AUTOSAR Classic Under POSIX Operating Systems

Perfect Interplay

POSIX operating systems are the foundation for new, dynamic, in-vehicle software applications. However, they lack automotive functions such as diagnostics, network management and SOME/IP. AUTOSAR Adaptive is a standard available to automotive OEMs which supports these tasks in new development projects. But how can the use of a POSIX operating system succeed if the vehicle platform is still based on AUTOSAR Classic?

ECU projects based on the AUTOSAR Classic software standard have so far been closed, static systems with subfunctions whose interactions with one another are fixed. POSIX operating systems such as Linux now offer the option of loading and executing software dynamically. These properties are an important prerequisite for new functions such as autonomous driving. However, not all automotive OEMs have made the change to AUTOSAR Adaptive yet. If a supplier is only provided with AUTOSAR Classic modules, the ECU developer is faced with a problem. How is it possible to operate proven AUTOSAR basic software such as diagnostic protocols or software components developed for it in a POSIX system? After all, the entire AUTOSAR Classic specification is based on a completely different type of operating system and, above all, on a static configuration process. Actually, it is entirely possible to use both types of ECU software simultaneously if certain properties are considered. But what needs to be considered exactly?

AUTOSAR basic software (BSW) is essentially a collection of C functions. It is no great challenge to operate these as a separate POSIX process (**Figure 1**). It does not get exciting until the system limits of the basic software are encountered. These limits are set by the peripheral hardware. Those wanting to extend these limits should observe the ground rule of making practically no changes in the Linux kernel space. However, an additional driver for a special hardware component is certainly permissible. Existing kernel components, on the other hand, should remain unchanged. Otherwise, even a normal security patch might lead to problems.

The example of a simple ECU project illustrates an actual implementation. Let us assume that Linux is to be used as the operating system for the ECU. The project specified an operating hours counter which is read out over the diagnostic interface. Practice-proven modules based on

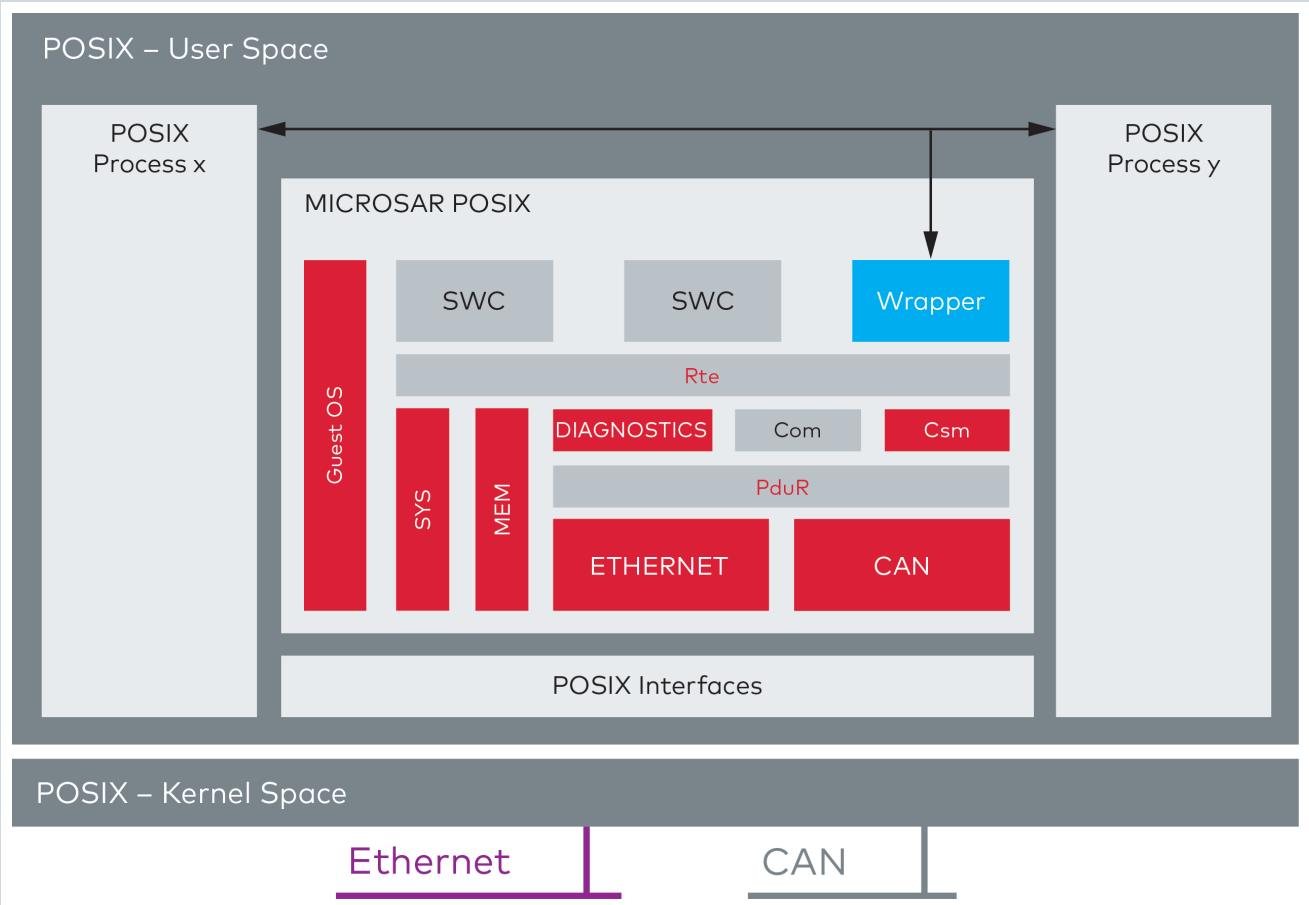

Figure 1: MICROSAR basic software from Vector as a POSIX process.

AUTOSAR Classic are ideal for diagnostic requests. What special considerations are there for such a configuration?

Ethernet as the Communications Interface

First, it is useful to look at the communications interface. Most of the ECUs relevant to this approach are Ethernet-based. Usually, the POSIX operating system provides the driver for managing the hardware controller and the software for the higher-level TCP/IP protocol. Now, the AUTOSAR Classic application is no longer connected to the network via the AUTOSAR driver, rather it utilizes BSD sockets instead. The advantage: The TCP/IP interface is available to all other Linux processes in its usual form. Applications with TCP/IP communications remain unchanged. This approach suffices for most applications. However, limits are reached if additional properties beyond a pure TCP/IP data stream are required, such as a time stamp (TSyn) or prioritization (QoS). These are only possible when project-specific solutions are implemented such as interventions into the drivers of the kernel space.

The socket approach is also recommended for communication over CAN. In this case, the CAN basic software, which consists of an interface and a transport protocol, is located on the socket instead of the driver. However, filtering of the received messages in the application software is then necessary. Since a dynamic system is required in this example, it is not possible to configure the driver with a pre-specified communications matrix. Unlike a pure embedded system, it is not possible to implement an Interrupt Mode or use FullCAN objects with this approach. Nonetheless, this generally does not cause any problems in practice.

State Manager and Watchdog

In AUTOSAR Classic, network management controls the switching between normal operation and the bus idle state. This procedure can still be used with a POSIX operating system. However, the state change is only valid within this special process. That is, it is impossible to rule out a situation in which other processes continue to be active on the CAN bus. For this reason, the state managers of system services in the AUTOSAR basic software (SYS modules) assume just limited significance. They are indeed useful for controlling the modules of the AUTOSAR process, but they do not affect other processes that are running in parallel. Therefore, an additional central control process is needed

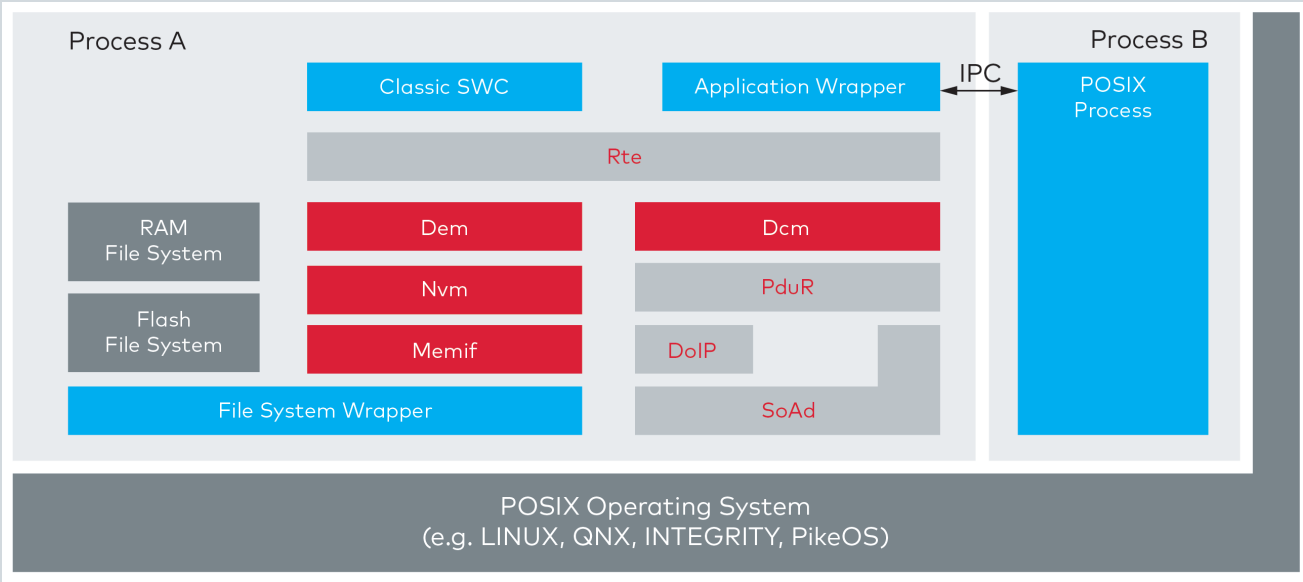

Figure 2: Classic diagnostics in a POSIX environment.

which monitors the system start and ECU shut-off. These considerations also apply to the watchdogs. An AUTOSAR watchdog manager is designed for use in a pure Classic system. It does not work under Linux, because watchdogs are not allowed to shut off a Linux system. Under Linux, a watchdog manager can only recommend restarting a Classic process. This restriction cannot be circumvented by controlling a hardware watchdog. The task is therefore to monitor the state of the AUTOSAR process and, in case of error, have the POSIX operating system initiate a restart of the process.

Handling of Diagnostics and the RTE

Implementation of a diagnostic function affects practically every ECU. AUTOSAR Classic defines suitable software modules for this. The advantage in a POSIX scenario: The type of operating system used is irrelevant for these diagnostic modules. However, this does not apply to such cases as providing nonvolatile memory to an error memory. In the above example of an operating hours counter, instead of accessing the AUTOSAR hardware driver, the file system of the POSIX operating system must be accessed. The software integrator creates a wrapper for this purpose, which maps the AUTOSAR interface to functions of the file system (**Figure 2**).

The actual application in AUTOSAR Classic is located on top of the upper BSW module layers such as the communication module (COM) or the Diagnostic Communication Manager (DCM). If another POSIX process provides or processes the data, a suitable wrapper is necessary, which connects the AUTOSAR interfaces to a mechanism for interprocess communication (IPC) in the host operating system. Naturally, it is also possible to interface AUTOSAR software components to the basic software via an AUTOSAR runtime environment (RTE).

However, one thing must be considered here: In contrast to other modules, the RTE makes very intensive use of the system services of the AUTOSAR operating system. An RTE can therefore only be used in combination with an AUTOSAR-OS.

It is not possible to operate two operating systems concurrently on the same kernel. They would compete and take resources from one another. One way to solve this would be to extend the desired POSIX operating system to include a virtual machine. This machine would suggest control of computing time and hardware to the AUTOSAR operating system. A "lightweight" alternative to this would be to use a specially developed variant of an AUTOSAR operating system. This variant of an AUTOSAR OS, known as a Guest-OS, does not access the hardware directly; rather it utilizes the services of the host operating system. The Vector basic software solution MICROSAR POSIX contains such a Guest-OS. This eliminates the need for extensions or other modifications of the Host-OS, which is a great advantage in practice.

Limitations of the Guest-OS

The Guest-OS offers all of the important system services contained in the AUTOSAR specification. However, there are limitations compared to a system in which an AUTOSAR operating system has full control over the processor. For instance, only scalability class SC1 is supported. A Guest-OS does not offer the runtime and memory protec-

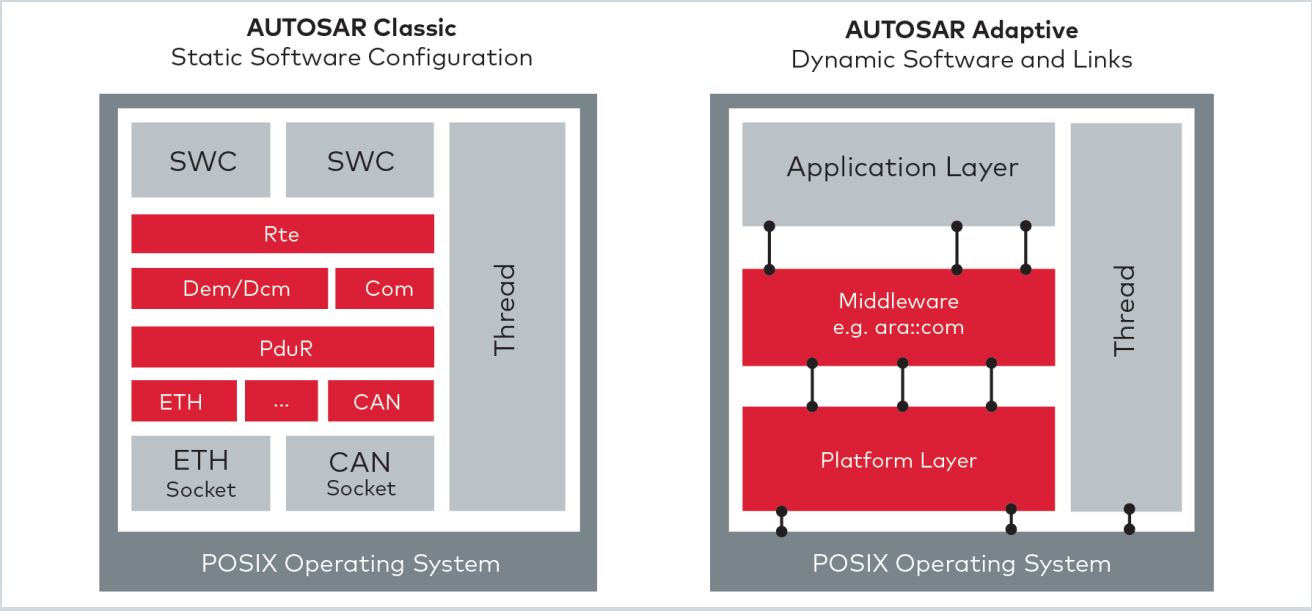

Figure 3: Comparison of a POSIX operating system with AUTOSAR Classic and one with AUTOSAR Adaptive.

tion defined in scalability classes SC2 to SC4, because it lacks direct control over the processor being used.
This also applies to the task of interrupt management, which is clearly a responsibility of the host operating system. System calls of the AUTOSAR operating system such as DisableAllInterrupts() / EnableAllInterrupts() would lead to serious conflicts with the host operating system if they were to actually disable or enable interrupts. In a pure AUTOSAR environment, this command pair is often used to safeguard read and write processes in memory areas shared by multiple tasks. Therefore, the Guest-OS must simulate this interrupt disabling internally by other mechanisms. For example, an interrupt disabling by DisableAllInterrupts() causes the Guest-OS to not permit thread changes, i.e., task changes. This achieves the goal of data consistency. It is therefore unnecessary to make changes to the code of other AUTOSAR components.

Another limitation applies to time behavior. Alarms can be defined as usual, and they can be used to activate tasks. In executing these tasks, however, the host system decides on process scheduling and thread scheduling indirectly. It is indeed possible to set an alarm so that it activates a task every 5 ms. However, this is of little use if the Host-OS only provides the AUTOSAR process with a processing slot every 10 ms. This is not really problematic. Rather it is more a question of how to configure the host system. In the end, the primary focus of a POSIX-based ECU is less on real-time capability and more on the flexibility and dynamic reloading of software.

Solution for Efficient Interprocessor Communication

A typical application case, such as communication with a tester, can be implemented in our example without any problems. If it is necessary to fulfill stringent real-time requirements or a higher safety level based on the ISO 26262 standard for functional safety, a multi-processor or System-on-Chip (SoC) architecture makes sense. Such an architecture makes it possible to shift the critical parts of the application to a pure AUTOSAR environment. In such a case, communication is necessary between the two operating system partitions. The implementation is challenging, because it requires a data description known to the two operating systems and a suitable transfer protocol. Vector offers an available solution for the interprocessor communication in the form of MICROSAR.IPC. This module package enables communication between the processors of any combination of operating systems based on POSIX, AUTOSAR Adaptive or AUTOSAR Classic. Its modular layout makes it possible to switch hardware drivers. MICROSAR.IPC can be used for both multi-core architectures (via shared memory) and multi-processor architectures (via serial transmission).

Conclusion

AUTOSAR Adaptive is used to develop a specific implementation for combining automotive functions with a POSIX environment. AUTOSAR Adaptive efficiently makes use of the available dynamic environment and thereby offers an ideal runtime environment for automotive applications (**Figure 3**).

The solution, which is based on AUTOSAR Classic, remains statically configured and is therefore less flexible than an AUTOSAR Adaptive solution. Existing solutions based on AUTOSAR Classic can, however, be integrated reliably into a system with the MICROSAR POSIX approach presented here. This allows the supplier to offer the right solution for every automotive OEM.

Dr. (Engineering) Helmut Brock
has been employed at Vector since 1999 and works as Solution Manager for Embedded Software.

Translation of a German publication in Hanser automotive, 1-2 / 2019.