What really happened on Mars?

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

La misión Mars Pathfinder fue ampliamente proclamada como "perfecta" en los días siguientes luego de su aterrizaje el 4 de julio de 1997. Entre sus éxitos se incluía su poco convencional "aterrizaje" - rebotando en la superficie marciana rodeada de bolsas de aire, desplegando el *Sojourner rover* y reuniendo y transmitiendo voluminosas cantidades de datos a la Tierra, incluyendo imágenes panorámicas que fueron un éxito en la Web. Pero luego de unos días en la misión, no mucho después que el Pathfinder inició la recolección de dato meteorológicos, la nave espacial empezó a experimentar reinicios totales del sistema, cada uno resultando en pérdidas de datos. La prensa reportó estos fallos con términos como "fallos de software "(*software glitches*) y "la computadora estaba intentando hacer muchas cosas al mismo tiempo".

Durante una charla en la semana del I Simposio de Sistemas en Tiempo Real de la IEEE, David Wilner, CTO de Wind River Systems, la empresa detrás de VxWorks el kernel de sistemas embebidos en tiempo real que fue usado en el Mars Pathfinder, explicó en detalle los errores de software reales que causaron los reinicios de la nave espacial Pathfinder, cómo fueron diagnosticados y cómo fueron resueltos.

VxWorks provee *preemptive priority scheduling* de hilos. Las tareas en la nave Pathfinder fuero ejecutadas como hilos con prioridades que fueron asignadas de la forma usual en la que se reflejan la urgencia relativa de estas tareas.

El Pathfinder contenía un "bus de información" el cual se podría concebir como un área compartida de memoria usada para pasar información entre diferentes componentes de la nave espacial. Una tarea de gestión del bus corría frecuentemente con una alta prioridad para mover ciertas clases de datos dentro y fuera del bus de información. El acceso al bus se sincronizaba por medio de *locks* de exclusión mutua (mutexes).

Las tareas de recolección de datos meteorológicos corrían como un hilo de baja prioridad poco frecuente, y usadas por el bus de información para publicar sus datos. Cuando publicaba sus datos, se adquiría un mutex, se escribía en el bus y se liberaba el mutex. Si una interrupción causaba el que hilo del bus de información fuera programado(*scheduled*) mientras este mutex estaba retenido(*held*) y si el hilo del bus de información intentaba adquirir el mismo mutex con el fin de recuperar los datos publicados, esto podía hacer que se bloqueara en el mutex, esperando hasta que el hilo meteorológico fuera liberara el mutex antes de poder continuar. La nave espacial también contenía una tarea de comunicaciones que corría con una prioridad media.

La mayoría del tiempo esta combinación trabajaba bien. Sin embargo, muy poco frecuente era posible que una interrupción ocurriera que causaba que la tarea de comunicaciones (prioridad media) fuera programada(*scheduled*) durante el corto intervalo durante el cual el bus de información(alta prioridad) estaba bloqueado esperando por el hilo meteorológico (baja prioridad). En este caso, la tarea de comunicaciones, teniendo una prioridad mayor que la tarea meteorológica, no podría correr, lo cual impedía a su ve que la tarea bloqueada del bus de información corriera. Luego de que pasara algun tiempo, el temporizador vigilante (*watchdog*) se apagaría, notando que la tarea

de bus de datos no había sido ejecutado por algún tiempo, concluyendo que algo había sucedido drásticamente mal y reiniciando el sistema por completo.

Este escenario es un clásico de inversión de prioridades.

How was this debugged? VxWorks puede correr en un mode donde registra un rastro total de todas los eventos interesantes del sistema, incluyendo cambios de contexto, usos de sincronización de objetos e interrupciones. Luego de la falla, los ingenieros del JPL pasaron horas y horas correinto el sistema en la replica exacta de la nave espacial en su laboratorio con el rastreo encendido, intentando replicar las condiciones precisas bajo las cuales ellos creían que el reinicio ocurría. Temprano en la mañana, luego de que todos los ingenieros menos uno se habían ido a sus casas, el ingeniero finalmente reprodujo el reinicio de sistema en la replica. Análisis en el rastro reveló la inversión de la prioridad.

How was the problem corrected? Cuando se creó, un objeto mutex VxWorks aceptaba un parámetro booleano que indica si la herencia de prioridad debía de ser realizada por el mutex. El mutex en cuestión había sido inicializado con el parametro en `off`, si hubiera estado `on`, el hilo meteorológico de baja prioridad habría heredado la prioridad del hilo bloqueado de más alta prioridad en el bus de datos mientras retenía el mutex, causando que fuera programado(*scheduled*) con una prioridad más alta que la prioridad media de la tarea de comunicaciones y así previendo la inversión de prioridad. Una vez diagnosticado, era claro para los ingenieros del JPL que usar herencia de prioridades iba a prevenir los reinicios que estaban viendo.

VxWorks contiene un intérprete de lenguaje C destinado a permitir a los desarrolladores escribir en C expresiones y funciones que van a ser ejecutadas al vuelo durante el *debugeo* del sistema. Los ingenieros decidieron fortuitamente lanzar la nave espacial con esta característica todavía activada. Por convención de código, el parámetro de inicialización para el mutex en cuestión fue almacenado en una variable global cuyas direcciones estaban en tablas de símbolos incluidas en el software inicial y disponibles en el intérprete de C. Un pequeño programa en C fue subido a la nave espacial, que cuando fue interpretado, cambió los valores de estas variables de `FALSE` a `TRUE`. No ocurrieron más reinicios.

Analysis and Lessons. Primero que todo, el diagnóstico de este problema en una caja negro hubiera sido imposible. Solamente a través de rastros detallados del comportamiento real del sistema se permitió capturar e identificar la ejecución de la secuencia defectuosa.

Segundo, el haber dejado las opciones de "debugging" habilitadas en el sistema salvó el día. Sin la habilidad de modificar el sistema en "caliente" el problema no hubiera sido corregido.

Finalmente, el análisis inicial del ingeniero en el cual decía que "la tarea del bus datos se ejecuta muy frecuentemente y es crítica en tiempo – no deberíamos pasar tiempo extra en ella para realizar herencia de prioridades" estaba equivocado. Es precisamente en ese tiempo crítico y situaciones importantes en donde la exactitud es esencial, inclusive a un costa de un impacto adicional en el rendimiento.

Human Nature, Deadline Presures. David comentó que luego los ingenieros del JPL confesaron que uno o dos reinicios del sistama habían ocurrido durante los meses de pruebas pre-vuelo. Nunca pudieron ser reproducibles o explicables, y por ello los ingenieros en una muy humana respuesta a la negación, decidieron que eso probablemente no era importante usando el razonamiento de "eso fue probablemente causado por un fallo de hardware". Parte de ellos fue en lo que se concentraron los ingenieros. Ellos estaban extremadamente enfocados en asegurar la calidad y operación perfecta del software de aterrizaje. Si hubiera fallado, la misión se hubiera perdido. Es totalmente entendible para los ingeniero descontar fallos ocasionales en software de misión de tierra menos crítico, particularmente dado que un reicinio de la nave espacial era una estrategia de recuperación viable en esa fase de la misión.

The Importance of Goot Theory/Algorithms. David también dijo que algunos de los héroes reales del problema fueron algunas personas del CMU [1] quienes publicaron un artículo del cual él había escuchado alguno años atrás en donde primero se identificaba el problema de la inversión de prioridades y se proponía una solución. Curiosamente los autores de dicho artículo estaban entre la audiencia: Lui Sha, John Lehoczky, and Raj Rajkumar. Al final de la charla fueron motivados *chair* a ponerse de pie y ser reconocidos.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?
2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?
3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?
4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?
5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

No ocurrieron más reinicios en el sistema.

[1] Carnegie Mellon University