Report for the

Seminar Series on Software Failures

Mars Pathfinder: Priority Inversion Problem

Submitted By: Risat Mahmud Pathan

"Even when you think you've tested everything that you can possibly imagine, you're *wrong*"

-- Glenn E. Reeves

(Pathfinder's Software Team Leader)

Preface

This report presents the software failure known as "priority inversion problem" that occurred in Mars Pathfinder which was designed and built at the Jet Propulsion Laboratory (JPL), California Institute of Technology. The problem due to priority inversion caused the spacecraft to reset itself several times after it was landed on Mars on July 4, 1997. Resetting the spacecraft resulted in significant delay in capturing scientific data, which was critical for the mission given that the lifetime of the spacecraft was limited. The details of the technical problem, debugging and correcting the problem are presented in this report along with the lessons that are learned.

The failure of Mars Pathfinder was discussed by David Wilner, Chief Technical Officer of Wind River System, in his keynote talk at the 18th IEEE Real-Time Systems Symposium (RTSS) in December 2-5, 1997. Wind River supplied the VxWorks Real-Time Operating Systems for Mars Pathfinder mission. Mike Jones (a researcher in the Operating Systems Research Group at Microsoft) who was present during David Wilner's keynote talk wrote a report based on this keynote regarding the priority inversion problem in Mars Pathfinder. He emailed this report to his friends and colleagues. And, later this email was widely redistributed in the Internet. After 8 days Mike Jones sent out his email, Glenn. E. Reeves, the software team leader of Mars Pathfinder at JPL, wrote another email with more technical details regarding the priority inversion problem of Mars Pathfinder, how they debugged and solved it from Earth.

This report is based on these two emails cited in [1, 2] and two magazine articles [3,4]. However, to understand the priority inversion problem which is a synchronization problem in real-time systems, I will also formally present the problem and its solution based on [5].

1. Introduction

Safety-critical real-time systems have both functional and non-functional requirements. The functional requirement specifies what an application does while the non-functional requirement specifies the quality of the functional behaviors. Examples of non-functional behaviors are throughput, energy consumption, timeliness, and so on. In the context of real-time systems, the correctness of the system depends not only on the logical output of the functions, but also on *when* the output is generated. In other words, generating the correct output within a certain *deadline* is of utmost importance for safety-critical real-time systems. Spacecraft is an example of a safety-critical real-time system where missing the deadline of some function may compromise the success of the mission which often costs several billion dollars for design, development and implementation.

Mars pathfinder is the first mission of NASA's Discovery program for investigating the atmosphere and other factors of Mars [8]. Although the project was successful from scientific point of view, its path to success was not smooth at the beginning. The system suffered from priority inversion problem – a real-time synchronization problem – soon after it was landed on Mars on July 4, 1997. The spacecraft was *resetting* itself due to the priority inversion problem and caused significant delay in collecting scientific data from the surface of the red planet [3, 6]. However, the problem was mitigated in few days by patching the onboard software from Earth. In this report, the priority inversion problem, its consequences, debugging, correcting the problem are discussed in details. The priority inversion problem is presented first as the means to understand the other details of the problem occurred in Mars Pathfinder.

1.1 Priority Inversion Problem

Many real-time applications (e.g., control and monitoring) are modeled as a collection of *periodic tasks*. Each periodic task become available for execution at the beginning of the period and must complete its execution before next period begins (i.e., the deadline of the task). Each instance of a periodic task is called a job.

Scheduling algorithm plays the vital role in determining which task to dispatch to processor for execution, at which time instant, when several tasks are available for execution. A task is said to be *active* when it has arrived and has not completed its execution. *Preemptive fixed-priority scheduling* is a preferable means in the industry for executing periodic tasks: each task is statically assigned one fixed priority and the scheduler always dispatches the highest priority active task by preempting, if any, the execution of a lower priority task [7]. In other words, a higher priority task can preempt the execution of a lower priority task and a lower priority task cannot preempt the execution of a higher priority task. A lower-priority task resumes its execution later when there is no higher priority active task awaiting execution.

There are mature fixed-priority scheduling algorithms for scheduling a set of periodic real-time tasks with the assumption that tasks are independent, i.e., the tasks do not share any other resource except the processing platform. However, this assumption does not always hold and problem begins when tasks are assumed to share other resources. ***Priority inversion problem*** is a synchronization problem due to shared resources in real-time systems.

Tasks generally share resources, for example, data structure, main memories, files, processor registers, I/O units, communication bus, and so on. It is important to enforce effective synchronization mechanisms when multiple tasks share a resource in order to ensure consistent view of the system. One way to ensure such synchronization is using semaphore for each shared resource. A task holding a semaphore has exclusive access to that resource. No task can hold a semaphore if it is currently locked by another task (i.e., at most one task can execute in the critical section).

We denote S^R the semaphore for shared resource R. A job invokes **wait(S^R)** and **signal(S^R)** commands to grab and release the resource R, respectively. If resource R is already in use by some job J2 of a lower priority task, then another job J1 of a higher priority task is blocked when invokes **wait(S^R)**. After job J2 releases resource R by calling **signal(S^R)**, job J1 can grab the semaphore and gets access to resource R. Note that a higher priority tasks can preempt the execution of a lower priority task. A lower priority task can block the execution of a higher priority task if they share some resource and the resource is currently in possession of the lower priority task. To see how it happens, consider the following two jobs J1 and J2 where J1 has higher fixed priority than job J2 and resource R is shared between the two jobs.

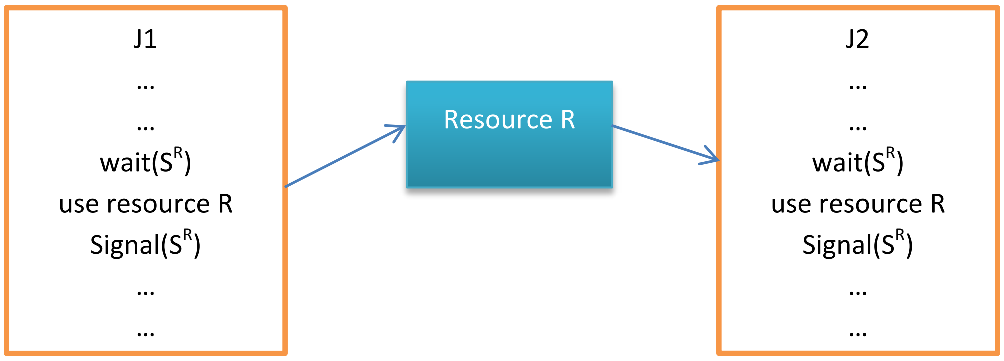


Figure 1: The request and release requests of the shared resource R by jobs J1 and J2. Blocking of the higher priority job J1 by the lower priority job J2 can happen depending on when tasks are executed.

Consider the following execution pattern in Figure 2 of the two jobs J1 and J2 where prio(J1) > prio(J2). This situation in Figure 2 is called *unavoidable blocking* where the higher priority job J1 must be blocked since the lower priority job J2 is holding R when J1 needs it.

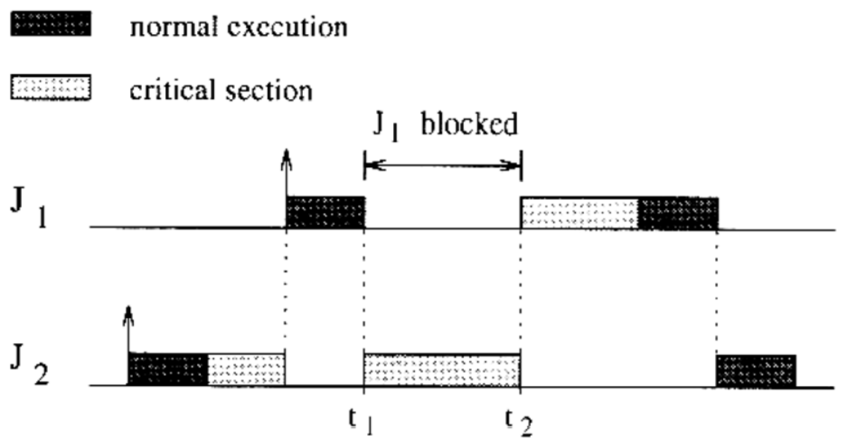


Figure 2: Unavoidable blocking of job J1 by job J2. Job J2 starts its normal execution and then access the semaphore and executes in the critical section. While J2 is executing its critical section, job J1 arrives, preempt the execution of J2 and starts its normal execution. At time instant t_1, job J1 needs resource R and is blocked since the semaphore of R is locked by job J2. At this moment, job J2 starts execution and completes the critical section by time instant t_2 and releases the semaphore. Then J1 starts execution by preempting job J2 and completes. Finally, job J2 resumes its execution and finishes it execution.

Now consider three jobs J1, J2 and J3 such that prior(J1)>prior(J2)>prio(J3). Assume that job J1 and J3 shares a resource and J2 **<u>does not</u>** share any resource with other tasks. The following scenario in Figure 3 is an example of priority inversion: a medium priority task is prioritized over a higher priority task while a lower priority task blocks the higher priority task.

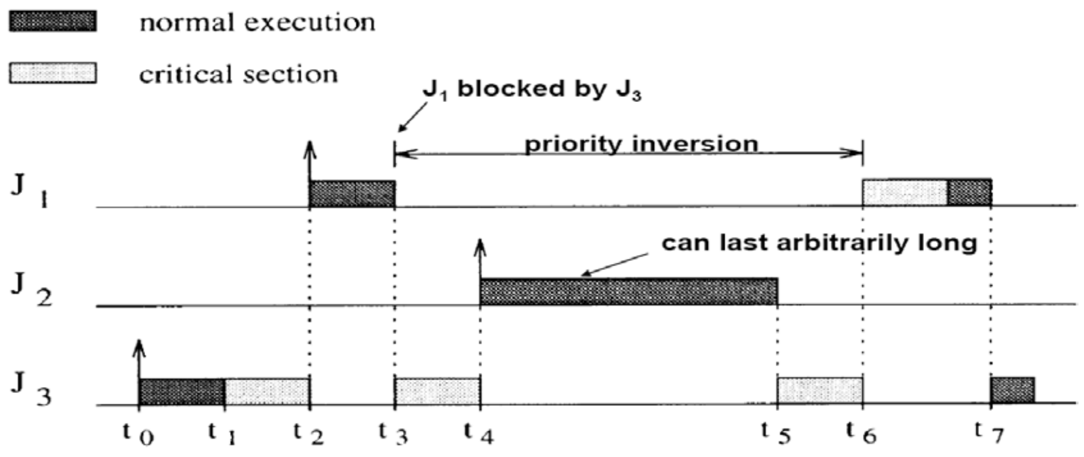


Figure 3: Priority inversion problem. Job J3 starts it normal execution at time t_0, accesses the shared resource at time t_1 and starts execution in the critical section. At time t_2, a higher priority job J1 arrives and preempts job J3 and executes up to time t_3 without requiring the shared resource. At time t_3, job J1 attempts to access the resource and is blocked since J3 has the resource. Then, job J3 starts execution of the critical section using the shared resource. However, at time t_4, a medium priority job J2 arrives, preempts job J3 and executes its normal execution. The medium priority job does not share any resource with any other jobs. Now while the medium priority job J2 is executing, the highest priority job J1 is blocked. Job J2 can block J1 arbitrary long even though job J1 has higher priority and does not share any resource with J2. It feels that priorities of J1 and J2 are inverted and this is called the priority inversion problem.

A protocol to resolve the priority inversion problem is proposed by Sha et al. in [5]. This is called the priority inheritance protocol (PIP): ***a lower priority task temporarily inherits the highest priority of all the blocking tasks***. In other words, when a lower priority task blocks a higher priority task, it temporality assumes the priority of the higher priority blocked task. Figure 4 shows how PIP solves the priority inversion problem.

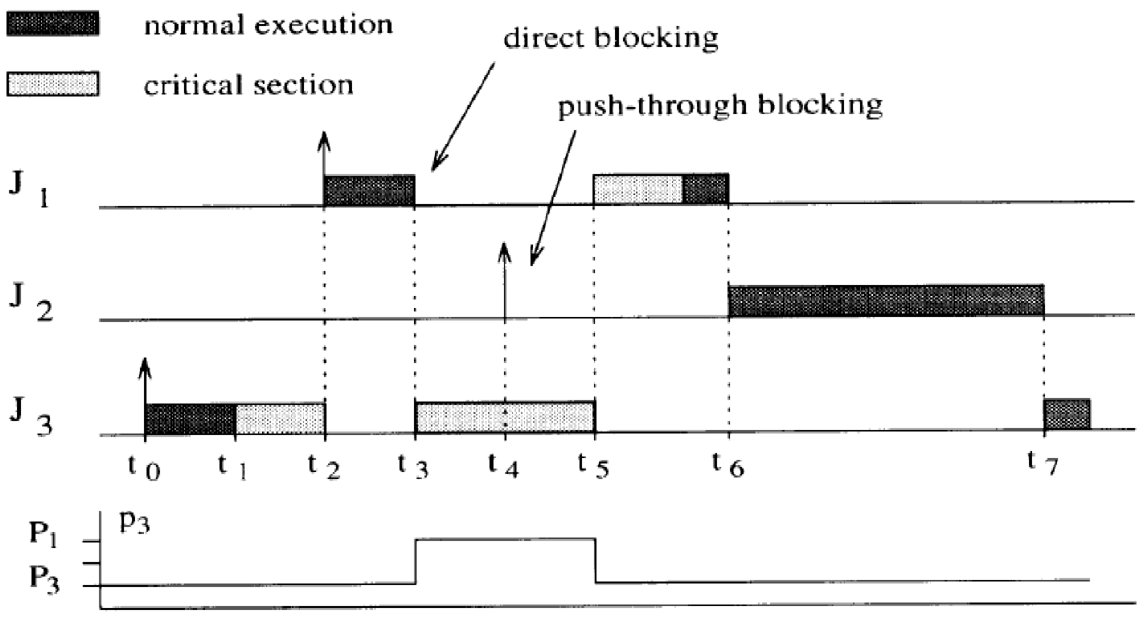


Figure 4: Priority inheritance protocol. When the medium-priority job J2 arrives at time t_4, job J2 cannot preempt the execution of job J3 since J3 has already assumed (inherited) the highest priority when blocked job J1 at time t_3. After finishing critical section execution of job J3 at time at time t_5, the higher priority job J1 can access the semaphore and can finish its execution. The highest priority job J1 is not blocked when the medium priority task is executed.

However, PIP can lead to deadlock [5]. We will not go into the details of such issues because the basic principle of PIP is enough to understand the priority inversion problem in Mars Pathfinder. Next chapter presents the case for Mars Pathfinder.

2. Mars Pathfinder

Mars Pathfinder is the first completed mission in NASA's Discovery program for planetary missions with highly focused scientific goals [6, 8]. The project started in October 1993 and incurred a total cost of 265 million dollars. Development time was 3 years, launched in December 4, 1996, and landed on Mars on July 4, 1997. The last data transmission was recorded on September 27, 1997.

The spacecraft had one instrumented ***lander*** and a free- moving ***rover***. The estimated life-times were 30 days for the lander and 7 days for the rover. The lander and rover survived 3 times and 12 times than the estimated lifetime, respectively. Figure 5 shows the lander and the free moving rover.

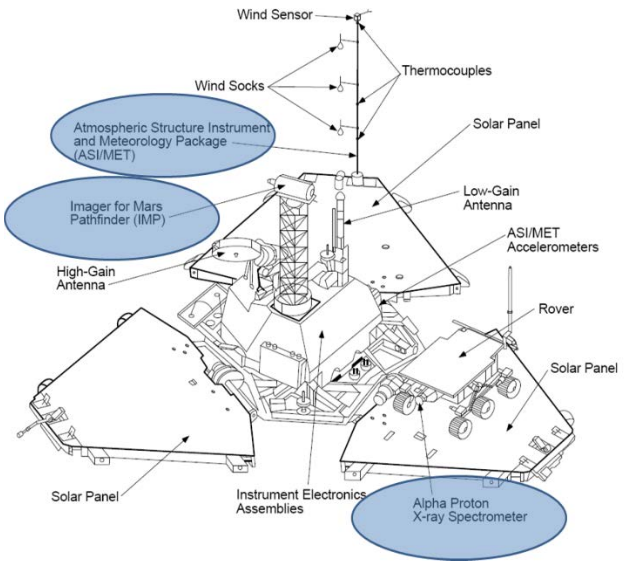

Wind Sensor
Wind Socks
Thermocouples
Atmospheric Structure Instrument and Meteorology Package (ASI/MET)
Solar Panel
Imager for Mars Pathfinder (IMP)
Low-Gain Antenna
ASI/MET Accelerometers
High-Gain Antenna
Rover
Solar Panel
Solar Panel
Instrument Electronics Assemblies
Alpha Proton X-ray Spectrometer

Figure 5: The lander and the rover. The right picture was sent by the Pathfinder from Mars.

The lander and the rover carried the following three important scientific instruments:

1. Imager for Mars Pathfinder (IMP): Hosted on the lander. Used for imaging the surface of Mars and helped to navigate the rover.

2. Atmospheric Structure Instrument and Meteorology Package (ASI/MET): Hosted on the lander. It was used to acquire atmospheric information (e.g., pressure, temperature, wind).

3. Alpha Proton X-ray Spectrometer (APXS): Hosted on the rover. It was designed to determine the elements that make up the rocks and soil on Mars.

These instruments accomplished a lot on behalf of Pathfinder: returned 2.3 billion bits of information, including more than 16,500 images from the lander, 550 images from the rover, more than 15 chemical analyses of rocks and soil, extensive data on winds and other weather factors.

2.1 The Problem

In few days after landing, the spacecraft began experiencing total system ***resets***. This happened about a half dozen times after the landing and each resulting in losses of data (according to press); in fact no data was lost, but collection was delayed by a day (critical for mission lifetime) [3]. The source of the problem was due to priority Inversion which subsequently caused a deadline-miss of a critical task, which was identified by a watchdog timer, and finally, the action in such faulty scenario was to reset the spacecraft. To understand the details, the overview of Mars Pathfinder's HW and SW is presented next.

2.2 Hardware and Software

The engineers at JPL used IBM's RS6000 processor and Wind River vxWorks RTOS. The hardware architecture of Mars Pathfinder is given in Figure 7.

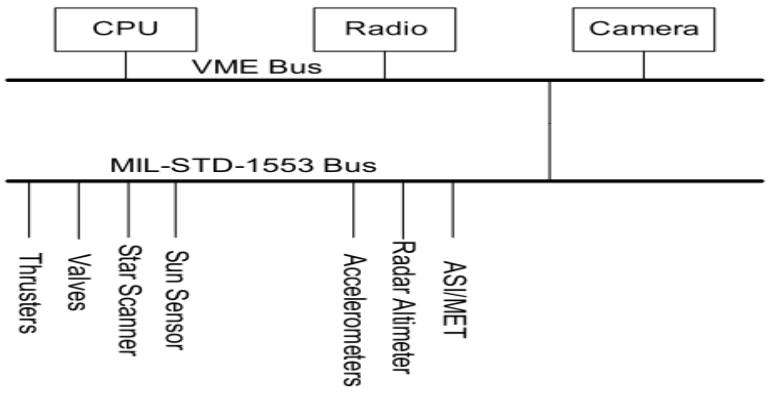


Figure 6: The hardware architecture of Mars Pathfinder

The functionalities and equipment on Pathfinder are integrated using two buses. The VME bus connected the CPU, radio (for communication) and the camera. The MIL-STD-1553 bus connected the VME bus, the lander part and the cruse part. And, the lander part consists of the ASI/MET, radar altimeter and accelerometers while the cruse part consists of sun scanner, star scanner, valves, and thrusters. Communication to and from CPU, radio and camera is done using the 1553 bus with the lander and cruse parts.

Wind River's VxWorks RTOS was used. VxWorks provides preemptive fixed-priority scheduling. Data from the lander part has to be transmitted through the 1553 bus to the radio in order to be transmitted to the Earth. On the other hand, command signal from the CPU had to move through the 1553 bus to the cruise and the lander part. The RTOS employs a cyclic scheduler @ 8 Hz rate (i.e., the same schedule is repeated in every 0.125 sec). The management of 1553 bus was implemented as two important and critical tasks

- **bc_sched:** bus scheduler task decides who will transmit and transmits the schedule for the next cycle.
- **bc_dist:** bus distribution task decides who will receive.

There are other tasks, for example, the "communication_task" for the radio that transmits data to the earth and the science function "ASI/MET task". The fixed priority ordering that is statically assigned by the engineers has the following relationship:

prio("bc_sched") > prior("bc_dist") > prio("communication_task") > prio("ASI/MET_task")

Using watchdog timer, the bc_sched task checks at the beginning of its execution whether bc_dist task had completed its execution in the previous cycle. *This test was violated on Mars Pathfinder and caused the resets.* The question is why the deadline of bc_dist was missed? To understand this, we need to understand how data is distributed by bc_dist.

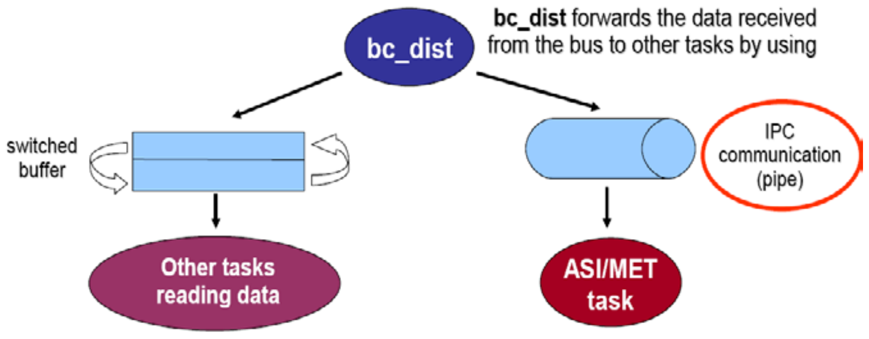


Figure 7: Data distribution mechanism by bc_dist task which uses pipe() facility for IPC with the ASI/MET task.

The bc_dist invokes the select() function to use VxWork's pipe() facility to distribute data to the ASI/MET task. In other words, the pipe() facility is used by the bc_dist task to provide command to the ASI/MET task in order to tell what scientific activities it must perform next. Similarly, the ASI/MET task also uses the pipe facility to provide the scientific data to be transferred through the 1553 bus to the radio and finally to Earth. The data structure (i.e., "waitlist" file descriptor) associated to the reading and writing sides of the pipe are shared resources protected by mutexes. In other words, the bc_dist (a higher priority task) and ASI/MET (a lower priority task) both shared the data structure of the pipe(). The following sequences of event occurred in the system and caused the reset (see Figure 8):

- ASI/MET called select() and grabs the mutex to update the shared "waitlist" at time t1.
- But before releasing the mutex, ASI/MET is preempted from higher priority task Bc_Dist at time t2.
- Bc_Dist when wanted to do IPC with the ASI/MET at time t3 cannot grab the mutex and get blocked at time t3.

- The ASI/MET task holding the mutex is further preempted by several other medium priority tasks (i.e., communication task, accelerometers and radar altimeters).
- By this time bc_sched task was activated at time t4 and it detected (using watchdog timer) at time t5 that the bc_dist task had not completed its execution.
- The reaction to this (deadline miss) by the spacecraft was to reset the computer at time t5. bc_sched reacted by reinitializing the lander's hardware and software and terminating all ground command activities (complete system reset).

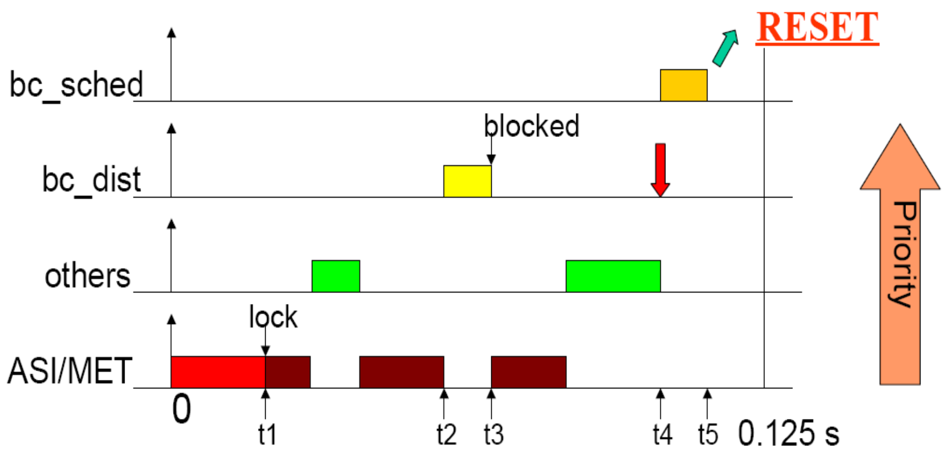


Figure 8: The schedule in which the priority inversion occurred and caused resetting

The failure, i.e., resetting the system, is a case of "priority inversion". The higher priority bc_dist task *was blocked* by the much lower priority ASI/MET task. The solution was obvious: priority inheritance. The priority inheritance flag for the mutex was set to "off" in VxWorks RTOS for performance reasons. After debugging on the pathfinder replica at JPL, engineers discovered the cause of malfunctioning as a priority inversion problem (i.e., priority Inheritance was disabled on pipe semaphores). The on-board software was updated from earth and semaphore parameters were changed to enable priority inheritance. The system was tested for possible consequences on system performance or other possible anomalies but everything was fine at that point.

How was it debugged? How did they detect that it is priority inversion? How was it corrected and patched? Some insights and lessons learned considering these questions are discussed in the remained of this report.

2.2 How was it debugged?

The software on Mars Pathfinder had several debug features. One of these tools is a *trace/log facility.* The feature remained in the software in the final version of the design because the engineers at JPL have the philosophy that "*test what you fly and fly what you test*". So, they did not remove the debug facility; it was there in Mars. After the problem occurred on Mars, they run the same set of recorded activities (sent by Pathfinder before resetting) over and over again in the lab and within three weeks they were able to reproduce the error in the replica at JPL. The priority inversion problem was obvious. The solution is to enable priority inheritance by setting the mutex flag for the select() calls of ASI/MET to "on". However, the fix is not so obvious for several reasons:

Concern 1: Setting the mutex flag is a global option and thus applicable to all mutex. Enabling it for ASI/MET would enable it for other tasks. How would this change the behavior of the rest of the system?

Concern 2: The priority inheritance option was deliberately "off" by Wind River for optimum performance. How will performance degrade if we turn priority inheritance "on"?

Concern 3: Would the select() mechanism become incorrect if priority inheritance was enabled ?

Wind River concluded that the performance impact would be minimal and that the behavior of select() would not change. The JPL engineers tested, analyzed and concluded that changing the flag on a global basis had no adverse impact. So, they decided to patch the software in Mars by enabling the priority inheritance option.

2.3 How the patch was uploaded?

VxWorks contained a C language interpreter to execute statements on the fly during debugging. The JPL engineers decided to launch the spacecraft with this feature still enabled. A short C program was uploaded to the spacecraft, which when interpreted, changed the values of the mutex flag for priority inheritance from false to true. No more system reset occurred!

3. Lessons Learned and Conclusion

There are several lessons learned from this case study. **They did not anticipate the worst-case.** Pathfinder's antenna performed better than expected because they were able to point the antenna at Earth much better. Thus, the medium priority (antenna/communication) task had lot more to transmit on Earth. Consequently, the high load of the antenna task preempt the ASI/MET task for longer period of task while the bc_dist was awaiting execution, and ultimately, missed its deadline. The bc_schd detected it via watchdog and reset the system. Determining the worst-case is crucial for system development.

Debugging tools were important in the final version. Only detailed traces enabled the faulty execution sequence to be identified. The debug facilities (C interpreter) enabled the mutex flag in Mars.

Project deadline leads to prioritize activities. There were one or two system resets in pre-flight testing; but not reproducible and they thought those as probably due to "hardware glitch”. However, they had it in mind but did not address it due to shortage of time. They were focused on ensuring the quality and flawless operation of the landing software. **Project deadline leaded to priority inversion.** The JPL engineers did not think of how the select/pipe mechanism would work. They somehow trusted the VxWorks implementation. No concrete decision was made regarding whether to enable priority inheritance “on” or “off”. Glenn Reeves quoted “A good lesson when flying COTS - make sure you know how it works”.

Reset was not effective for permanent faults. Because a system being blocked or hanged can never be excluded, every system should contain a watchdog timer task. However, such watchdogs are effective only for transient fault. Since the fault in the Mars Pathfinder was a design (permanent) fault, restarting the system did not help.

References

[1] Mike Jones, email- Subject: What really happened on Mars?, Sunday, December 07, 1997.
http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/Mars_Pathfinder.html

[2] Glenn E. Reeves, email-Subject: Re: [Fwd: FW: What really happened on Mars?], Monday, December 15, 1997.
http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/authoritative_account.html

[3] Tom Durkin, What the Media Couldn't Tell You About Mars Pathfinder, Robot Science and Technology Magazine, Issue 1, 1998.

http://people.cis.ksu.edu/~hatcliff/842/Docs/Course-Overview/pathfinder-robotmag.pdf

[4] N.J. Keeling, Director of Marketing, Northern Real Time Applications (NRTA), Real-Time Magazine, 99-4, 1999.
http://www.imd.uni-rostock.de/ma/gol/bsys/pdf/nrt.pdf

[5] L. Sha, R. Rajkumar, and J. P. Lehoczky. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. Comput.* 39, 9 (September 1990), 1175-1185.

[6] Mars Pathfinder, NASA Facts, National Aeronautics and Space Administration, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109.

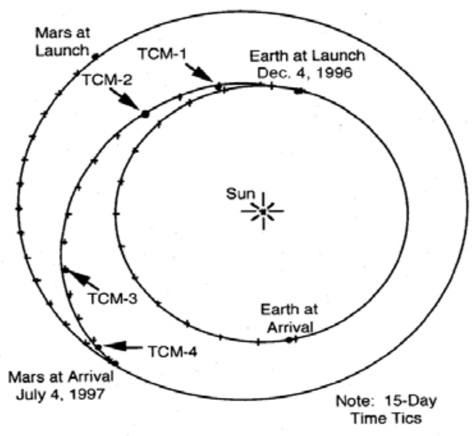
www.jpl.nasa.gov/news/fact_sheets/mpf.pdf

[7] Audsley, N.C.; Bate, I.J.; Burns, A.; , "Putting fixed priority scheduling theory into engineering practice for safety critical applications," IEEE *Real-Time Technology and Applications Symposium, 1996.*

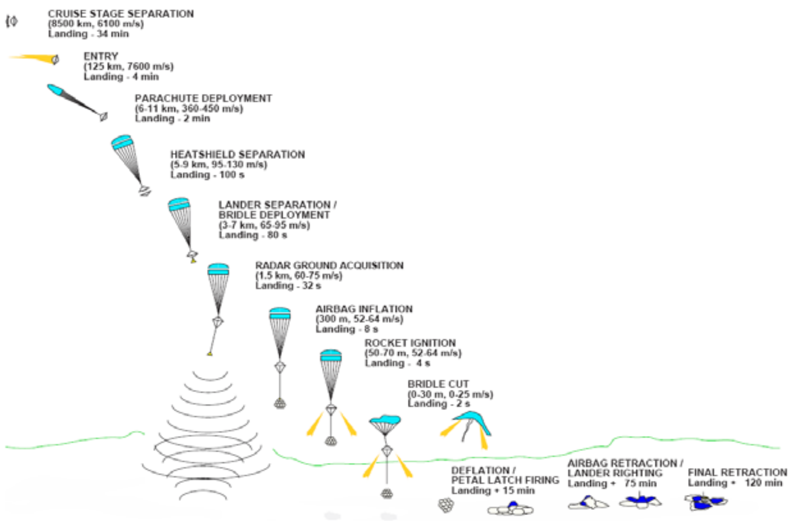
[8] Mars Pathfinder Official Website. http://www.nasa.gov/mission_pages/mars-pathfinder/index.html

APPENDIX

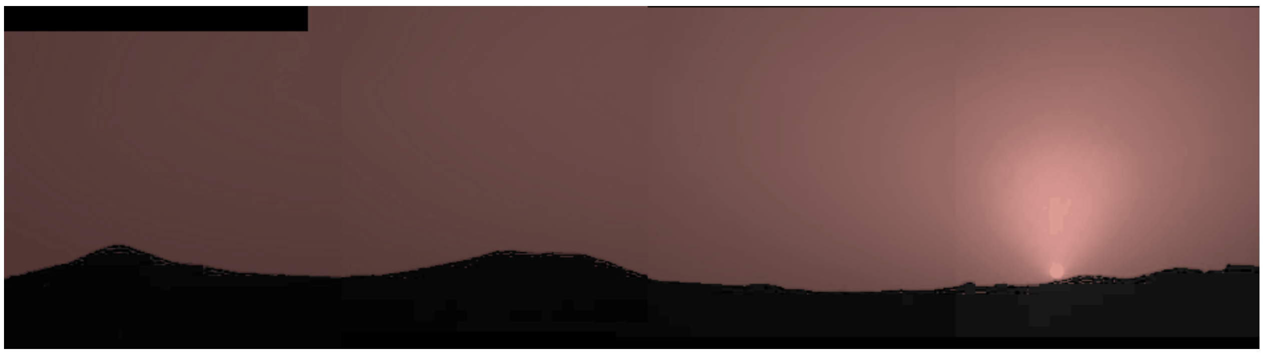
[Some pictures that I find really interesting]



Mars Pathfinder's Earth-Mars trajectory


CRUISE STAGE SEPARATION (8500 km, 6100 m/s) Landing - 34 min
ENTRY (125 km, 7600 m/s) Landing - 4 min
PARACHUTE DEPLOYMENT (6-11 km, 360-450 m/s) Landing - 2 min
HEATSHIELD SEPARATION (5-9 km, 95-130 m/s) Landing - 100 s
LANDER SEPARATION / BRIDLE DEPLOYMENT (3-7 km, 65-95 m/s) Landing - 80 s
RADAR GROUND ACQUISITION (1.5 km, 60-75 m/s) Landing - 32 s
AIRBAG INFLATION (300 m, 52-64 m/s) Landing - 8 s
ROCKET IGNITION (50-70 m, 52-64 m/s) Landing - 4 s
BRIDLE CUT (0-30 m, 0-25 m/s) Landing - 2 s
DEFLATION / PETAL LATCH FIRING Landing + 15 min
AIRBAG RETRACTION / LANDER RIGHTING Landing + 75 min
FINAL RETRACTION Landing + 120 min

Landing of Mars Pathfinder



The Best Sunset on Mars