ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology
Vol. 2, Issue 7, July 2013

An Overview of AUTOSAR Multicore Operating System Implementation

Devika K[1], Syama R[2]

Post Graduate Student, Dept. of Computer Science, SCT College of Engineering, Trivandrum, Kerala, India[1]

Asst. Professor, Dept. of Computer Science, SCT College of Engineering, Trivandrum, Kerala, India[2]

Abstract: A multicore processor is a single computing component with two or more independent processing units for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks. Multi-core processors are widely used across many application domains including automotive domain. Integrated systems that deliver entertainment and information in automobiles are called In Vehicle Infotainment (IVI) systems. IVI user can experience applications such as bluetooth wireless technology telephony, internet radio, location-based services (including traffic reporting), and navigation in vehicle. In order to achieve IVI vehicle should be connected to the internet. When vehicle become connected to the internet, demand for internet-based entertainment applications and services increases. This leads to the requirement of high performance processors like multicore processors in automobiles. Designing operating systems for multicore processors is very crucial because, the improvement in performance depends very much on the software algorithms used and their implementation. The AUTOSAR standard has introduced support for development of multi-core operating system for embedded real-time systems. New concepts introduced are locatable entities, multi-core startup/shutdown, Inter OSApplication Communicator (IOC) and spinlocks. This paper discussing available techniques towards AUTOSAR multicore operating system implementation

Keywords: OSEK/VDX, AUTOSAR, Spinlock, IOC, Synchronization

I. INTRODUCTION

Embedded system applications are widely used today in aerospace, automotive areas. After the introduction of application specific operating systems, specialized real time operating systems called Vehicular Application Specific Embedded Operating Systems (VASOS) are developed. Some VASOS are VxWorks [1], QNX [2], pSOS [3] etc. They provide functions of vehicle domain such as device drivers, fundamental network functions. OSEK and its successor AUTOSAR plays an important role in software development in automotive domain.

A. OSEK/VDX

OSEK/VDX (Open systems and corresponding interfaces for automotive electronics / Vehicle Distributed executive) is an open vehicular industry standard. It provide environment for developing and redeveloping ECU software and improve compatibility of those applications. Advantages of OSEK/VDX are portability and reliability. The OSEK OS is specification for a single processor. Various parts of OSEK/VDX are OS (the basic services of the real-time kernel), COM (the communication services), NM (the Network Management services) and OIL (OSEK Implementation Language). OSEK/VDX is an operating system meant for distributed embedded control units. OSEK/VDX provides following OS services.

1) Task management : OSEK/VDX specification provides two different task types, basic task and extended task. A basic task does not contain system calls that are able to block the task. Synchronization points are only at the beginning and the end of the task. An extended task contains invocations of OS services which may result in a WAITING state. Extended tasks differ from the basic tasks by the fact that they can use the *WaitEvent* system call which can lead that task in a waiting state. In this state the operating system can switch to a lower priority task. The management of the extended tasks is more complex and requires more resources (memory, processor time). The priority of a task is statically defined and cannot be modified during the run-time of an application. The tasks are activated with system calls: ActivateTask or ChainTask[4]. A task can be managed in three ways, totally preemptive, non-preemptive, and mixed preemptive. In preemptive operation a task in ‘active’ state can change state to ‘stand-by’ if a predefined condition occurs. This technique imposes constraints on the response times of the tasks. Tasks can only be scheduled from predefined places through system calls. The mixed-preemptive strategy supposes the coexistence in the same system of the totally-preemptive and non-preemptive tasks.

2) Conformance classes: The conformance classes define several versions of the real-time executive in order to adapt it to different application requirements. Four conformance classes were defined which are determined by three attributes and some implementation requirements such as type of task, possibility of recording multiple activation requests for a basic task and number of tasks per priority level. BCC1 and BCC2 classes only allow basic tasks while ECC1 and ECC2 classes allow basic and extended tasks. For BCC1 and ECC1 classes all the tasks have different priority levels and it is not possible to record multiple activation requests for the tasks. For BCC2 and ECC2 classes several tasks can share the same priority level and it is possible to record activation requests for basic tasks[2].

3) Scheduling policy : Scheduling is independent of the conformance classes and type of task. Static priorities are assigned to tasks. Several tasks can share the same priority level according to the conformance class they use. Scheduling can be pre-emptive, non pre-emptive or mixed pre-emptive. In the case of mixed pre-emptive, task can select appropriate mode either pre-emptive or non pre-emptive. Tasks that are sharing common internal resources are grouped. Inside the group the tasks can't pre-empt among themselves.

4) Task synchronization : Synchronization comes only in the case of extended tasks. Synchronization is based on the private event mechanism i.e., only the owner task can explicitly wait for the occurrence of one or more of its events. The setting of occurrences can be made by tasks (basic or extended) or ISRs (Interrupt Service Routine). There is no time-out associated to the *WaitEvent* service. Temporal behavior can be monitored using the alarms.

5) Resource management : Concurrent access to shared resources is handled using OSEK-PCP (Priority Ceiling Protocol). OSEK-PCP protocol is also known as Highest Locker protocol and it is more simple than original PCP [5] [6]. When a task gets a resource, its priority is immediately raised to the resource priority. So other tasks that share the same resource cannot get the CPU. The resource sharing is allowed between tasks and ISRs or between ISRs [7]. For that purpose, a virtual priority is assigned to every interrupt. Two services allow controlling access to resources: *GetResource* and *ReleaseResource*. A predefined system resource "Res Scheduler" allows a task to lock the processor. This allows a task to execute some code in non pre-emptive scheduling mode.

6) Alarms and counters : They allow the management of periodic tasks and watchdog timers for the monitoring of various situations. A counter is an object intended for the counting of "ticks" from a source. An alarm allows to link a counter and a task. An alarm will expire when a predefined counter value is reached. At the occurrence of the alarm an action is carried out. An alarm can be defined to be a single alarm or a cyclic alarm, the corresponding counter value being absolute or relative to the current counter value.

7) Communication: Communication in OSEK/VDX system is by means of message passing. There are two types of messages, one type using blackboard model or unqueued message and another type is using FIFO.

8) Interrupts : OSEK supports two types of interrupt service routines, Type1 and Type 2. Type1 ISRs does not use system services. When the routine finishes the execution of the program continues from the point it was interrupted; so the interrupt does not affect the tasks management. Type2 ISRs contains system calls and when the routine finishes a switching task can occur if higher priority tasks are active. DisableAllInterrupts, EnableAllInterrupts, SuspendAllInterrupts, ResumeAllInterrupts are used for enabling/disabling Type1 interrupts and SuspendOSInterrupts, ResumeOSInterrupts are used for Type 2 interrupts. Their goal is to protect critical portions from the interrupt code. ISR are triggered by an external event resulting in an interrupt request on the host microcontroller. Each ISR is assigned a static priority. The priority levels used for ISR should be greater than the priority levels of tasks. ISR execution can be preempted by the activation of another ISR. This is depending on underlying hardware of microcontroller. OSEK/VDX OS allows resource sharing between task and ISR2. Some implementations consider ISR2 as tasks by allowing context switching from an ISR2 to a task and back. Schedulability analysis techniques do not make any difference between tasks and ISR.[8]

9) Interprocess communication : OSEK includes an events mechanism for interprocess synchronization. The events are entities controlled by the operating system and assigned to extended tasks only. Each extended task has a predefined number of events and it is their owner. When an extended task is activated its events are reset. Through the events the tasks are able to exchange information. Any task can set any event owned by any task but only the owner of an event can reset the event or wait for that event.

10) Task Management **:** All the objects of application are configured off-line at the time of system configuration activity. They are created before starting execution of the application and never destroyed. So they are static. The

system is usually modeled as a set of tasks, each task being characterized by different attributes, e.g. WCET, activation law, deadline, shared resources, etc.

B. AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture)[11] is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. Its objective is to create and establish open standards for automotive E/E (Electrics/Electronics) architectures that will provide a basic infrastructure to assist with developing vehicular software, user interfaces and management for all application domains. This includes the standardization of basic systems functions, scalability to different vehicle and platform variants, transferability throughout the network, integration from multiple suppliers, maintainability throughout the entire product life-cycle and software updates and upgrades over the vehicle's lifetime are some of the key goals. Autosar OS is backward compactable. i.e, it will support all services provided by OSEK.

1) Scheduling Policy : AUTOSAR scheduling policy is an extended version of scheduling policy used by OSEK/VDX. Highest Priority First Policy is used in AUTOSAR. If many tasks share same priority level, FIFO is used as a second criterion. There exists a notion of group, for tasks that share a common internal resource. An internal resource is automatically locked by a task of the group when it gets the CPU and released when it terminates, waits for an event or invokes the Schedule service. Usual preemption rules are used for the tasks which are not in the group, according to their priority level. Inside the group the tasks cannot preempt among themselves. AUTOSAR OS specific extensions concern timing protection service. The purpose of this service is to prevent the occurrence of timing faults. To do so, this service calls a user provided function each time when ever one of the following conditions occurs:

- The execution time of a task or ISR2 is greater than the (offline specified) expected value.
- The execution time of a task or ISR2 while holding a shared resource is greater than the (offline specified) expected value.
- The arrival rate of a task or ISR2 is greater than its (offline specified) expected limit. For a task, this limit is the time that an instance of the task can spend in the running state in a given time frame. This mechanism is closed to the a periodic server concept of real-time scheduling theory[8]. For an ISR2, this limit is the number of activations in a given time frame.

2) Shared resources management : AUTOSAR OS coordinates the concurrent access to shared resources with the OSEK-PCP protocol (Priority Ceiling Protocol)[9]. When a task gets a resource, its priority is immediately raised to the resource priority, so that other tasks that share the same resource cannot get the CPU.

3) Alarm, counters and schedule tables : AUTOSAR OS uses OSEK/VDX OS alarm and counter related services with some minor modifications, and extends these services through the schedule table concept. Alarms and counters allow the processing of recurring phenomena, for instance timer ticks, or signals from mechanical organs of a car engine. When associated with a timer, they allow the management of periodic tasks. A counter is an object intended for the counting of "ticks" from a source. Each counter has a maximal value: when this value is reached, the counter is reset to 0. An alarm allows to link a counter and a task. It expires when the counter reaches a predefined value. Then, a statically defined action is taken: either the activation of the associated task, or the setting of an event of the task. An alarm can be defined to be single-shot or cyclic, the corresponding counter value being absolute or relative to the current counter value [10].

Schedule tables are an extension of the alarm concept. Like an alarm, a schedule table is linked to a counter. It is composed of a set of expiry points, whose corresponding counter value is relative to the activation of the schedule table. When an expiry point is reached, one or more actions (task activation or event setting) are taken. A schedule table can be defined to be single-shot or cyclic, the corresponding counter value being absolute or relative to the current counter value. Schedule table is usually associated to offline scheduling techniques[11]. A schedule table describes completely the activation points of all the tasks of the system. When a task is activated, it runs until it completes (no preemption).

C. AUTOSAR Multicore OS Architecture

Multi-core processors are becoming increasingly prevalent, with multiple multi-core solutions being offered for the automotive sector. On understanding this trend, the AUTomotive Open System ARchitecture (AUTOSAR) standard version 4.0 has introduced support for multi-core embedded real-time operating systems[12]. AUTOSAR Version 4.0 standard has introduced new concepts such as locatable entities (LEs), multi-core startup/shutdown, Inter-OS-Application Communicator (IOC), and SpinlockTypes.

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology
Vol. 2, Issue 7, July 2013

A locatable entity is an entity that has to be located entirely on one core. The assignment of LEs to cores is defined at configuration time[12]. Application, Tasks, Counter, alarms all are LEs in AUTOSAR 4.0. The way cores are started depends heavily on the hardware. The hardware starts only one core, referred as the master core, while the other cores (slaves) remain in halt state until they are activated by the software. Synchronized and individual shutdown [12] concepts are defined in Multicore version of AUTOSAR OS. In synchronized shutdown concept it is possible to shutdown all cores simultaneously.

A key element of the AUTOSAR multi-core specification is the spinlock mechanism for inter-core task synchronization. This mechanism is used to support mutual exclusion for tasks in different Cores. Another important feature is the Inter OSApplication Communication (IOC) which allows the communication between the different Application located on same or different cores. The IOC provides sender-receiver (signal passing) communication [12].

Software manufacturers are now increasing speed of processors and go toward parallel processing to increase performance. Single-kernel operating system was not enough to manage multi-core processors so the idea of multi-kernel operating system arises. One of these kernels serves as the master kernel and the others serve as slave kernels. Multi-core processors came to solve the efficiencies of single core processors, by decreasing power consumption while increasing bandwidth.

Modern cars are highly complex systems consisting of a huge number of mechanical and electrical parts. An observable trend in the automotive industry shows that more and more features and innovations are based on electrics/electronics and therefore on software. Hence the demand of computational resources in the vehicle is growing rapidly.As the demand for computing power is quickly increasing in the automotive domain, car manufacturers and tier-one suppliers are gradually introducing multicore ECUs in their electronic architectures. Additionally, these multicore ECUs offer new features such as higher levels of parallelism. Automotive systems have recently seen a tremendous growth in terms of advanced software features such as driver-assist technologies, active safety systems, and interactive entertainment platforms. Multi-core processors constitute an effective means for realizing such computationally-intensive applications in future automotive platforms. Recognizing these changes in application characteristics and processor capabilities, the AUTomotive Open System ARchitecture (AUTOSAR) Version 4.0 standard has introduced support for multi-core embedded real-time operating systems. New concepts such as locatable entities (LEs), multi-core startup/shutdown, Inter-OS-Application Communicator (IOC), and SpinlockTypes have been introduced in the AUTOSAR multi-core OS architecture specification to extend the single-core OS specifications.

II. OVERVIEW OF AUTOSAR MULTICORE FEATURES AND IMPLEMENTATION

A. *AUTOSAR Spinlocks*

In single-core processors mutual exclusion is facilitated by AUTOSAR using a function called GetResource(). This function leverages the priority ceiling protocol: when a task acquires a resource, its priority is temporarily raised to the highest priority among all the tasks that can use the resource. This reduces the duration for which any task waits for a lower-priority task to release a shared resource. It bounds and significantly reduces priority inversion. Priority inversion cannot be completely eliminated when logical resource sharing is required, but it can be minimized by using appropriate protocols. In the current context, priority inversion cannot be eliminated since the lower-priority task holding the resource has to still release it before the higher-priority task can proceed due to the non-preemptive and mutually-exclusive nature of shared resources. However, GetResource() uses the priority ceiling protocol, which ensures that the priority inversion is no more than the duration of a single resource-holding duration. This mechanism does not scale to multi-core processors since priorities are insufficient in preventing access from tasks executing on other cores.

A key extension in the AUTOSAR 4.0 multi-core specification is a new mechanism for mutual exclusion across tasks on different cores called as the SpinlockType. This is a busy-waiting mechanism that polls a (lock) variable until it becomes available using an atomic test and set functionality. Once a lock variable S_l is obtained by a task, other tasks on other cores will be unable to obtain the lock variable S_l, effectively ensuring the constraint of mutually exclusive access to the resource R_l protected by the lock S_l. This mechanism works with multi-core processors, since it relies on shared memory locks as opposed to the priority attribute that does not span cores[13].

With a spinlock-mechanism a thread or a task asks for a boolean lock, to enter a critical section. If the lock held by another task, the boolean variable is set to true by asking aask has to wait. At the same time the thread asks continuously for the lock (polling) and keeps active. This mechanism is called busy waiting. This technique is especially used for mutual exclusion with short times of waiting. To make sure that no other thread tries to enter the

protected section in the time between testing the lock variable and set the lock to true, an atomic test-and-set-function has to be provided. The SpinlockType is an approach to address the shortcomings of the priority ceiling protocol when extending AUTOSAR for use in multi-core processors. However, it introduces two problems deadlock and starvation.

1) Deadlock : In the context of non-nested spinlocks, the SpinlockType mechanism could potentially lead to deadlocks. The deadlock happens when a lower priority task holding a resource protected by SpinlockType A gets preempted by a higher priority task that tries to acquire the same SpinlockType A.

The current solution presented in the AUTOSAR specification is to (a) return an error to TASK/ISR2 trying to acquire a spinlock assigned to another TASK/ISR2 on the same core or (b) protect a TASK by wrapping the spinlock with a SuspendAllInterrupts() call so that the task cannot be preempted [12]. With respect to (a), returning an error to the second task trying to acquire the spinlock A is not quite useful. It would be desirable to avoid this scenario in the first place, since application developers should not have to worry about such errors resulting from operating system scheduling decisions [13]. The advantage of (b) is that using SuspendAllInterrupts() reduces the amount of remote blocking suffered during multi-core synchronization. In the context of multi-core mutual exclusion, remote blocking is defined as the duration for which a task waits for a shared resource to be released on a remote core

One way to avoid deadlock is not allow nesting of different spinlocks. Optionally if spinlocks shall be nested, a unique order has to be defined. Spinlocks can only be taken in this order whereas it is allowed to skip individual spinlocks[12]. Another way is wrapping the spinlock with SuspendAllInterrupts, so that it cannot be interfered by other tasks[12]. Priority ceiling protocol used for single core does not work with multicore. Because each core has its own priority tasks. So Multiple Priority Ceiling Protocol[14] is introduced.

2) Starvation : Depending on the hardware implementation of the test-and-set mechanism and the task set characteristics, it could lead to starvation with a task potentially never getting the Spinlock. This can happen if one or several tasks get a lock a number of times, while a other tasks is still spinning for its first access to the shared resource. The danger of starvation can be reduced by a proper scheduling algorithm. The most used scheduling algorithms for multicore systems based on proportionate fairness[17]

B) AUTOSAR IOC

Inter-core communication consists of two primary actions: Data movement and Notification including synchronization.

1) Data Movement

The physical movement of data can be accomplished by several different techniques:

- Use of a shared message buffer — The sender and receiver have access to the same physical memory.
- Use of dedicated memories — There is a transfer between dedicated send and receive buffers.
- Transitioned memory buffer — The ownership of a memory buffer is given from sender to receiver, but the contents do not transfer.

Using a shared memory buffer means that a message buffer is set up in a memory that is accessible by both sender and receiver, with each responsible for its portion of the transaction. The sender sends the message to the shared buffer and notifies the receiver. The receiver retrieves the message by copying the contents from a source buffer to a destination buffer and notifies the sender that the buffer is free. It is important to maintain coherency when multiple cores access data from shared memory[18].

It is also possible for the sender and receiver to use the same physical memory, but unlike the shared memory transfer common memory is not temporary. Rather, the buffer ownership is transferred, but the data does not move through a message path. The sender passes a pointer to the receiver and the receiver uses the contents from the original memory buffer[18].

2) Notification Methods

List of notification methods are discussed in [18]. They are

Non-blocking Polling : In this method, the receiver checks to see if there is a descriptor waiting for it in the receive queue. If there is no descriptor, the receiver continues its execution.

Blocking Polling : In this method, the receiver blocks its execution until there is a descriptor in the receive queue, then it continues to process the descriptor.

*Interrupt-based Notification :*In this method, the receiver gets an interrupt whenever a new descriptor is put into its receive queue. This method guarantees fast response to incoming descriptors. When a new descriptor arrives, the receiver performs context switching and starts processing the new descriptor.

ISSN: 2319-8753

International Journal of Innovative Research in Science, Engineering and Technology
Vol. 2, Issue 7, July 2013

Delayed (Staggered) Interrupt Notification : When the frequency of incoming descriptors is high, the navigator can configure the interrupt to be sent only when the number of new descriptors in the queue reaches a programmable watermark, or after a certain time from the arrival of the first descriptor in the queue. This method reduces the context switching load of the receiver.
QoS-based Notification : A quality-of-service mechanism is supported by the Multicore Navigator to prioritize the data stream traffic of the peripheral modules; this mechanism evaluates each data stream with a view to delaying or expediting the data stream according to predefined quality-of-service parameters. The same mechanism can be used to transfer messages of different importance between cores.

In [15] Autosar multicore implementation in Simultaneous Multithreading (SMT) processor is explained. An inherent problem in multithreaded environments is the synchronization of concurrent threads that need to access shared data. Lamport [20] devised a solution called "Bakery Algorithm" extending Dijkstra's work [19] in this domain. Another solution was presented by Peterson [19]. However, both approaches use busy waiting to enter a critical section that is currently blocked by another thread. Such busy waiting will waste processing cycles that could be utilised otherwise by the blocking thread to finish the critical section earlier. Therefore, typical solutions for mutual exclusion suspend the execution of the blocked thread until the lock is free. The AUTOSAR task model cannot cope with this.

A mechanism called task filtering is proposed in [15] to fulfill the AUTOSAR resource management requirements. For this purpose the OSEK PCP and scheduling are extended and thus adapt it for its use in multithreaded processors.

Definition 1 (Containers)

- The Execution Set E contains all tasks that are currently scheduled into the SMT processor's thread slots.
- The Active Set A contains all active tasks (states ready and running), i.e. the tasks from all FIFO queues of the scheduler .
- A task's resource set R_T comprises all resources that a task T will access during its execution.
- A resource's task set A_R references the active tasks that might access resource R ($T \in A_R \Leftrightarrow R \in R_T$).
- The Task Filter TF contains tasks, that are ready and should be executed according to their priority, but can-not be scheduled yet due to resource

Definition 2 (Terms)

- Two tasks T1, T2 have a resource conflict, if they might use at least one resource R concurrently at runtime (i.e. $\{T_1, T_2\} \subset A_R$ resp. $R \in R_{T1} \cap R_{T2}$)
- A task T is ready concerning the resource management (resource-ready), if
  – it does not need any resource, or
  – $\forall R \in R_T$: T has the highest priority in A_R

Definition 3 (Task Filtering)
An active task $T \in A$ can be put into the execution set E, i.e. can be scheduled for execution, if it is resource-ready. All other tasks are in their FIFO queues. If a task T_1 has a resource conflict with a running lower-priority task T_2, which cannot be preempted, T_1 is buffered in the task filter, until T_2 can be rescheduled.

With this approach, deadlocks are completely avoided. The filtering of the tasks ensures, that only such tasks are executed that do not have any resource conflict among each other, i.e. the execution set E is free of resource conflicts. For the same reason, no direct blocking can occur.

In [16] a proposal called M-AUTOSAR is introduced whose architecture is shown below.

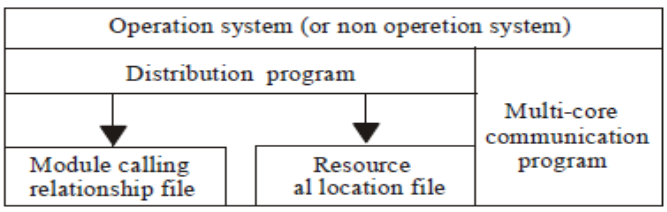

Fig. 1: Internal architecture of Multi-core Service module

M- AUTOSAR[16] architecture increases the Multi-core Service module, which provides the calling information related to multi-core and resource allocation information and inter-core communication information. These modules can be executed by the operating system. The internal components of this composition include module calling relations file, resource allocation file and inter-core communication program and a distribution program.

Working relationship between the various modules is shown in Fig.1. Module calling relationship file[16] is responsible for storage the distribution file, which described the operation allocation for each program of each processor. The resource allocation file is responsible for the system resource using moment and the using processor. The allocation program is used by the operating system when there is an operation system which is used in the project. If there is not an operation system existed in the project, the main program will call the allocation program in the initialization stage for resource allocation. The multi-core communication program include the interlock program and data exchange mode between cores, it is used by the operating system and called by the upper application.

Module calling relationship file describes the corresponding processor core id for each module. This also describes the corresponding interrupt information for each module. Resource allocation file is a list file, including the resource name and which one uses these resources. Resource allocation file also includes hardware interrupt handlers for interrupts. Of course, some interruption can be used by any one of multiple processors for processing. It will also be in the file specified. Below is a part of a instances files for the Resource allocation file. This part includes the allocation of resources and registers. Program allocation module is based on the above two files and this module is response for the resource allocation and resource management. The main module can be performed either in support of the early completion of the build system or by main function. M-AUTOSAR chose the software module called by the main function because the build system still does not support the processing of these files.

III. CONCLUSION

Due to the increasing demand for electronic security and comfort in vehicles it seems to be necessary to use more multi core ECUs. It is the only way to increase the performance. AUTOSAR 4.0 is a first step made in this direction. In the area of code migration from single core code to multi-core code there is an enormous development pressure of automated solutions. How far and fast multi core processors spreading depend on a number of criteria. Multicore operating system is used in cars if there is a large need for more processing power and the performance advantage of multi processor ECUs. Multi core ECUs can only cover the market, if the costs of the software development are not growing dramatic. This can only succeed if existing single core code can be migrated cost-effectively on multi core systems.

REFERENCES

[1] Vx'xworks Progrommer's Guide, 5.1, Wind River Systems,1993.
[2] D. Hildebrand, "An architectural overview of QNX," in Prnc. Usenix Workshop on Micro-Kernels and Orher Kernel Architectures, April 1992.
[3] L. M. Thompson, "Using pSOS+ for embedded real-time computing,'' in COMPCON, pp. 282-288, 1990.
[4] Yuan Sun and Fei-Yue Wang, "A Design Architecture for OSEK/VDX-based Vehicular Application. Specific Embedded Operating Systems," IEEE, 2005.
[5] R. Rajkumar and J. Lehocsky. Priority Inheritance Protocols: an Approach to Real-Time Synchronisation. IEEE Transaction on Computer, 39(9):1175–1185, 1990.
[6] J. W. S. Liu. Real-Time Systems. Prentice Hall Inc, 2000.
[7] Yuan Sun and Wang F.Y, "Trampoline - An OpenSource Implementation of the OSEK/VDX RTOS Specification" Intelligent Vehicles Symposium, 2005. Proceedings. IEEE
[8] Hladik P E, Deplanche A M, Faucou S, Tringuet Y, "Adequacy between AUTOSAR OS specification and real-time scheduling theory", Industrial Embedded Systems, 2007. SIES '07. International Symposium , pp 225-233
[9] Kopetz,H. ; Obermaisser, R. ; El Salloum, C. ; Huber, B., "Automotive Software Development for a Multi-Core System-on-a-Chip", Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07.
[10] Senthilkumar,K. , Ramadoss, R."Designing Multicore ECU architecture in vehicle networks using AUTOSAR", Advanced Computing (ICoAC), 2011 Third International Conference
[11] AUTOSAR Specification of Operating System, Version 3.1
[12] Autosar Specification of Multi-core OS Architecture, Version 4.0
[13] Karthik Lakshmanan, Gaurav Bhatia, Ragunathan (Raj) Rajkumar "AUTOSAR Extensions for Predictable Task Synchronization in Multi- Core ECUs", SAE 2011 World Congress & Exhibition
[14] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," Distributed Computing Systems, 1990. Proceedings., 10th International Conference on , vol., no., pp.116-123..
[15] Florian Kluge, Chenglong Yu, Jorg¨ Mische, Sascha Uhrig, Theo Ungerer, "Implementing AUTOSAR Scheduling and Resource Management on an Embedded SMT Processor" SCOPES '09 Proceedings of th 12th International Workshop on Software and Compilers for Embedded Systems, pp. 33-42
[16] Bin Sun and Yue-li Hu, "Improved AUTOSAR Based on Multi-Core Architecture and its Application in the Body Control" , 2011, Research Journal of Applied Sciences, Engineering and Technology 3(10): pp. 1197-1202
[17] G. Nelissen, J. G. Vandy Berten, und D. Milojevic." An Optimal Multiprocessor Scheduling Algorithm without Fairness". Aus The 31st IEEE Real-Time Systems Symposium. Parallel Architectures for Real-Time Systems (PARTS) Research Center Universite Libre de Bruxelles, 2010.
[18] E. W. "Dijkstra. Solution of a problem in concurrent pro-gramming control. Commun. ACM", 8(9):569, 1965.
[19] L. Lamport. A new solution of Dijkstra's concurrent pro-gramming problem. Commun. ACM, 17(8):453–455, 1974
[20] M. Herlihy. Wait-free synchronization. ACM Trans. Pro-gram. Lang. Syst., 13(1):124–149, 1991.
[21] www.autosar.org
[22] www.windriver.com