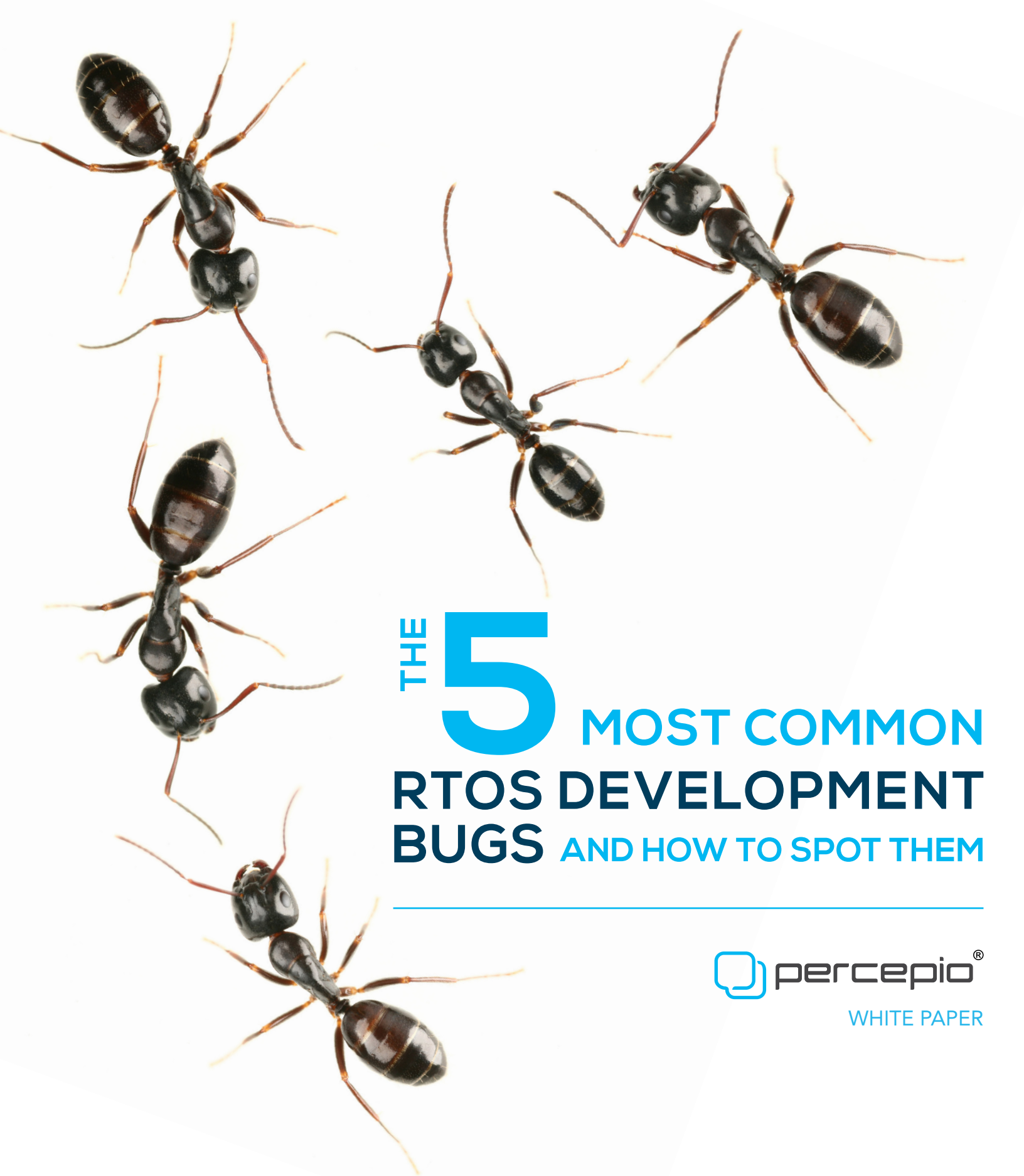


THE 5 MOST COMMON RTOS DEVELOPMENT BUGS AND HOW TO SPOT THEM

percepio®

WHITE PAPER

percepio®

WHITE PAPER

WHAT MAKES RTOS DEVELOPMENT SO HARD?

When using a real-time operating system (RTOS), embedded developers work at a higher level of abstraction, similar to the shift from programming in assembly to C, which can make it easier to design complex applications. But while an RTOS reduces the complexity of the application source code, it does not reduce the inherent complexity of the application itself. That can make the application difficult to validate and debug.

The main job of an RTOS is to provide multitasking, where several program modules—or tasks—can execute in parallel to achieve a common goal. The RTOS creates an illusion of parallel execution by rapidly switching between tasks. Developers can retain some control over elements such as RTOS task priorities, which in turn enables deterministic real-time behavior, but they also lose some control over the finer details. For instance, the program flow is no longer apparent from the source code, since the RTOS decides which task to execute at any given moment.

What makes RTOS-based development so difficult is that RTOS tasks are not isolated entities, but have dependencies that may delay task execution in unexpected ways. Subtle coding choices can result in elusive errors or performance issues in the final product. A set of seemingly simple RTOS tasks can result in surprisingly complex runtime behavior when executing together as a system; there can be countless execution scenarios that are impossible to fully cover by testing or code reviews.

The challenge for developers is that when development moved up the abstraction ladder—from assembly code and super loops to C and an RTOS with scheduling—debugging tools for real-time application development didn't keep up. While standard debugging tools are still mostly focused on breakpoints and single-stepping through source code, RTOS trace and visualization gives developers full insight into the application's behavior at system level.

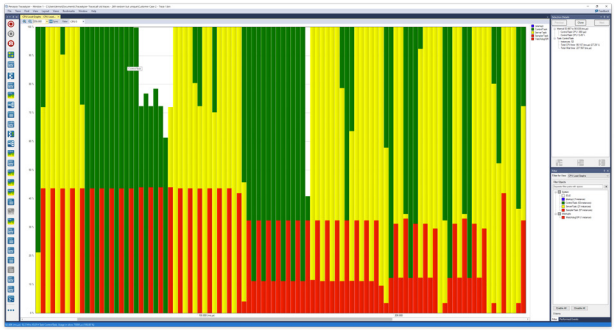
What makes RTOS-based development so difficult is that RTOS tasks are not isolated entities, but have dependencies that may delay task execution in unexpected ways.

We've identified five of the most common real-time application bugs. Using RTOS trace and visualization, we show how developers can work at a higher abstraction level and capture timing and synchronization aspects of their application that traditional debuggers can't see.

1. CPU Starvation

Symptoms: • Tasks run slowly or not at all

In embedded systems using multitasking, some tasks may become starved of CPU time. A common reason is that task priorities are wrong. Higher-priority tasks always execute before lower-priority tasks, but if the former use too much CPU time, the latter may receive too little CPU time to do their jobs. This is known as task starvation.



Starvation occurs when CPU load approaches 100%.

A naïve fix would be to raise priorities of the affected tasks, but this ultimately renders prioritization useless. Instead, reserve high priority for tasks that are predictable, execute in a recurring pattern, and have a short execution time compared to the recurring interval. High-priority, time-critical tasks that consume a lot of CPU may need to be split into several tasks. This isolates the time-critical elements in a small, high-priority task which then alerts a medium- or low-priority task to do the bulk of the processing.

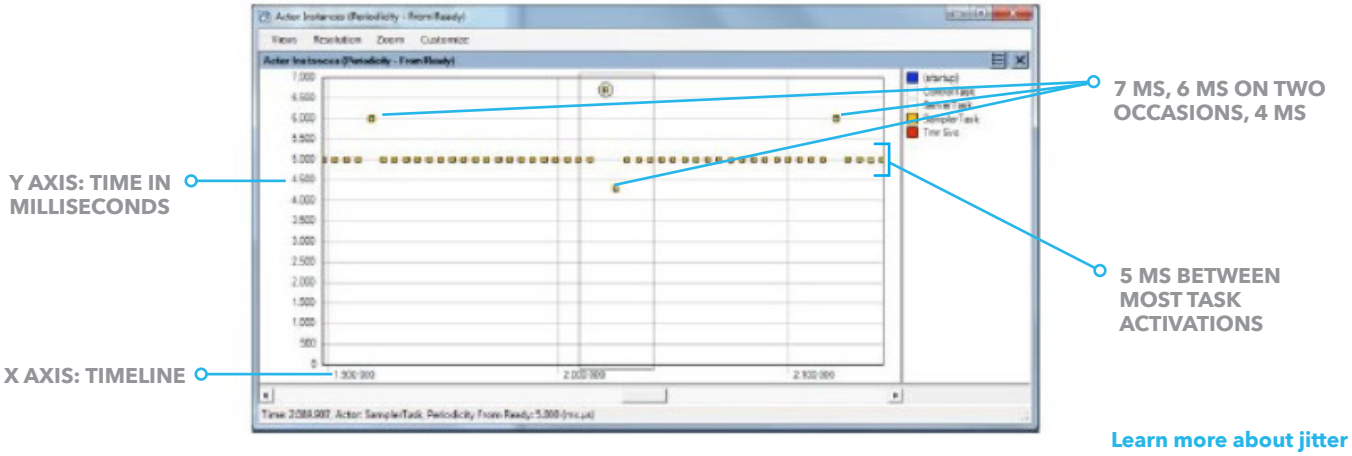
Learn more about CPU starvation

2. Jitter

Symptoms: • Poor control performance
• Intermittent data loss or malfunction

When a task is supposed to execute at regular intervals, such as to adjust motor speed every 5 milliseconds, the system is sensitive to random delays, also known as jitter. When a task experiences jitter, it must wait more than its intended sleep time before the next activation. While minor jitter is hard to avoid and may not be a problem, excessive jitter is a different story.

The visible symptoms of excessive jitter can be poor control performance and intermittent data loss. The cause can be CPU starvation, which is fixed by getting the task priorities right and avoiding long-running high-priority tasks, but there are several other possible causes, such as suboptimal RTOS configuration. Developers should also look at the RTOS tick rate (how often the RTOS timer interrupt occurs). Ideally, the time between two ticks should be much shorter than the period time of the most frequent tasks in the system.

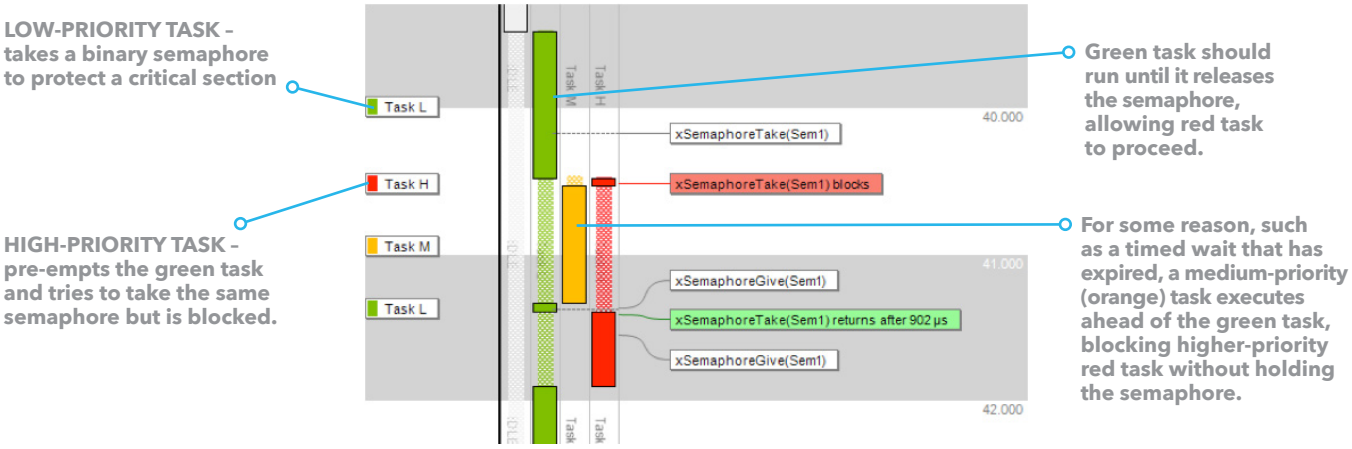


Learn more about jitter

3. Priority Inversion

Symptoms:
- System momentarily stops responding
- System crashes randomly

An RTOS with a fixed-priority scheduler should schedule high-priority tasks ahead of those of lower priority. However, the inverse can sometimes occur and the lower-priority tasks get run first. This condition is called priority inversion, and it can occur in conjunction with a synchronization object such as a message queue or a mutex. Priority inversion can be mitigated, provided the underlying RTOS supports so-called priority inheritance, but completely eliminating it requires careful application design. The trick is to identify that it is occurring.

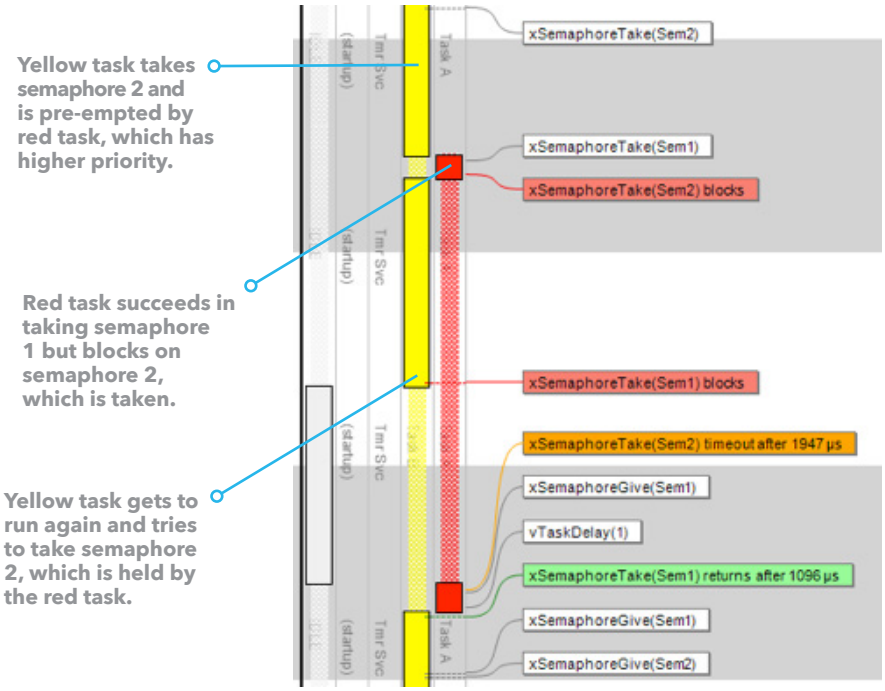


Learn more about priority inversion

4. Deadlock

Symptoms:
- Tasks suddenly stop executing, although no higher priority tasks are running

A deadlock occurs when two or more application tasks are blocked waiting for each other, leaving both tasks unable to proceed. Generally speaking, a deadlock can be a fatal condition, so identification is critical.



- The two tasks block each other and neither of them can continue execution.
- Deadlock is unsatisfactorily resolved by a timeout (the orange event) in the red task. It could not acquire both semaphores, so it released the one it held and went back to sleep, which allowed yellow task to take semaphore 1, run to completion, and release both semaphores.

Learn more about deadlock

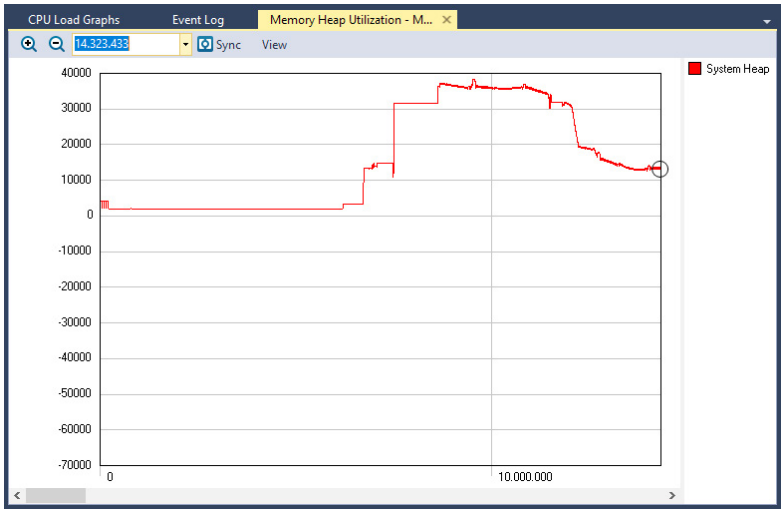
5. Memory Leaks

Symptoms: • Memory allocation fails during operation

Dynamic memory allocation is typically not recommended for embedded software, but is sometimes motivated for various reasons (right or wrong). Moreover, third-party software libraries or external development teams may use it without the primary developer's knowledge. The catch with dynamic memory allocation is that every allocated block of memory must be freed once the memory block is no longer in use. Any missed case will cause a memory leak and the application will eventually run out of memory.

A memory leak is especially dangerous if it only occurs occasionally as a slow memory leak is easily missed during functional testing but can cause critical errors later in a deployed unit. Given the long-running nature of many embedded systems, combined with potentially deadly or spectacular failures of safety-critical systems, it is important to identify and fix memory leak bugs early. With Tracealyzer, developers can monitor RTOS calls for dynamic memory allocation and highlight suspected memory leaks.

The Memory Heap Utilization view shows the amount of currently allocated memory over time.



Learn more about memory leaks

How to Find–and Fix–RTOS Development Bugs

These are just the five most common RTOS development bugs, but there can be countless others hiding in your code. You can try to guess your way through debugging, trying one thing after another to get the application to run properly. But to solve the problem, you need to understand the specific sequence of software events that caused it, including the interactions between the application and the RTOS. Traditional debugging tools cannot offer this insight.

RTOS trace visualization, which can be thought of as a slow-motion video of the application's internals, provides confidence that an RTOS application runs as designed and can be the fastest way to detect and correct bugs.

FIND YOUR BUGS FAST!

Download a free, fully functional evaluation of Tracealyzer at https://percepio.com/download.

The installer includes a pre-recorded demo trace so you can quickly explore all 30+ interconnected views. It is available for leading RTOSes and Linux systems and supports most 32-bit and 64-bit processors out-of-the-box. The Tracealyzer tool runs on Windows and Linux hosts.

percepio®

percepio.com