Specification Control Document for Operating Systems

1. Functional specification

1.1. Operating System Overview

The real-time Operating System in an embedded automotive ECU offers the foundation for dynamic behavior of the software applications. It manages the scheduling of processes and events, the data exchange and synchronization between different processes and provides features for monitoring and error handling. This chapter describes requirements addressed to the Operating System. Applications, in particular Adaptive Applications may not have the system rights to fully use or configure these aspects directly.

1.2. Process Handling

[SWS_OSI_01040] Start Execution Management as Initial Process. dThe Operating System shall allow starting the Execution Management as the Initial Process of the AUTOSAR Adaptive Platform.c(RS_OSI_00105)

[SWS_OSI_01006] Multi-Threading Support dThe Operating System shall allow running multiple execution contexts (threads) such that the process can execute multiple code flows.c(RS_OSI_00100, RS_OSI_00203, RS_OSI_00206)

On multi-core platforms, multiple threads permitted by [SWS_OSI_01006] may execute concurrently on different cores. All the threads belong to some process, so it is possible that multiple threads in the same process may execute on multiple cores concurrently. Additionally, Execution Management requires the ability to bind a specific Process to a core as part of resource management [SWS_EM_02104].

[SWS_OSI_01012] Specification of Core Affinity dThe Operating System shall provide mechanisms for binding processes to CPU cores.c(RS_OSI_00203) In general, a process provides at least the following:

• A main() function as the entry point of the first execution thread of the process.
• A local memory context (address space), providing local, non-shared memory, that includes at least the code, data and heap of the process.
• Some level of memory protection, such that incorrect or invalid memory accesses are detected by the underlying Operating System.
• Operating System descriptors permitting access to OS managed resources.

[SWS_OSI_01008] Multi-Process Support dThe Operating System shall support multiple processes.c(RS_OSI_00206)

2. Scheduling Policies

The Operating System Scheduler is designed to keep all system resources busy allowing multiple software control flows to share the CPU cores in an effective manner. The main goals of the scheduling mechanisms may be one or more from the following:

• Maximizing throughput in terms of amount of work done per time unit. • Maximizing responsiveness by minimizing the time between job activation and actual begin of data processing.

• Maximizing fairness in terms of ensuring appropriate CPU time according with priority and workload of each job.

• Assuring a timelined and ordered activation of jobs according to some policydependent job execution eligibility (e.g. priority, deadline, tardiness, etc).

In real life these goals are often in conflict, implementing the scheduling mechanisms is therefore always a compromise. [SWS_OSI_01003] Default Scheduling Policies dThe AUTOSAR Adaptive Platform Operating System shall support the following scheduling policies defined in the IEEE1003.1 POSIX standard: SCHED_OTHER, SCHED_FIFO, SCHED_RR.c (RS_OSI_00100).

In order to overcome the above mentioned conflicts and to achieve portability between different platforms, the AUTOSAR Adaptive Platform Operating System provides the following scheduling policies categorized in two groups:

- Fair Scheduling Policies
  - SCHED_OTHER
- Real-time Scheduling Policies
  - SCHED_FIFO
  - SCHED_RR

Since the above mentioned default scheduling policies may not guarantee proper execution for all real-time scenarios, the Adaptive Application vendor may provide additional scheduling policies to fulfill any execution requirement. For example, additional non-POSIX scheduling policies like SCHED_DEADLINE (Earliest Deadline First algorithm) could be introduced to satisfy hard real-time requirements.

3. Time Triggered Execution

POSIX PSE51 provides a means to do time-based periodic processing, using the timer API (e.g. timer_settime()) along with POSIX signals. However, signals are sometimes discouraged for safety-critical applications, because they disrupt the execution flow

4. Device Support

The OSI shall support device access as defined in POSIX PSE51.

5. Resource control

While correct behavior is expected from each application, intentional or unintentional misbehavior must be contained for system stability. Simultaneously, some level of dynamic behavior must be allowed. From a feature perspective, applications can be assembled in groups such that they can follow a similar usage pattern, sharing memory, CPU time, and in general resources.

[SWS_OSI_02000] ResourceGroup minimum requirement dThe Operating System shall support the configuration of at least 8 groups of processes in the system.c(RS_OSI_00201, RS_OSI_00202)

Depending on the Operating System, the number of usable ResourceGroups may vary. Furthermore, when OS-level-virtualized containers are used, some Operating Systems may additionally constrain the number of usable ResourceGroups, with an extreme of just 1 available ResourceGroup.

[SWS_OSI_02001] Memory ResourceGroups dThe Operating System shall support a mechanism to define groups of processes that may dynamically allocate memory from a configuration-defined limit.c(RS_OSI_00201) The memory taken in consideration for the limit covers:

• Code and read-only Data from the Executable • Modifiable Data from the Executable
• Memory used for thread stack for each thread of the process
• Heap
• System memory that is used by the Operating System for holding the kernel resources allocated to the process (e.g. thread control block, semaphore, page table entries for MMU mapping, etc)
• Shared memory between processes of the same group
• Implicitly loaded shared objects between processes of the same group

6. Startup

The startup steps of an Operating System have to be executed in an implementation-specific way. These steps include starting any Operating Systemrelated middleware, including device-drivers and services handling low-level middleware, as well as starting Execution Management.

As an important remark, it is expected that Execution Management will be started early on during the system boot, ideally as the first process, in order to allow booting all the required Processes. However, depending on the Operating System, other system services and supporting middleware may be started before or in parallel. An example may be a filesystem service, if the Operating System has one that is not part of its kernel.

7. Shutdown

Similarly, shutdown steps for an Operating System are implementation-specific. They may include flushing some middleware buffers, shutting down some peripherals, and optionally turn off the entire system, depending on the system configuration.

8. API Specification

The AUTOSAR Adaptive Platform does not specify a new Operating System for highly performant microcontrollers. Rather, it defines an execution context and programming interface for use by Adaptive Applications.

8.1. C++ language binding Operating System

There are several Operating Systems on the market, e.g. Linux, that provide POSIX compliant interfaces. However Applications are required to use a more restricted API to the Operating Systems as compared to the platform services and foundation. In particular, the starting assumption is that an Adaptive Application may use PSE51 as OS interface whereas platform-specific Application may use full POSIX.

The implementation of platform Foundation and platform services functionality may use non-PSE51 APIs, even OS specific ones. The use of specific APIs will be left open to the implementer of the AUTOSAR Adaptive Platform and is not standardized.

8.2. API Common Data Types

There are no additional data types specific to the AUTOSAR Adaptive Platform defined in this document. Please refer to the SWS_AdaptiveCore document for base AUTOSAR Adaptive Platform definitions.

8.3. API Reference

There are no additional APIs specific to the AUTOSAR Adaptive Platform defined in this document. Please refer to the SWS_AdaptiveCore document for base AUTOSAR Adaptive Platform definitions.

9. Service Interfaces

Operating System does not define any AdaptivePlatform Service Interface.

9.1 Type definitions Operating System does not define any AdaptivePlatform Service Interface type definitions.

9.2 Provided Service Interfaces Operating System does not define any AdaptivePlatform Service Interface.

9.3 Required Service Interfaces Operating System does not require any AdaptivePlatform Service Interface.

9.4 Application Errors Operating System provides error reporting through its C/C++ Application Interface.