Introduction to Real-Time Operating Systems: Part 1

Call: +44 (0)23 8098 8890
E-mail: sales@itdev.co.uk (mailto:sales@itdev.co.uk)

Posted 4th October 2017, By James H (/company/about/team/james-h)

Introduction

In the embedded world, the use of a Real-Time Operating System (RTOS) is commonplace, and with the advent of the Internet of Things (IoT) they are becoming more so. At the same time, the RTOS market is also becoming more fragmented with many different options to choose from, as if it wasn't fragmented enough already!

You might be deciding to use an RTOS for the first time, or perhaps you are thinking of moving to a new one. This may be especially true when considering moving from a commercial vendor to an open source collaboration, such as FreeRTOS[1]. You might also be considering one of the newer players that are designed specifically for the extremely low power and low resource IoT world, like Contiki, the Zephyr Project[2] or RIOT OS[3].

If you're new to the RTOS world and are taking your first steps, you will probably want to answer the fundamental question of what exactly is an RTOS?

What is an RTOS?

An Operating System (OS) is in charge of managing the resources of your computer. It will put tasks on the CPU and take tasks off the CPU, giving each task the illusion that it owns the entire processor. It will implement some kind of scheduling policy, which dictates when a task should be given time to run on the CPU and for how long. It will also manage the memory in the system and the various bits of hardware attached.

A general OS, such as Windows or Linux, is normally designed for throughput and fairness, with a reasonably fast response on average. The OS wants to get as much work done as possible during any period of time. It is designed to be fair, so that most tasks will get at least some time on the CPU, which means that even if things run slowly, you can still use your text editor and listen to your favourite music at the same time. The time taken to respond to events, such as your sound card requesting more audio data, will on average be quick. If the system is busy, however, it may or may not handle the event in a timely manner and the worst case here is that you get a brief "glitch" in your music.

But what if that "glitch" in your music was actually something more serious? What if the event was "a child has stepped out in front of a driverless car"? In this case we want the OS to service this event as soon as possible, 100% reliably. We'd certainly be happy for our music to be interrupted to service this event! This use case requires an RTOS, which will guarantee to service the event within a defined, minimal time.

These are two extremes and there are clearly cases in the middle. Perhaps the event is a machine exceeding a safety limit and the result is just a broken machine. Either way, in all of these scenarios, if the event is not serviced within its deadline, something "bad" happens. In RTOS parlance we call the ability to execute the right thing at the right time "correctness".

An RTOS guarantees something that a more general OS won't: it is deterministic and therefore predictable, which means that with a correctly designed system it will reliably meet its deadlines. It favours responsiveness over throughput and correctness over fairness. This becomes especially important in busy systems, where no matter how busy it gets, the OS has to guarantee that it will respond to critical events.

This can be summed up nicely with the following Jan Altenberg quote from his Linux Foundation talk (emphasis added)[4]:

> *"Real-time is not about fast execution or performance … It's basically about **determinism** and **timing guarantees.** Real-time gives you a guarantee that something will execute within a given time frame. You don't want to be as fast as possible, but as **fast as specified**."*

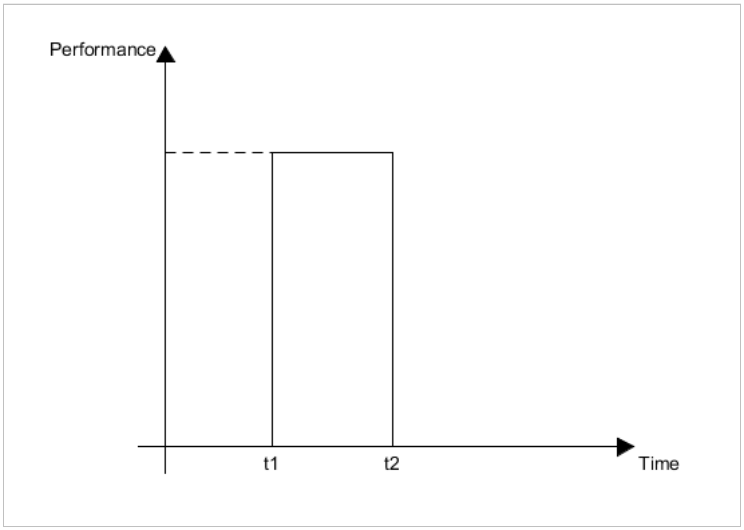
What is "soft" and "hard" real-time?

You often hear the terms "soft" and "hard" real-time used. In his talk, Jan Altenburg's says: "…please forget about this word"[4]. He gives the example of your wife telling you that she is kind-of pregnant. Well, she either is or she isn't, and likewise your OS is either real-time or it isn't.

What is “soft” or “hard”, however, is the scenario in which the RTOS is being used. That is, it is not the RTOS itself, but its use case. If failure to meet a deadline would lead to someone being run over, this could not be tolerated under any circumstances. The system must hit 100% of the deadlines 100% of the time. This is a hard real-time use case.

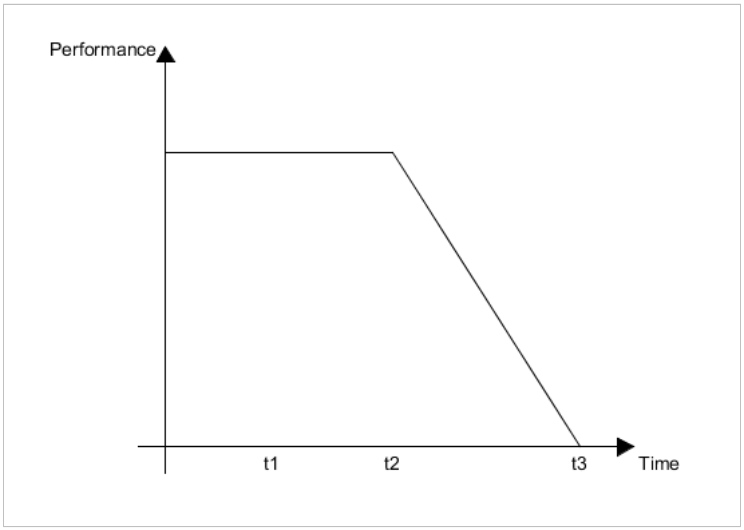
If the use case requirement was that the RTOS must hit 100% of its deadlines 95% of the time, we would call this a “soft” real-time constraint. For example, if the only consequence of missing the deadline is a broken machine, and no broken people, we might be willing to tolerate some margin for error. But then again, we might not. So what is “hard” or “soft” real-time can be somewhat subjective.

The figure below shows another way of describing a hard real-time system.



The system has a step function in its performance: it is all-or-nothing, which represents our requirement that the system hit 100% of its deadlines 100% of the time. If it can perform a job, be it a periodically scheduled job or a response to an event, all is fine, but if it cannot, then the result is complete performance failure. For periodic tasks an early job output might be as unacceptable as a late output, and this is what the solid line represents. The dotted line represents a system that can accept an early output. When considering response to asynchronous events, we normally consider a step rather than a notch (use the dotted line).

In contrast, the performance of a soft real-time system can be represented using the figure below.



The graph represents the concept that performance can degrade whilst still providing an acceptable output.

So, these are the basics of what a real-time OS is and why you may need one. The next article in this series will discuss the methods an RTOS uses to provide determinism, predictability and timing guarantees.

To receive an update when the next blog in this series is available please follow us on LinkedIn (http://www.linkedin.com/company/itdev/), Facebook (https://www.facebook.com/itdevltd)or Twitter (https://twitter.com/itdevltd).

References

1. http://www.freertos.org/ (http://www.freertos.org/)
2. https://www.zephyrproject.org/ (https://www.zephyrproject.org/)
3. https://riot-os.org/ (https://riot-os.org/)

4. J. Altenberg, “Introduction to Realtime Linux,” 04 04 2017. [Online]. Available: https://www.youtube.com/watch?v=BKkX9WASfpI (https://www.youtube.com/watch?v=BKkX9WASfpI).

Attributes

- Photo: "A Google self-driving car at the intersection of Junction Ave and North Rengstorff Ave in Mountain View (https://commons.wikimedia.org/wiki/Category:Google_driverless_car#/media/File:Google_driverless_car_at_intersection.gk.jpg)" by Grendelkhan (https://commons.wikimedia.org/wiki/User:Grendelkhan) is licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/)

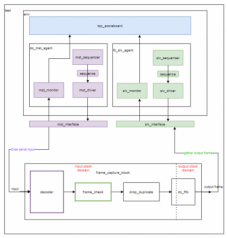
< prev next > (/blog/introduction-to-real-time-operating-systems-part-2)

Latest Blog Posts

(/blog/reflection-2020)

A Reflection on 2020

(/blog/reflection-2020)
Posted 29th January 2021, By Jon O
(/company/about/team/jon-o)
Having started 2021 much as we ended 2020, we thought it would be good to look back to before the pandemic and reflect on how the year turned out against ...more
(/blog/reflection-2020)



(/blog/universal-verification-methodology-design-reuse)