to TRUE. No more system resets occurred.

ANALYSIS AND LESSONS

First and foremost, diagnosing this problem as a black box would have been impossible. Only detailed traces of actual system behavior enabled the faulty execution sequence to be captured and identified.

Secondly, leaving the "debugging" facilities in the system saved the day. Without the ability to modify the system in the field, the problem could not have been corrected.

Finally, the engineer's initial analysis that "the data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance" was exactly wrong. It is precisely in such time critical and important situations where correctness is essential, even at some additional performance cost.

HUMAN NATURE, DEADLINE PRESSURES

David told us that the JPL engineers later confessed that one or two system resets had occurred in their months of pre-flight testing. They had never been reproducible or explainable, and so the engineers, in a very human-nature response of denial, decided that they probably weren't important, using the rationale "it was probably caused by a hardware glitch".

Part of it too was the engineers' focus. They were extremely focused on ensuring the quality and flawless operation of the landing software. Should it have failed, the mission would have been lost. It is entirely understandable for the engineers to discount occasional glitches in the less-critical land-mission software, particularly given that a spacecraft reset was a viable recovery strategy at that phase of the mission.

THE IMPORTANCE OF GOOD THEORY/ALGORITHMS

David also said that some of the real heroes of the situation were some people from CMU who had published a paper he'd heard presented many years ago who first identified the priority inversion problem and proposed the solution. He apologized for not remembering the precise details of the paper or who wrote it. Bringing things full circle, it turns out that the three authors of this result were all in the room, and at the end of the talk were encouraged by the program chair to stand and be acknowledged.
They were Lui Sha, John Lehoczky, and Raj Rajkumar. When was the last time you saw a room of people cheer a group of computer science theorists for their significant practical contribution to advancing human knowledge? :-)It was quite a moment.

POSTLUDE

For the record, the paper was:

L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An
Approach to Real-Time Synchronization. In IEEE Transactions on Computers,
vol. 39, pp. 1175-1185, Sep. 1990.