

AD VMIC

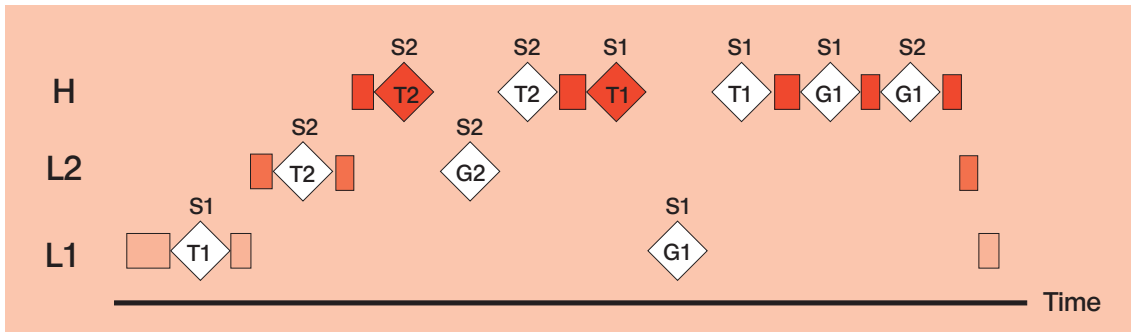


Figure 5.

locks. So the programmer may make malloc() calls but be unaware of the potential for long blocking times. In some cases calls to malloc() themselves are implicit.

Extended blocking of a high priority task is simple to introduce yet it is very difficult to find the longest total blocking time through testing (it depends upon a run of 'bad luck' in the phasing of the tasks in order to see worst-case scenario).

A second problem that may occur is deadlock as illustrated in Figure 6.

Deadlock can occur when H and L share two semaphores (S1 and S2) for mutual exclusion over two buffers. L locks S1 and is then pre-empted by H, which successfully locks S2 and then attempts to lock S1. At this point the priority of L is boosted and it resumes executing. L then tries to lock S1 and has to block (since H already holds S1). Now neither H nor L will ever be able to run again (in effect, the response times are infinitely long, not a good thing for a real-time system!). The tasks are in a fatal embrace - i.e. deadlock - and the application has failed. To be sure, deadlock can be avoided if the application is designed carefully with this in mind, but again, it is an easy mistake to make and unlikely to be encountered in testing.

A SOLUTION: PRIORITY CEILING PROTOCOL

What would be ideal is a solution to the priority inversion problem that also addressed deadlock and multiple blocking. This sounds like a tall order but such a mechanism exists: the Priority Ceiling Protocol (PCP). Although PCP is more complex than Priority Inheritance there is a variant of PCP (called the Immediate Inheritance PCP, or IIPCP) that can be implemented much more efficiently than even basic

semaphores yet has the same real-time behavior as PCP. With PCP a given task is blocked at most once, even when locking several semaphores. A side-effect of this property is that a system using PCP to lock semaphores is guaranteed to be deadlock free. The protocol requires that we know which tasks can access which semaphores: each semaphore has an attribute called the "Ceiling Priority" derived from this information. The ceiling of a semaphore is the priority of the highest priority task that can lock the semaphore. Ceilings typically need to be calculated off-line and in fact SSX5, the RTOS produced by my own company, is one of the few commercially available products to implement IIPCP. It is able to do this thanks to an offline application configuration tool.

The application of IIPCP to our simple example is shown in Figure 7.

The semaphore S has a ceiling of H because H is the highest priority task that can lock S. When L locks the semaphore its priority is raised to the ceiling of S, i.e. L runs with high priority. While L holds the semaphore, neither H nor M can be executing. This means that H cannot access the buffer while L holds the semaphore and so IIPCP maintains the functional behavior of regular semaphores. Once L unlocks S its priority is restored and H can begin executing (because H is now the highest priority ready task). When H finishes M can begin executing and only when M has finished will L complete.

CONCLUSION

The problems of priority inversion, blocking time and deadlock are all issues that face the developer of hard real-time applications. Observation of such problems often depends upon subtle race conditions and so

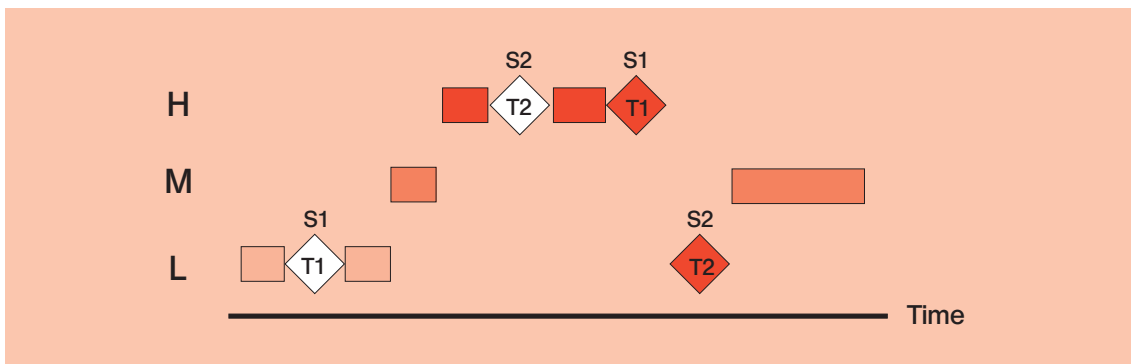


Figure 6.

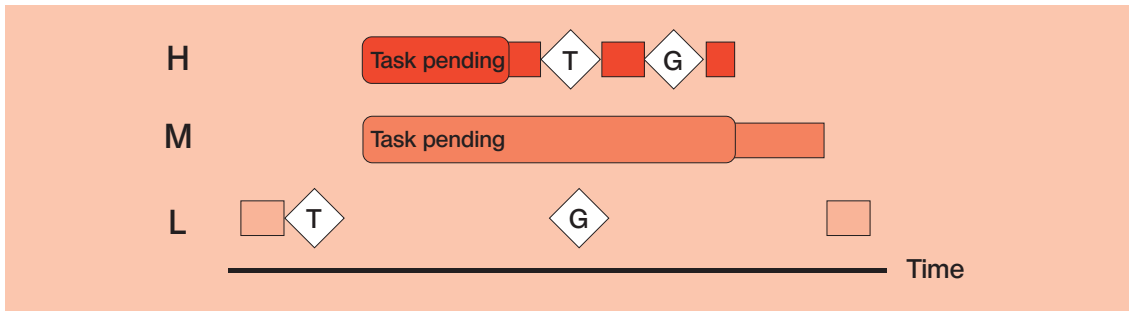


Figure 7.

these problems are almost impossible to encounter during testing. For example, the statistical likelihood of all tasks exhibiting their worst-case behaviour during merely a few hundred hours of testing in the lab is slim. However, given a couple of million hours of 'flying time' out in the field, the chances of manifestation increase somewhat. Indeed, in very high volume applications the total 'flying hours' can be more than a billion hours. Perhaps Murphy's Law applied to high volume real-time systems should be "If it can go wrong then after a billion hours it will go wrong." It is not just critical systems where meeting deadlines is vital. For example, if just 1% of drivers bring their automobiles to a workshop because of a false trigger of a dashboard warning light (itself caused by a software timing error) then total costs to the manufacturer will soon become a huge amount of money. With this in mind it is clear that

the rigor and determinism offered in part by the IIPC can be a major benefit to the designers of hard real-time embedded systems ■

Nick Keeling is a recent recruit to NRTA and fills the Director of Marketing role. He joined from Blue Wave Systems where he spent ten years at various stages on the software engineering career ladder before moving over to the marketing 'dark side' about three years ago. Initially recruited from college (where he studied computer science) by Plessey Telecommunications, Nick has a great deal of experience designing, writing, analyzing and debugging real-time software.

Priority inversion causes task response times to be extended, often far beyond their deadlines. Worse still, such deadline overruns typically occur intermittently and so escape detection during system testing. This is exactly what happened to the NASA JPL "Pathfinder" probe to Mars in July 1997.

The spacecraft was controlled by a single processor on a VME bus that also contained interface cards for the radio, a camera, and an interface to an on-board MIL-STD-1553 bus. The bus connected two parts of the spacecraft together, with the VME system containing interfaces to among other things the ASI/MET meteorological science instrument.

In the first few hours of operation on Mars there were problems where the spacecraft began experiencing resets. A reset re-initializes all hardware and software and terminates the current ground command activities and the remaining activities are postponed until the next day.

The resets were due to priority inversion. The JPL engineers selected a conventional RTOS to implement the control application in the main processor. A task with 125ms period but a short deadline - and so a high priority - was driving the on-board bus taking data from a low priority instrument task. This task collected data from the ASI/MET instruments and handing it over to the bus task. The two tasks communicated via a regular semaphore.

On Earth extensive testing was done but the problem

only manifested itself when the ASI/MET data was being collected at the same time as a high load due to medium priority tasks. Although a reset was seen in testing, the problem was not repeatable and could not be isolated. Because the problem was rare the system was deemed sufficiently reliable (the space business is inherently risky and there's no point engineering a limited-life space vehicle to a much higher reliability than the launch technology).

On Mars the task execution patterns were different and priority inversion occurred again. Medium priority tasks stopped the low priority instrument task from running which stopped the high priority bus task from running. When the bus task misses its deadline the system assumes a fault and resets itself. However, since the fault was a design fault - using conventional semaphores in a real-time system - re-starting the system just triggers the fault again and another reset.

Space systems are usually carefully designed to let Earth-based control gain access and upload new software, and so the JPL engineers were able to patch over the problem and upload new software using basic priority inheritance. Although priority inheritance is not a good algorithm, for Pathfinder it was sufficient in the specific use of semaphores to get the response time of the bus task short enough. The system worked OK for the rest of the mission, which was highly successful.

Side explanation 1. Priority inversion on Mars