Risk Forum: 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.

---

ob Rahe <bob@hobbes.dtcc.edu>
Wed, 10 Dec 1997 10:30:01 EST

In RISKS-19.49, Mike Jones <mbj@MICROSOFT.com> writes a fascinating account of "What really happened on Mars Rover Pathfinder". It's just that kind of

behind-the-scenes articles that make RISKS so great. But, near the end he talks about the 3 authors from CMU who had written a paper in 1990 that he says had first identified the priority inversion problem and proposed a solution.

Not to take anything from those authors but that exact type of problem was addressed in at least one mainframe operating system way back in the early 1970s. The Burroughs MCP (Master Control Program) that ran (and still runs as the Unisys A-Series MCP) their mainframe systems, had a locking scheme that could produce exactly that kind of problem. A lock procured by a low-priority task which then get's pre-empted by a medium- priority task, could lock out a high priority task. Their solution at the time (and still is the last time I looked) was to bump the priority of any task procuring a global system lock such as the one mentioned in the article. Simply, if priorities ran from 0 (low) to 99 (high), procuring a lock under their algorithm would add 100 to the priority of the locking program. When the lock was released the priority would be dropped back by the same amount.

While not as elegant a solution, in my opinion, it certainly was adequate for the problem given that these were not 'real-time' systems, and was certainly an enduring solution.

Another example of there apparently being nothing new in software as well as under the sun?

Bob Rahe, Delaware Tech&Comm Coll., Computer Center, Dover, Delaware
Internet: bob@hobbes.dtcc.edu

Ken Tindell <ken@nrtg.com>
Fri, 12 Dec 1997 19:16:15 -0000

>This scenario is a classic case of priority inversion.

So classic that it has happened before many times in many projects. And I fear will continue to happen. Today, people are building critical real-time systems based on Windows NT. But NT doesn't implement priority inheritance. Instead it contains a "priority randomizer" which randomly selects tasks and alters their priorities in the hope that eventually the priority inversion goes away. Whilst this may be adequate for a general-purpose computer in a workstation environment, this is unlikely to be adequate for a critical real-time system.

>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.

I must point out that their work appeared much earlier in technical reports and conference proceedings and was widely cited before the 1990 paper appeared. Interested readers might like to read the following paper, which

gives an historical perspective on when major results were made available:

"Fixed Priority Scheduling: An Historical Perspective", Audsley, Burns, Davis, Tindell, Wellings, Real-Time Systems journal, March 1995, Volume 8, No. 2/3, pp. 173-198.

I find it outrageous that engineers in 1997 are building critical systems that contain serious defects that were detectable and correctable ten years ago. I do wonder at what point failure to be aware of these risks constitutes negligence.

What really happened on Mars Rover Pathfinder (Mike Jones, R-19.49)

"Fred B. Schneider" <fbs@CS.Cornell.EDU>
Mon, 5 Jan 1998 18:29:27 -0500 (EST)

Readers of RISKS could get the wrong impression about who did what and when from what David Wilner is reported to have said in Mike Jones' item on the Mars Pathfinder mission in RISKS-19.49. This note attempts to provide some missing information.

Jones' Mars Pathfinder article ends with:

> "THE IMPORTANCE OF GOOD THEORY/ALGORITHMS
>
> David [Wilner] 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. ...
>
> 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."

Actually, a "priority inheritance" protocol can be found in

B. W. Lampson and D. D. Redell.
Experience with processes and monitors in Mesa.
{\it Communications of the ACM},
vol. 23, no. 2, pp. 105--117, February 1980.

which is a reprint of a paper that appeared in Dec 1979 (7th ACM Symposium on Operating System Principles). Below is the relevant excerpt; it is almost -- but not exactly -- what Sha et al. investigate.

> "4.3 Priorities

In some applications it is desirable to use a priority scheduling discipline for allocating the processor(s) to processes which are not waiting. Unless care is taken, the ordering implied by the assignment of priorities can be subverted by monitors. Suppose there are three priority levels (3 highest, 1 lowest), and three processes P_1, P_2, and P_3, one running at each level. Let P_1 and P_3 communicate using a monitor M. Now consider the following sequence of events:

P_1 enters M
P_1 is preempted by P_2
P_2 is preempted by P_3
P_3 tries to enter the monitor, and waits for the lock
P_2 runs again, and can effectively prevent P_3 from running, contrary to the purpose of the priorities

A simple way to avoid this situation is to associate with each monitor the priority of the highest-priority process which ever enters that monitor. Then whenever a process enters a monitor, its priority is temporarily increased to the monitor's priority..."

So, it would be incorrect to credit Sha et al. for first *identifying* the problem or for first *proposing a protocol* to solve it. Lampson & Redell do not give any quantitative analysis of their prio scheme, though.

The development of this thread of research in real-time scheduling is accurately described in section 5 of Audsley et al., as noted by Ken Tindell.

A parable comes to mind. School children in the U.S. are taught that "Columbus discovered America". Ultimately they learn that Columbus was preceded by, among others, the Vikings. So why aren't they taught that "The vikings discovered America"? Perhaps it is because when Columbus discovered America, it stayed discovered.

---

ike Jones <mbj@MICROSOFT.com>
Fri, 9 Jan 1998 14:13:58 -0800

> Date: Monday, December 15, 1997 10:28 AM
> From: Glenn E Reeves <Glenn.E.Reeves@jpl.nasa.gov>
> Subject: Re: [Fwd: FW: What really happened on Mars?]
>
> What really happened on Mars ?
>
>By now most of you have read Mike's (mbj@microsoft.com) summary of Dave
>Wilner's comments given at the IEEE Real-Time Systems Symposium. I don't
>know Mike and I didn't attend the symposium (though I really wish I had now)
>and I have not talked to Dave Wilner since before the talk. However, I did
>lead the software team for the Mars Pathfinder spacecraft. So, instead of
>trying to find out what was said I will just tell you what happened. You
>can make your own judgments.
>

>I sent this message out to everyone who was a recipient of Mike's original
>that I had an e-mail address for. Please pass it on to anyone you sent the
>first one to. Mike, I hope you will post this wherever you posted the
>original.
>
>Since I want to make sure the problem is clearly understood I need to step
>through each of the areas which contributed to the problem.
>
>THE HARDWARE
>
>The simplified view of the Mars Pathfinder hardware architecture looks like
>this. A single CPU controls the spacecraft. It resides on a VME bus which
>also contains interface cards for the radio, the camera, and an interface to
>a 1553 bus. The 1553 bus connects to two places : The "cruise stage" part
>of the spacecraft and the "lander" part of the spacecraft. The hardware on
>the cruise part of the spacecraft controls thrusters, valves, a sun sensor,
>and a star scanner. The hardware on the lander part provides an interface
>to accelerometers, a radar altimeter, and an instrument for meteorological
>science known as the ASI/MET. The hardware which we used to interface to
>the 1553 bus (at both ends) was inherited from the Cassini spacecraft. This
>hardware came with a specific paradigm for its usage : the software will
>schedule activity at an 8 Hz rate. This **feature** dictated the
>architecture of the software which controls both the 1553 bus and the
>devices attached to it.
>
>THE SOFTWARE ARCHITECTURE
>
>The software to control the 1553 bus and the attached instruments was
>implemented as two tasks. The first task controlled the setup of
>transactions on the 1553 bus (called the bus scheduler or bc_sched task) and
>the second task handled the collection of the transaction results i.e. the
>data. The second task is referred to as the bc_dist (for distribution)
>task. A typical timeline for the bus activity for a single cycle is shown
>below. It is not to scale. This cycle was constantly repeated.
>
> |< ------------ .125 seconds ----------------------->|
>
> |<***************| |********| |**>|
>
> |<- bc_dist active ->| bc_sched active
> |< -bus active ->| |<->|
>
>
> ----|----------------|--------------------|--------|---|---|-------
>   t1   t2   t3   t4 t5 t1
>
>The *** are periods when tasks other than the ones listed are executing.
>Yes, there is some idle time.
>
>t1 - bus hardware starts via hardware control on the 8 Hz boundary. The
>transactions for the this cycle had been set up by the previous execution of

>the bc_sched task.
>t2 - 1553 traffic is complete and the bc_dist task is awakened.
>t3 - bc_dist task has completed all of the data distribution
>t4 - bc_sched task is awakened to setup transactions for the next cycle
>t5 - bc_sched activity is complete
>
>The bc_sched and bc_dist tasks check each cycle to be sure that the other
>had completed its execution. The bc_sched task is the highest priority task
>in the system (except for the vxWorks "tExec" task). The bc_dist is third
>highest (a task controlling the entry and landing is second). All of the
>tasks which perform other spacecraft functions are lower. Science
>functions, such as imaging, image compression, and the ASI/MET task are
>still lower.
>
>Data is collected from devices connected to the 1553 bus only when they are
>powered. Most of the tasks in the system that access the information
>collected over the 1553 do so via a double buffered shared memory mechanism
>into which the bc_dist task places the latest data. The exception to this
>is the ASI/MET task which is delivered its information via an interprocess
>communication mechanism (IPC). The IPC mechanism uses the vxWorks pipe()
>facility. Tasks wait on one or more IPC "queues" for messages to arrive.
>Tasks use the select() mechanism to wait for message arrival. Multiple
>queues are used when both high and lower priority messages are required.
>Most of the IPC traffic in the system is not for the delivery of real-time
>data. However, again, the exception to this is the use of the IPC mechanism
>with the ASI/MET task. The cause of the reset on Mars was in the use and
>configuration of the IPC mechanism.
>
>THE FAILURE
>
>The failure was identified by the spacecraft as a failure of the bc_dist
>task to complete its execution before the bc_sched task started. The
>reaction to this by the spacecraft was to reset the computer. This reset
>reinitializes all of the hardware and software. It also terminates the
>execution of the current ground commanded activities. No science or
>engineering data is lost that has already been collected (the data in RAM is
>recovered so long as power is not lost). However, the remainder of the
>activities for that day were not accomplished until the next day.
>
>The failure turned out to be a case of priority inversion (how we discovered
>this and how we fixed it are covered later). The higher priority bc_dist
>task was blocked by the much lower priority ASI/MET task that was holding a
>shared resource. The ASI/MET task had acquired this resource and then been
>preempted by several of the medium priority tasks. When the bc_sched task
>was activated, to setup the transactions for the next 1553 bus cycle, it
>detected that the bc_dist task had not completed its execution. The
>resource that caused this problem was a mutual exclusion semaphore used
>within the select() mechanism to control access to the list of file
>descriptors that the select() mechanism was to wait on.
>
>The select mechanism creates a mutual exclusion semaphore to protect the

>"wait list" of file descriptors for those devices which support select. The
>vxWorks pipe() mechanism is such a device and the IPC mechanism we used is
>based on using pipes. The ASI/MET task had called select, which had called
>pipeIoctl(), which had called selNodeAdd(), which was in the process of
>giving the mutex semaphore. The ASI/ MET task was preempted and semGive()
>was not completed. Several medium priority tasks ran until the bc_dist task
>was activated. The bc_dist task attempted to send the newest ASI/MET data
>via the IPC mechanism which called pipeWrite(). pipeWrite() blocked, taking
>the mutex semaphore. More of the medium priority tasks ran, still not
>allowing the ASI/MET task to run, until the bc_sched task was awakened. At
>that point, the bc_sched task determined that the bc_dist task had not
>completed its cycle (a hard deadline in the system) and declared the error
>that initiated the reset.
>
>HOW WE FOUND IT
>
>The software that flies on Mars Pathfinder has several debug features within
>it that are used in the lab but are not used on the flight spacecraft (not
>used because some of them produce more information than we can send back to
>Earth). These features were not "fortuitously" left enabled but remain in
>the software by design. We strongly believe in the "test what you fly and
>fly what you test" philosophy.
>
>One of these tools is a trace/log facility which was originally developed to
>find a bug in an early version of the vxWorks port (Wind River ported
>vxWorks to the RS6000 processor for us for this mission). This trace/log
>facility was built by David Cummings who was one of the software engineers
>on the task. Lisa Stanley, of Wind River, took this facility and
>instrumented the pipe services, msgQ services, interrupt handling, select
>services, and the tExec task. The facility initializes at startup and
>continues to collect data (in ring buffers) until told to stop. The
>facility produces a voluminous dump of information when asked.
>
>After the problem occurred on Mars we did run the same set of activities
>over and over again in the lab. The bc_sched was already coded so as to
>stop the trace/log collection and dump the data (even though we knew we
>could not get the dump in flight) for this error. So, when we went into the
>lab to test it we did not have to change the software.
>
>In less that 18 hours we were able to cause the problem to occur. Once we
>were able to reproduce the failure the priority inversion problem was
>obvious.
>
>HOW WAS THE PROBLEM CORRECTED
>
>Once we understood the problem the fix appeared obvious : change the
>creation flags for the semaphore so as to enable the priority inheritance.
>The Wind River folks, for many of their services, supply global
>configuration variables for parameters such as the "options" parameter for
>the semMCreate used by the select service (although this is not documented
>and those who do not have vxWorks source code or have not studied the source

>code might be unaware of this feature). However, the fix is not so obvious
>for several reasons :
>
>1) The code for this is in the selectLib() and is common for all device
>creations. When you change this global variable all of the select
>semaphores created after that point will be created with the new options.
>There was no easy way in our initialization logic to only modify the
>semaphore associated with the pipe used for bc_dist task to ASI/MET task
>communications.
>
>2) If we make this change, and it is applied on a global basis, how will
>this change the behavior of the rest of the system ?
>
>3) The priority inversion option was deliberately left out by Wind River in
>the default selectLib() service for optimum performance. How will
>performance degrade if we turn the priority inversion on ?
>
>4) Was there some intrinsic behavior of the select mechanism itself that
>would change if the priority inversion was enabled ?
>
>We did end up modifying the global variable to include the priority
>inversion. This corrected the problem. We asked Wind River to analyze the
>potential impacts for (3) and (4). They concluded that the performance
>impact would be minimal and that the behavior of select() would not change
>so long as there was always only one task waiting for any particular file
>descriptor. This is true in our system. I believe that the debate at Wind
>River still continues on whether the priority inversion option should be on
>as the default. For (1) and (2) the change did alter the characteristics of
>all of the select semaphores. We concluded, both by analysis and test, that
>there was no adverse behavior. We tested the system extensively before we
>changed the software on the spacecraft.
>
>HOW WE CHANGED THE SOFTWARE ON THE SPACECRAFT
>
>No, we did not use the vxWorks shell to change the software (although the
>shell is usable on the spacecraft). The process of "patching" the software
>on the spacecraft is a specialized process. It involves sending the
>differences between what you have onboard and what you want (and have on
>Earth) to the spacecraft. Custom software on the spacecraft (with a whole
>bunch of validation) modifies the onboard copy. If you want more info you
>can send me e-mail.
>
>WHY DIDN'T WE CATCH IT BEFORE LAUNCH ?
>
>The problem would only manifest itself when ASI/MET data was being collected
>and intermediate tasks were heavily loaded. Our before launch testing was
>limited to the "best case" high data rates and science activities. The fact
>that data rates from the surface were higher than anticipated and the amount
>of science activities proportionally greater served to aggravate the
>problem. We did not expect nor test the "better than we could have ever
>imagined" case.

>
>HUMAN NATURE, DEADLINE PRESSURES
>
>We did see the problem before landing but could not get it to repeat when we
>tried to track it down. It was not forgotten nor was it deemed unimportant.
>
>Yes, we were concentrating heavily on the entry and landing software. Yes,
>we considered this problem lower priority. Yes, we would have liked to have
>everything perfect before landing. However, I don't see any problem here
>other than we ran out of time to get the lower priority issues completed.
>
>We did have one other thing on our side; we knew how robust our system was
>because that is the way we designed it.
>
>We knew that if this problem occurred we would reset. We built in
>mechanisms to recover the current activity so that there would be no
>interruptions in the science data (although this wasn't used until later in
>the landed mission). We built in the ability (and tested it) to go through
>multiple resets while we were going through the Martian atmosphere. We
>designed the software to recover from radiation induced errors in the memory
>or the processor. The spacecraft would have even done a 60 day mission on
>its own, including deploying the rover, if the radio receiver had broken
>when we landed. There are a large number of safeguards in the system to
>ensure robust, continued operation in the event of a failure of this type.
>These safeguards allowed us to designate problems of this nature as lower
>priority.
>
>We had our priorities right.
>
>ANALYSIS AND LESSONS
>
>Did we (the JPL team) make an error in assuming how the select/pipe
>mechanism would work ? Yes, probably. But there was no conscious decision
>to not have the priority inversion enabled. We just missed it. There are
>several other places in the flight software where similar protection is
>required for critical data structures and the semaphores do have priority
>inversion protection. A good lesson when you fly COTS stuff - make sure you
>know how it works.
>
>Mike is quite correct in saying that we could not have figured this out
>**ever** if we did not have the tools to give us the insight. We built many
>of the tools into the software for exactly this type of problem. We always
>planned to leave them in. In fact, the shell (and the stdout stream) were
>very useful the entire mission. If you want more detail send me a note.
>
>SETTING THE RECORD STRAIGHT
>
>First, I want to make sure that everyone understands how I feel in regard to
>Wind River. These folks did a fantastic job for us. They were enthusiastic
>and supported us when we came to them and asked them to do an affordable
>port of vxWorks. They delivered the alpha version in 3 months. When we had

>a problem they put some of the brightest engineers I have ever worked with
>on the problem. Our communication with them was fantastic. If they had not
>done such a professional job the Mars Pathfinder mission would not have been
>the success that it is.
>
>Second, Dave Wilner did talk to me about this problem before he gave his
>talk. I could not find my notes where I had detailed the description of the
>problem. So, I winged it and I sure did get it wrong. Sorry Dave.
>
>ACKNOWLEDGMENTS
>
>First, thanks to Mike for writing a very nice description of the talk. I
>think I have had probably 400 people send me copies. You gave me the push
>to write the part of the Mars Pathfinder End-of-Mission report that I had
>been procrastinating doing.
>
>Special thanks to Steve Stolper for helping me do this. The biggest thanks
>should go to the software team that I had the privilege of leading and whose
>expertise allowed us to succeed: Pam Yoshioka, Dave Cummings, Don Meyer,
>Karl Schneider, Greg Welz, Rick Achatz, Kim Gostelow, Dave Smyth,
>Steve Stolper. Also, Miguel San Martin, Sam Sirlin, Brian Lazara (WRS),
>Mike Deliman (WRS), Lisa Stanley (WRS)
>
>Glenn Reeves, Mars Pathfinder Flight Software Cognizant Engineer
>glenn.e.reeves@jpl.nasa.gov

Re: Priority Inversion and early Unix

Greg Rose <ggr@qualcomm.com>
Fri, 09 Jan 1998 08:11:19 +1100

"Fred B. Schneider" <fbs@CS.Cornell.EDU>, in "What really happened on Mars Rover Pathfinder (Mike Jones, R-19.49)" R-19.53, mentions Lampson and Redell's 1980 paper.

John Lions' 1977 book "Commentary on UNIX 6th Edition", which contains the source code for this early version of Unix, and an extremely lucid examination of that code, has recently been republished (Peer-to-Peer Communications, ISBN 1-57398-013-7).

Lions (p8-4) comments:

> "Some critical sections of code are executed by interrupt handlers. To protect other sections of code whose outcome may be affected by the handling of certain interrupts, the processor priority is raised temporarily high enough before the critical section is entered to delay such interrupts until it is safe, when the processor priority is reduced again. There are of course a number of conventions which interrupt handling code should observe ..."

The code to which this refers was of the form:

```
oldpri = spl7();
/* critical region here */
spl(oldpri);
```

The processor priorities of the PDP-11 were allowed to dictate this structure to some extent; priority 7 was the highest, and disabled all interrupts. There was no attempt to set the priority to the lowest necessary, as the lack of different levels (0, 4, 5, 6, 7 for device interrupts) meant that 7 was almost always the lowest such level, although in some cases spl6() was used analogously, that being the clock interrupt priority. Most local critical regions, such as device drivers, used the simpler:

```
spl5();
...
spl0();
```

(Here I have used 5 as an example of a known interrupt priority for this device. All "normal" code ran at a processor priority level of 0, with a separate software priority and context switching mechanism.) Clearly Thompson and Ritchie had thought about and found a way to solve the priority inversion problem for the limited case of Unix on a single CPU. Their code predates Lions' commentary by a couple of years at least (1975 or earlier); I don't know exactly which version of Unix introduced that construction.

Greg Rose, QUALCOMM Australia, 6 Kingston Avenue, Mortlake NSW 2137 Australia
http://people.qualcomm.com/ggr/ ggr@qualcomm.com +61-2-9743 4646