RTOS and Linux Are Defining In-Vehicle Computing – Together

As the industry moves from function-specific hardware and software to centralized in-vehicle computing and software-defined vehicles, it is clear that Linux will play a significant role in future vehicle platforms. Linux has a long and proven history in other industries and is extremely flexible, which makes it a natural choice for automotive applications.

But as OEMs introduce applications with specialized requirements — especially around functional safety for fully or partially automated driving — there is a need for a real-time operating system (RTOS) that is optimized for deterministic latency in safety-critical use cases.

In evaluations of potential future in-vehicle software architectures, it is important to understand the relative strengths of Linux and RTOSes and how they complement each other in this changing landscape.

KEY DIFFERENCES

Not all applications have the same timing requirements, and the differences among those applications necessitate different approaches to the operating systems that support them.

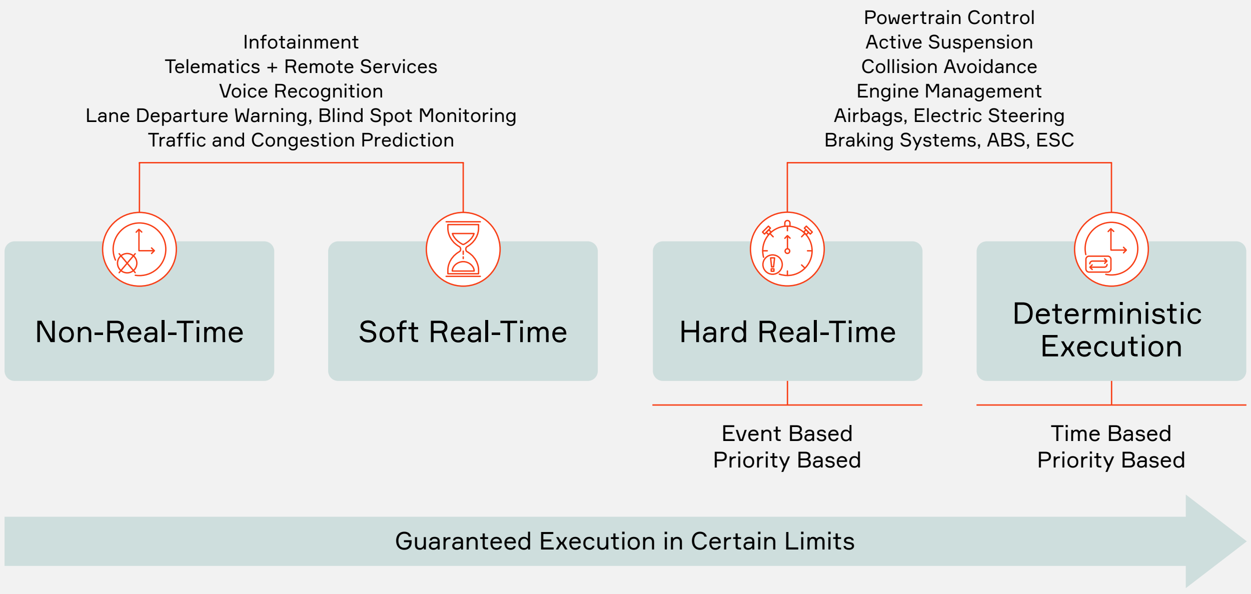
Some applications do not need to guarantee that they will complete their tasks within a specific time frame. The application should run as fast as possible, but minor delays are acceptable. Automotive examples include telematics, remote services and traffic prediction. These are non-real-time (non-RT) applications.

With other applications, a delay might result in a less-than-optimal user experience. In the automotive world, these applications include some functions of advanced driver-assistance systems, such as lane-departure warning and blind-spot monitoring. They deliver important, timely information but do not require an immediate response by the driver. The OS needs to enable the scheduling of tasks but does not need to guarantee predictable performance or the protection of the tasks from interference from other applications. These are considered “soft” RT applications.

But for other applications, a delay could have serious consequences. Examples include software functions that control a vehicle’s movement, such as automatic emergency braking, lane changing, and autonomous vehicle maneuvering to avoid collisions. These are called “hard” RT applications, and they are often safety-critical.

What Is Real-Time?

A real-time system can guarantee that tasks consistently execute in a specific time constraint. Determinism is the characteristic that describes how consistently a system executes tasks within a time constraint. In a deterministic system, the sequence of executing tasks is always the same.



The contenders

As a general-purpose operating system, without modifications, Linux is suitable only for applications where delays can be tolerated.

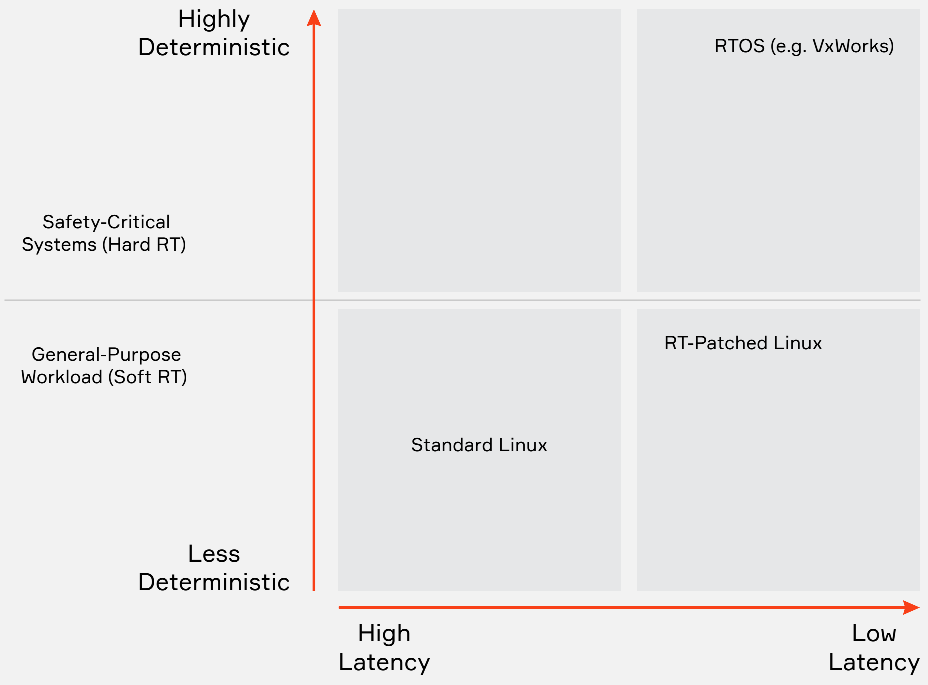
Linux has proliferated across enterprises and industries partly due to its low cost and vast ecosystem of open-source developers and tools. OEMs have adopted or evaluated Linux for a wide range of vehicle applications, and it has already proven ideal for some.

In contrast, an RTOS is designed exclusively for real-time applications, with features that guarantee that a given input will produce the same outcome on time, every time. The RTOS enables RT applications to reserve OS resources at specific times and use them to complete tasks by strict deadlines. It ensures that events take place at the correct time, not just as soon as possible. These characteristics are indispensable for hard RT applications.

In addition, RTOSes have fewer lines of code compared with general-purpose OSes. This gives OEMs several benefits. From a security perspective, less source code translates into less surface area for attackers to target, reducing risk and the work required to mitigate it. That said, it is still important to use an RTOS designed with a secure development lifecycle. In terms of safety, a smaller, less complex OS takes less effort to certify for functional safety standards. These advantages can help OEMs bring a vehicle to market faster.

Right Tools for the Job

Different variations of operating systems are appropriate for different applications within a vehicle.



Running RT workloads on an RTOS can also reduce costs. With less source code, an RTOS has lower CPU and memory requirements, which can translate into a lower hardware bill of materials. If an RTOS has been precertified for safety, this saves OEMs the considerable cost of certifying it themselves. In addition, RTOSes — especially their source code and APIs — change more slowly, reducing maintenance and security costs, preserving an OEM's investment, and reducing the need to re-certify the operating system due to changes.

MODIFYING LINUX FOR SOFT RT APPLICATIONS

While RTOSes are the clear choice for hard RT applications, several Linux patches now make it possible to run soft RT applications on versions of Linux.

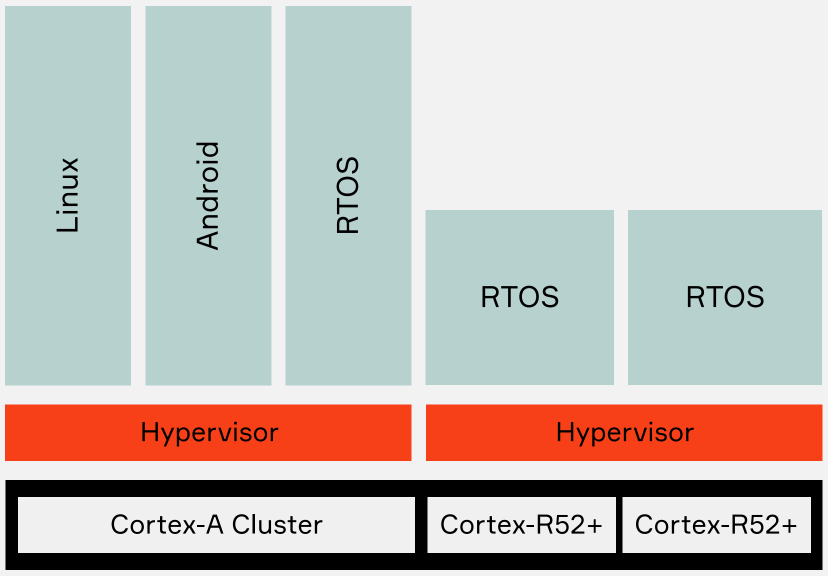
These allow OEMs to shift some real-time functions from an RTOS to Linux, but doing so may affect performance and manageability in some ways, including the following:

Timer slack

A patch to the Linux kernel increases the resolution of kernel timers by making it possible to wake up the CPU more frequently. This allows for nanosecond precision, which some RT application threads require. However, waking up the CPU more often may increase power consumption and reduce CPU efficiency. The patch also enables more accurate timing in RT threads but does it by using timer slack, a technique that delays events in threads with normal timing policy. This can cause more jitter in asynchronous applications.

Mix and Match

Various operating systems can coexist side-by-side on the same hardware, but they should be logically separated via a hypervisor, and on different cores as needed.



Process suspension

Because RT systems are engineered to execute the current top-priority task on deadline, it is often necessary to use a custom scheduling method rather than the Linux kernel's standard method. If an RT process is trying to use 100 percent of a CPU, the kernel's standard scheduling method suspends the process for 50ms every second. A patch for RT support deactivates the suspension mechanism, but this feature is so essential to the kernel's scheduling system that removing it may cause instability or failure.

There is industry momentum behind making Linux more applicable for safety-critical applications, including an initiative called Enabling Linux in Safety Applications (ELISA), which is working with member companies, certification authorities and standardization bodies to establish how Linux can be used as a component in safety-critical systems.

HOW RTOS AND LINUX CAN COEXIST

By running all hard-RT vehicle applications on an RTOS and devoting Linux to non-RT and soft-RT workloads, OEMs enjoy the inherent benefits of the RTOS while minimizing the complexity of Linux. This is likely to reduce software development and maintenance costs and improve security.

Centralizing onboard processing gives OEMs an ideal opportunity to run Linux and an RTOS side by side. Shared computing platforms can host multiple OSes running applications with different levels of criticality by using two key technologies: software virtualization through a hypervisor, and hardware isolation on separate processor cores.

Software virtualization

Soft-RT applications can run on the same set of cores as Linux and other OSes in a virtualized system that allows the flexible use of resources. (See example in sidebar.) Each OS runs in its own virtual machine, with a hypervisor (such as Wind River Helix Virtualization Platform) managing common memory, computing resources and processor cores across a cluster of ARM Cortex-A performance cores. Within applications, containerization may provide a further layer of virtualization for more flexible development and maintenance.

Hardware isolation

Safety-critical applications can run on one or more RTOSes and coexist on the same system-

USE CASE: AUTOMATIC EMERGENCY BRAKING

One example of how Linux and an RTOS might work together in a vehicle is the implementation of automatic emergency braking, a hard-RT application.

As a vehicle approaches an object, its cameras, radars, lidars and inertial sensors send data to zone controllers, which forward the data to the central vehicle controller. Software running on Linux on that device uses computer vision and machine learning algorithms to fuse the sensor inputs and put them in context.

The Linux software sends that data to the braking regulator, a small application connected to the braking system. The braking regulator runs on a safety-certified RTOS to ensure that it can apply the brakes at precisely the right time. By applying an algorithm to the contextualized sensor data, the braking regulator constantly calculates the risk of a crash and applies the brakes when necessary.

on-a-chip as non-safety-critical applications, but isolated on a “safety island” using Cortex-R cores — as long as the RTOSes, the hypervisor and the cores are rated to support ASIL-D risk levels. The safety island may contain multiple OSes virtualized through a hypervisor. Isolation on the safety island prevents failures in less-critical applications from crashing applications that affect life safety.

Evolving with hardware architectures

These configurations can be implemented in current and future vehicle architectures with varying degrees of centralization in domain architectures, zonal architectures or a combination. In all cases, middleware is the essential glue that enables communication and coordination among all containers, applications and OSes in the vehicle. Multiple OSes will run on many cores under one software stack, with cores assigned to different OSes as necessary.

EMBRACING THE POSSIBILITIES

The trend away from purpose-built electronic control units and software in favor of software-defined vehicles has opened up space in vehicles for a variety of OSes, including Linux. With high performance and a large development ecosystem, Linux complements deterministic, safety-certified RTOSes. OEMs that understand the benefits and limitations of each can deploy them on centralized hardware with unprecedented flexibility, improving future vehicles and making development faster and more cost-effective.

In 2022, Aptiv acquired Wind River, which specializes in providing tools that support robust, reliable and secure embedded solutions, including Wind River Linux and the Wind River VxWorks RTOS. Both offer unique strengths for automotive applications and draw from a long pedigree of supporting safety-critical operations in other industries.

ABOUT THE AUTHORS

Florian Baumann
Senior Director, Software, Advanced Vehicle Architecture, Aptiv

Florian Baumann leads a team focused on developing cutting-edge technologies to solve some of the most demanding problems of the automotive sector. With experience in machine learning, software development, DevOps and cloud architectures, Florian is applying his lifelong obsession with technology to create Aptiv's next-generation in-vehicle software platform, reducing its complexity while providing a seamless developer experience.

Rob Woolley
Principal Technologist, Wind River

Rob Woolley is a Principal Technologist at Wind River in the CTO Office. He has 25 years of experience with both enterprise and embedded Linux and more than 15 years with the VxWorks RTOS. He is actively involved with the open source community as the maintainer of the Robot Operating System framework for OpenEmbedded and participates in Zephyr RTOS and ELISA from the Linux Foundation. His current focus is on using cloud-native technologies to orchestrate workloads on edge devices, including software-defined vehicles.

LEARN MORE AT APTIV.COM/MWD →