SHARB: Shared Resource Arbitration in Partitioned Multicore Systems via Library Interposition
Andreas Knirsch, Pierre Schnarz, Joachim Wietzke

Abstract—With the increasing computational capabilities of multicore hardware platforms for embedded systems, the extent of software-based functionalities also grows continuously. Concurrently, formerly distributed functionalities are now integrated into a common platform. This is supported by multicore architectures, which allow the parallel computation of independently developed software components. This constitutes new opportunities and challenges. One of the latter is to integrate software components with different temporal requirements on a common hardware platform. Shared system resources compound the temporal interference between different software components. In the following an approach is presented that supports the development and integration process through the use of static priorities to manage the access to shared resources from independently executed components. The domain of In-Car Multimedia is utilized to illustrate challenges and the proposed solution. The applicability is demonstrated through the use of a prototypical implementation.

Index Terms—multicore processing, parallel programming, embedded software, multimedia systems, automotive applications

I. INTRODUCTION

For the past few decades the extent of automotive software (SW) based functionalities has grown continuously. More recently, this has particularly applied to automotive infotainment systems, which are now available for all classifications of cars. A change in that process of growth is not foreseeable. Current in-vehicle infotainment (IVI) systems provide a rich variety of functionality and information to the vehicles’ passengers. In the past, the main focus was on applications that offered information and entertainment facilities. Meanwhile, IVI systems implement the human machine interface (HMI) for a number of comfort functions using a link to different vehicular fieldbus networks. Within this context the interaction with the user gains significance. This shifts the focus from the straight supply of contents to interactive multimedia systems, known as in-car multimedia (ICM) systems.

The latest ICM systems are connected to infrastructure-based wireless access networks. With this connectivity vehicles evolve to mobile network nodes, which enables a wide range of new applications. Based on this development, the evolutions within the area of consumer electronics (CE) have already influenced the applications available within cars and the demands of their users. Formerly limited to the synchronization of data with CE devices, they are now fully integrated and aim to make use of the vehicle’s visualization and operation capabilities [1], [2].

A. Knirsch, P. Schnarz, and J. Wietzke: In-Car Multimedia Labs, Faculty of Computer Science, University of Applied Sciences Darmstadt, Haardtring 100, D-64295 Darmstadt, Germany Email: andreas.knirsch@h-da.de, pierre.schnarz@h-da.de, and joachim.wietzke@h-da.de.

For future systems, the incorporation of mobile devices might be realized as dynamic integration on the abstraction-level of applications (apps), similar to the current state-of-the-art application deployment within the domain of smartphones. A prerequisite is the availability of appropriate SW frameworks and open standards to support the functional development of the ICM systems’ SW components [3], [4].

Connectivity to both the vehicle’s environment and in-vehicle subsystems constitute another important prerequisite for future applications, which also includes the area of driver assistance applications. For example, available sensor technology like satellite positioning receivers (e.g. GPS), yaw-rate sensors, or wheel-speed sensors has already extended beyond the vehicular system’s boundaries to factor in the current situation of the traffic for route calculation and guidance. Networking positively affects future driver assistance systems through an augmented set of sensors. These may require a very accurate determination of the geographic position and anticipated driving directions. An ICM system can provide both of these by considering a satellite and an odometric based positioning system after alignment to accurate map data. A prerequisite for very accurate map data is the capability to dynamically update the on-board data using appropriate communication channels. Thereby ICM systems act as the producer and consumer for data used by other electronic control units (ECU) and remote applications ("within the cloud"). Although ICM systems were operated within a safety relevant environment before, they are gaining significance within the context of functional safety and dependability regarding their increasing role as handler for data and information at the vehicular system’s boundary.

ICM systems still act, among other things, as FM-tuner, media player for local audio/video files and network streams, navigational device, phone, address book, email client and Internet browser. These applications are offered to the driver and other passengers through several displays and control elements. They are presented to the users in terms of an integral and context sensitive HMI, allowing multimodal operations and are adaptable to individual users. The underlying functionality of these applications is integrated into a SW system on a common hardware (HW) platform: the "head-unit" [4].

ICM SW systems are decomposed into interdependent components to counter a growing system complexity [5] and to achieve short development cycles. Independent third-party suppliers concurrently develop the SW components under a high division of labor. Thereby the tier-1 original equipment manufacturer (OEM) adopts the role of the integrator. The individual components and their interfaces are essentially
defined by the use of a functional viewpoint, although they also have to fulfill different temporal requirements and provide sufficient responsiveness to user inputs. This includes and combines both time and event triggered tasks [6], for which operating systems provide adequate support. They provide control regarding the assignment of computational capacity to tasks using appropriate scheduling strategies and task priorities. In practice round robin scheduling (SCHED_RR), in combination with different priority levels, has demonstrated applicability for ICM systems, not least because of the relatively good maintainability and predictability for systems with many tasks (>500). Unfortunately this cannot be ensured for all components implemented by third party suppliers. Also, it has to be considered that different scheduling strategies are not necessarily compatible with each other (e.g. cooperative and preemptive), which also applies for independently structured priority concepts.

The adaptation of SW components using incompatible scheduling strategies and task priorities to achieve a homogeneous system that fulfills all temporal requirements implies significant coordination efforts, both at the organizational level and for the developers and respectively the integrators. A possible implication might be the re-engineering of considerable parts of certain SW components, which on their own already fulfill all the functional and non-functional requirements, but which are not suitable for the concurrent use of shared resources. In this case the requirements for the components could be qualified as not appropriate. However, it also has to be noted that the already existing components of former systems ("legacy SW") and third-party SW (COTS) have to be considered, which "cannot" be changed at all. The underlying causes might be manifold: an integrator (tier-1 OEM) probably has no interest in financing the necessary efforts or the third-party supplier does not want to incorporate the required changes due to strategic reasons. Hence, with an increasing number of parties involved, a homogenization of the SW components and their behavior becomes more complex. Also, the rising number of components puts emphasis on the challenge to enforce and fulfill superordinate temporal requirements. Additionally, even though the integration of components is done routinely, it is characterized by the use of heuristic and improvised approaches [7].

As a result of the integration onto a common platform the available system resources are concurrently used by different components. Besides the CPU this also applies for memory and input/output (IO) devices. Thereby, and with the focus on the overall system architecture, the complexity is reduced and simplified in comparison to a federated architecture consisting of separate HW components. The reason for this is the avoidance of additional HW components and necessary communication facilities for inter-connecting. The simplification at HW level and intercommunication led to an increased integration density at SW level. According to the rising demands regarding the functionality of ICM systems the demand for computational power increases. Here multicore (MC) based HW platforms provide a solution [8, p.167 ff.]. Their assignment issues a new challenge in respect of the implementation and the necessary synchronization of parallel computed tasks [9], [10], and in particular regarding the interplay and unwanted interference in between certain SW components.

The availability of multiple processor cores forms the basis for an approach to structure SW components through the use of execution domains (ED) [11]. The utilization of a single operating system avoids the additional overhead introduced through a virtualization-based approach for structuring components. With EDs, formerly physical isolated SW systems can be integrated into a highly integrated head-unit. The temporal behavior of the targeted system also becomes predictable for high system load situations.

Even though MC platforms provide multiple computational cores, there are still a number of resources that are available only once. Access to such shared resources is realized using a concurrent behavior, meaning each accessing component has to compete with others for the shared resource. On a single core system such access is implicitly arbitrated by the task scheduler and controlled using task-priorities: access to a certain resource is only granted as long as the accessing task is scheduled for computation by the task scheduler and when the state is changed to "running". This relies on the limitation that in a single core system only a single task in the state "running" is possible. If there are multiple computational cores available, multiple tasks can be computed in parallel and they can also potentially access the same shared resource in parallel. The temporal order of the latter is not deterministic. This affects the temporal behavior and predictability of those accessing tasks. Therefore, the correct functioning of time critical tasks cannot be guaranteed, especially in high system load situations. This might lead to the perception that the use of single core systems can ease the integration of multiple SW components due to implicit arbitration using the operating system’s task scheduler. However, with such systems the "bottleneck" is the scheduling of tasks using incompatible priorities and scheduling strategies, not forgetting to mention the insufficient computational power. The move to MC systems is unavoidable for CPU-intensive parallel applications. This implies a move of the "bottleneck", where multiple tasks compete for a single resource in parallel. A configurable arbitration of such concurrent access is not available.

In the following, an approach is presented to arbitrate the access to shared resources in a parallel computed and component-based SW system that is configurable for the integrator. Therefore, the example of an ICM system is used. The main objective is to improve the predictability of the temporal behavior especially for high system-load situations and therewith to improve the reliability of the integrated system. For this purpose further detail is included on the problem definition and a set of requirements for a resource arbiter are specified. A prototype is realized based on those specifications, which is also presented.

II. PRIORITIZED ACCESS CONTROL

The arbitration of resource access for shared resources is not a new field of research [12]. Generally speaking, the management of resource access has to be done at some point. This
can be realized by interrupting the processing of an accessor (e.g. preempting a low-prioritized thread within a singlecore environment) or at the other extreme: delegated to the targeted system’s user (e.g. multiple different audio streams consumed by the user in parallel). If the arbitration has to be predefined, based on the system requirements the access control needs to be reflected by the system. With multiple computational resources and accessors this control has to be provided to the developer, respectively the integrator. Hence, the utilization of MC systems in combination with component-based systems requires a practicable solution to affect the temporal order of concurrent access. The SW components described here are developed independently, but depend on each other due to functional relationships. This requires efficient communication facilities, provided by the use of shared memory regions and adequate mechanisms for synchronization like semaphores, mutual exclusions and condition variables. SW frameworks can provide the necessary abstraction and support for the development process and improve the maintainability of the targeted system [13]. Such an abstraction may include a usable and domain-specific interface for concurrent and parallel processing, e.g. by providing capabilities to define EDs [11] and their priorities to enforce temporal requirements. Regarding the access of shared resources, a framework should also enable an integrator to define the temporal order of accessing these to improve the predictability of the system’s behavior. In relation to the exemplary domain of ICM systems, the targeted system has to provide the functionality in a coherent and uniform way in agreement with the vehicle’s user interface design. This supports achieving the goal of providing the perception of an ensemble in one piece. This also includes the predefined behavior of the system.

A. Scope of application

The approach relies on a Symmetric Multiprocessing (SMP) system. Therewith additional costs are saved in comparison to system designs that are based on virtualization or Asymmetric Multiprocessing (AMP) and with respect to computational power and memory usage due to multiple instances of operating systems. However, the proposed approach is also portable to such architectural designs. Furthermore, it may be that they are also required. Within AMP and virtualization based systems concurrent access to shared resources might be necessary, depending on the integration density and hence the shared use of common resources.

Further, the static scheduling of tasks is presumed to improve predictability and therefore maintainability due to simplified analyze capabilities during runtime. Additionally, these can be principally processed within their Worst Case Execution Time (WCET), although this condition might be limited to a certain threshold due to the mix of time- and event-triggered task characteristics. This means the SW system is a “schedulable taskset” as long as the concurring access to shared resources is not considered and the system is not operating within a high-load scenario. This implies that during such a high system-load neither low prioritized tasks nor low prioritized access is scheduled for computation. The use of dynamic scheduling (e.g. based on deadlines calculated during runtime) would improve the efficiency with regard to processing cores and resource utilization, but decrease predictability. Within the context of ICM systems that employ various tasks of differing importance, triggered either by time or events, and clustered into components that are developed in parallel by independent organizations, the need for predictability prevails to foster a deterministic temporal behavior of the overall system. This is further supported by the problematic and costly maintenance for vehicular systems after they have left the production line.

The shared resources addressed here exclude the multiple available CPU cores, but include I/O devices (e.g. automotive fieldbus connections like CAN and MOST, serial connections, files). It is further presumed that these devices are utilized using operating systems and available HW drivers.

B. Requirements

Based on the problem statement the following requirements for an arbiter can be derived:

REQ-1 The access latency for a shared resource is predictable.
REQ-2 A change of third-party SW is not necessary.
REQ-3 The access to shared resources can be temporally ordered using static defined priorities.

Although this enumeration is not complete, it defines the most essential architectural driver for the implementation of a resource arbiter.

C. Related work

The Automotive Open System Architecture (AUTOSAR) [14] is the outcome of a consortium consisting of automobile manufacturers, suppliers and producers of development tools. It is a standardized architecture, development approach and application programming interface (API). With release 4.0 it also supports MC platforms. The main focus of AUTOSAR is on mechanisms regarding the communication between applications running on different processor cores. The utilization of shared resources by applications which are deployed on different cores is not supported [15, p.45]. This limits the degree of freedom regarding the structuring of SW components and is thus a disadvantage for systems consisting of many components, e.g. ICM systems.

The ACTORS project [16] addresses embedded SW intensive systems in combination with resource utilization and high demands regarding adaptability and efficiency. This problem domain can also be mapped to ICM systems. The project proposes virtualization techniques for SW isolation to improve predictability and reliability. The resource management is implemented in user-space and adapts the resource reservation dynamically during runtime while considering optimal system occupancy. Although this promises good utilization of the system’s resources, the resource management does not take into account semantic dependencies in relation to the desired temporal system behavior. This could be achieved through the allocation of static priorities that are defined by an integrator.
Nesbit et al. predict that the available mechanisms and strategies for managing the resources of future MC systems will be insufficient. They further describe that a shared utilization of resources by tasks executed simultaneously could lead to unpredictable individual durations of the respective threads related to the involved tasks. This may include the violation of requirements and associated Qualities of Service (QoS) [17]. Similar to the ACTORS project, they propose a spatial separation by use of virtualization and feedback channels for the adaptive management of resource utilization during runtime. Their main focus is on the computational resources of the underlying HW platform.

Waldspurger et al. identify the conflation of concurrent access to physical HW as a central challenge within the context of I/O virtualization [18]. This includes prioritization and arbitration. They specify a minimal additional overhead for the redirection as critical. Although this is unquestioned, one of the most important characteristics of a resource arbiter within the following "overhead" is regarded as secondary in relation to the predictability to fulfill temporal requirements during high system-load situations.

III. ARBITRATION OF I/O ACCESS WITHIN USER-SPACE

The specified requirements were realized and validated for applicability by use of a prototypical implementation, the Shared Resource Arbiter (SHARB). A preliminary discussion of the general operation accompanied by an early proof-of-concept of the applied techniques is presented in [19]. However, this work had to be substantially refactored with a focus on applicability and thus evolved to the current implementation, presented in the following section. Therefore, SHARB utilizes OpenICM [20] as SW infrastructure. OpenICM is an academic SW framework maintained at the ICM Labs of the University of Applied Sciences Darmstadt. It implements the concepts described in [13] by with the use of the Portable Operating System Interface (POSIX) API. OpenICM does not yet provide any facilities for shared resource arbitration apart from computational resources. However, it provides a mature API that abstracts from an underlying operating system with a focus on ICM systems. This includes features to modularize a SW system into parallel computed components with respect to different priorities and an efficient asynchronous intercommunication. This means SHARB does not only make use of the existing facilities of OpenICM, but it is also a suitable enhancement to OpenICM to provide the necessary abstraction for a predefined temporal behavior of the concurrent access to shared resources.

A. Architecture constraints and design decisions

With consideration of REQ-1 SHARB avoids the use of dynamic memory and employs efficient communication facilities for internal synchronization and data transfer. Therefore, the abstractions provided by OpenICM are utilized for shared memory, message queues and the parallel execution of tasks.

The interface to the application layer on top of SHARB as well as the interface to the operating system below conforms to the POSIX API to achieve portability. With the use of

the library interposition mechanism [21] the arbitration layer introduced with SHARB is hidden to SW components that access managed shared resources. Applying this mechanism, also referred to as interposing, introduces an intermediate layer that provides capabilities to modify, prevent or substitute the functionality of referenced libraries. This appears transparent to both the caller (the application) and the callee (library), only based upon the configuration of the loader and runtime linker. The loader overloads resource-access relevant symbols during runtime for the dynamic binding with the use of symbols provided by the interposing resource arbiter. This means SHARB forms an additional layer on top of the operating systems and system libraries to intercept certain calls, reinterpret those and redirect them to available system libraries where appropriate. This obviates any functional changes regarding the access to resources from the application layer. Hence, through the use of SHARB no modifications to the already existing SW are necessary. This means there is no recompilation of supplied binary SW required, with respect to REQ-2.

Further, no change within the layer of the operating system is necessary because SHARB operates in user-space. In combination with its conformance to POSIX this eases a port to other system platforms.

Through OpenICM as an infrastructural SW framework for the application layer, the creation of threads is abstracted and unified. This includes the association of contextual data of a particular SW component with the thread by use of the POSIX API. Thereby SHARB is able to identify the context of a particular SW component by the implicit identifier of an accessing thread. In combination with the identifier of the accessed resource, SHARB determines the priority of a certain access based on statically defined priorities as part of the configuration associated with the SW components’ contextual data.

B. Architecture and functional principle

In the following section the internal functional principles of SHARB are illustrated to describe the arbitration of concurrent access to shared resources. The essential architectural components are depicted in Figure 1. Application SW components are EDs, based on the concepts as proposed in [11]. For the prioritization of resource accesses the Device Manager (DM) delegates all relevant calls from EDs to a Service Driver (SD). Relevant calls include primitives like open, read, write and close, as well as those used for the control and initialization
of resources, as defined within the POSIX API. For each association between an ED and a resource, a dedicated SD is created, which is executed as a thread within the context (process frame) of the associated ED. A SD is connected to a Device Instance (DI) to actually perform the access to the resource abstraction provided by the OS. The details about the access to a certain resource are encapsulated within its associated DI, which is executed as a thread in an independent context.

Within this context a thread pool manages standby worker threads, which receive prioritized jobs (DI-job) triggered by events. A DI-job represents a read or a write access to a certain resource. With the use of the thread pool, additional costs for thread creation can be avoided during runtime. The communication between DM, SD and DI relies on shared memory, POSIX message queues and binary semaphores. The prioritization is realized using static thread priorities given to the DI-jobs and the SDs, whereas all DI-jobs and SDs associated to a common DI are bound to a common processor core. This "cpu affinity" supports both a deterministic order of resource access and an efficient communication within SHARB through the use of a common cache memory hierarchy.

In Figure 4 three exemplary EDs are depicted which access two resources. It illustrates the co-operation of multiple accessors competing for shared resources (R). For this scenario the following assumptions are given:
1) ED\textsubscript{1} utilizes R\textsubscript{A}
2) ED\textsubscript{2} utilizes both R\textsubscript{A} and R\textsubscript{B}
3) ED\textsubscript{3} utilizes R\textsubscript{B}

This means, R\textsubscript{A} and R\textsubscript{B} are shared resources. The order of access is not defined for EDs executed in parallel (e.g. each on a different core).

SHARB introduces a new abstraction layer, as depicted in Figure 5.
- For each shared R, a separate DI is created during initialization.
- For each association between ED and R, a separate SD is created during runtime.
- For each DI, n connections are handled during runtime.

Derived from the example introduced in Figure 5, the effective threads for the access of ED\textsubscript{1} and ED\textsubscript{2} are depicted in Figure 6 with their SHARB Priority Levels (SPL). The SPLs are a concept to order the access-priority in dependency of the relation between a certain ED and R. This concept is implemented by the OS’s task scheduler. SPL-0 supersedes all other SPLs and is reserved for the management of the effective resource access, realized by DIs. The subsequent SPLs correspond to the respective access priority derived from the relation between ED and R. A single SPL represents a task queue. The task queues related to a single resource must be

Fig. 2. Main activities to process a call to open()

Fig. 3. Main activities to process a call to read()
bound to a single processing core. Figure 6 illustrates both the assignment of threads to SPLs and the partitioning into distinct scheduling domains using core \( c_1 \) and core \( c_2 \). Therefore, the example shown in Figure 4 is refined:

4) \( ED_3 \) has higher access priority than \( ED_2 \) for \( RA \)
5) \( ED_3 \) has higher access priority than \( ED_2 \) for \( RB \)

For the OS’s task scheduler the threads of the arbiter are stringed on different priority scheduling queues, as depicted in Figure 7. The DI-jobs represent effective access calls to the corresponding resource (e.g. \( ED_1 \) is accessing RA two times; \( ED_2 \) is accessing RA three times; both are assigned to core \( c_1 \)). The OS’s priority levels utilized for the SPLs may supersede the priorities assigned to EDs which are collocated on the same processing core. Although this is not a prerequisite, it prevents unwanted temporal interference between SHARB and the applications. Alternatively, dedicated processing cores might be reserved for SHARB, e.g. general-purpose cores with a reduced clock rate or features within a heterogeneous MC HW architecture.

**C. Impact**

With the described approach the access to resources can be prioritized to achieve a more predictable temporal behavior. A prerequisite is the definition of a static configuration, which provides the capability to specify priorities for resources (and groups of them) in need of the accessing SW components. An access priority is not specified for an ED or a resource, but for a combination of both of them. Hence, an ED could make use of different priorities for different resources. This offers the required degree of freedom to the integrator, which is necessary to achieve a predefined temporal system behavior without the need to modify the implementations of the accessing SW components.

Further, it is possible to arbitrate the access to a selected set of resources. This implies that additional costs for the management do only occur when necessary. With this selective deployment of SHARB the overhead introduced through the additional indirection is kept to a minimum, with the focus being on the overall system. It is also possible to combine SHARB with the implementation of another resource arbiter through selective resource configuration.

Additionally, the attributes and strategies used to access a specific resource can also be optimized using the decoupling of SW components and the resources. This means for example, the use of new HW with legacy SW is feasible without the need to change the SW. Apart from the demultiplexing of accesses, SHARB can enforce an optimal configuration for the abstracted resources or HW devices (e.g. rate of transmission, byte order, synchronization methods) through the intercepting of control calls. This might provide further freedom during the integration process.

Moreover, the described abstraction provides the capability to realize different access strategies with regards to the resource or the accessing SW component. This applies to read access in particular. The following enumeration lists a selection of such strategies for stream-oriented resources:

- A call of read() starts at the position after a previous read() of an access of the same component.
- A call of read() starts at the position after a previous read() of an access of any component.
- A call of read() returns the most recent data, independent of any previous access.

Furthermore, the abstraction allows the implementation of filters which may manipulate, discard or add transmitted data that is transparent for the application layer. In addition, the latter could be used to substitute, simulate or emulate unavailable resources in the early development stages and therefore to reduce risks during system integration.

**IV. Evaluation**

In the following section the evaluation of the proposed approach based on the described prototypical implementation is discussed.

**A. Test setup**

The test system is an x86 MC HW platform, based on two Intel Xeon E5504 processors with four cores each. Even
though equivalent performance is not available for current ICM systems, the test environment’s characteristics can be compared with next generation head-units. The operating systems used are GNU/Linux 3.6, GNU/Linux 3.6 patched with PREEMPT_RT real-time extension (in the following referred to as PREEMPT_RT), and QNX Neutrino 6.5. The shared resource is a dedicated kernel module, which implements a character oriented device driver (char dev) that returns a given number of bytes into a target buffer on a POSIX read request. To simulate latency within a real resource driver and respectively a real device, the module responds to read access with a fixed latency of 50 ms, independent of the number of requested bytes or repetitions. Although, a real driver or device might not answer using a fixed latency, this testing environment prevents additional variance during the time measurements. This leads to reproducible results. The scope of the evaluation is reduced to the testing of the prototype (and not any driver or device implementations). Further, the module behaves as a blocking device, which implies only one accessor can read at a time.

B. Results - prioritization

To evaluate the correctness of the approach, the time to read a predefined number of bytes is measured. Therefore, two components (ED\textsubscript{1} and ED\textsubscript{2}) are bound to different processor cores. These components are implemented to read from the described device driver (RA) concurrently and start simultaneously. The dependencies are illustrated in Figure 8. The coordination of the simultaneous start is implemented through the use of a semaphore, triggered for both ED\textsubscript{1} and ED\textsubscript{2}. For ED\textsubscript{1} a high and for ED\textsubscript{2} a low resource access-priority is configured. Due to the implementation using the library interposing mechanism the arbiter can be activated without much effort by adapting the search path of the dynamic linker. The measurement is performed as a loop to collect values from 10,000 runs.

The results of the measurements show that the high-prioritized ED\textsubscript{1} successfully completes read before the low prioritized ED\textsubscript{2} gets access for every test run when SHARB is activated. With the arbiter deactivated, the number of test runs where ED\textsubscript{1} finishes before ED\textsubscript{2} compared with where ED\textsubscript{2} finishes before ED\textsubscript{1} is uniformly distributed.

In addition to the previously described empirical methods to prove predictable access prioritization, a theoretical consideration of SHARB is provided in the following.

Therefore a set of parameters to define points in time and periods of time are provided in Table I. Furthermore, Table II lists eight permutations of task arrivals related to the concurrent access of different EDs using different priorities within a common scheduling domain. Hence, they cause preemption of, or introduce latency to tasks. The table also specifies the conditions of the concurring tasks, using the symbols of Table I to support the clear distinction between computing and the arriving task. Corresponding to Figure 6, tasks related to ED\textsubscript{1} are of higher access priority than tasks related to ED\textsubscript{2}. This means SD\textsubscript{1.A} and DI-job\textsubscript{1.A} are configured for a high SPL, whereas SD\textsubscript{2.A} and DI-job\textsubscript{2.A} are configured for a low SPL, respectively. A more trivial scenario with only one accessing ED is not considered here, because in such a case access prioritization has no effect on the tasks’ computation order.

In order to show the correctness of SHARB three exemplary scenarios selected from Table II are detailed in the following section. They show either the delay in an already computing task’s execution time or a delay in the start of the computation (latency). For reasons of clarity and comprehensibility any additional latency caused through the tasks’ context switches was neglected. Further, the illustration is reduced to the two fundamental types of task derived from the architectural elements of SHARB. These differ in their behavior regarding pre-emption: The SDs are pre-emptible whereas the DI-jobs are non-pre-emptible. The latter is caused through the DI-jobs main task of accessing the actual resource, which implies the current access must be finished before switching to a
subsequent DI-job (although this is highly dependent on the type of the actual resource and therefore leaves space for further optimization). Following previous notations, Figures 9, 10 and 11 visualize tasks separated by their configured SPL. The focus is on the scheduling order of the subsequent tasks.

Figure 9 depicts the occurrence of a low-priority access during a computing high-priority access. In particular, scenario 6 of Table II is addressed here. The low-priority tasks SD$_2A$ and DI-job$_2A$ are delayed until all high-priority tasks are finished. This implies an introduced latency for SD$_2A$ which may affect ED$_2$:

$$L_{SD2A} = f_{DI-job1A} - a_{SD2A}$$

Figure 10 depicts scenario 3. A high priority SD (and subsequent DI-job) pre-empts a low priority SD. In particular, the task-scheduler of the OS pre-empts SD$_2A$ and starts the computation of SD$_1A$ immediately after the arrival of SD$_1A$. SD$_2A$ resumes computation after SD$_1A$ and the subsequent DI-Job$_1A$ finish computation. This causes a delay which may affect ED$_2$:

$$D_{SD2A} = f_{DI-job1A} - a_{SD1A}$$
$$E_{SD2A} = D_{SD2A} + E_{SD2A}$$

Figure 11 depicts scenario 7. A high-priority DI-job arrives while a low priority DI-job is computing. The DI-job$_2A$ locks resource R$_A$ until the current job is finished. DI-job$_1A$ is delayed until the low-priority DI-job$_2A$ releases R$_A$. The maximum latency for a high-priority DI-job is the computation time of a single low-priority DI-job. Respectively, this may affect ED$_1$:

$$L_{DI-job1A} = f_{DI-job2A} - a_{DI-job1A}$$

To summarize, the temporal order of the concurring access to the shared resource is configurable with SHARB. Hence, with its use the predictability of the behavior increases and REQ-3 is met.

C. Results - temporal overhead

For the temporal overhead a single ED reads from a single R. The results provided are based on 90,300 measurements. These were taken natively (without SHARB/not prioritized) and arbitrated to visualize the costs in terms of temporal overhead. The amount of data (1-256 byte) and the number of repetitions (1-30) iterated during the data collection, whereas for each permutation 25 measurements were recorded to achieve reliable and evaluable results. In this context repetitions are related to the number of calls to read() between an open() and close(). This means, a single read consists of the call “open() - read() - close()”, whereas an access with eight repetitions consists of the calls “open() - 8*read() - close()”.

Figures 12 and 13 visualize the results as a percentage in relation to non-arbitrated access for Linux, PREEMPT_RT and QNX. The respective overhead results from the mean of the...
absolute time measurements for an arbitrated access, less the mean of the absolute time measurements for a native access.

Figure 12 shows the overhead as the mean of different data sizes in relation to the repetitions. For a few repetitions a significant percentage of overhead can be observed. This is caused by the initial setup phase to establish the internal administration and communication infrastructure. These temporal costs amortize when accessing the resource five times or more. Further, this effect can be mitigated by the pre-initialization of SHARB (the evaluation was performed by initialization on-demand). Nevertheless, the temporal overhead levels off at about 20 percent for QNX and Linux.

Figure 13 shows the effect of SHARB related to different amounts of data, whereas for each data size the mean duration based on different repetitions of the read() access is measured. The overhead with QNX is significantly higher compared to the use with PREEMPT_RT. However, the temporal overhead with QNX is much more stable in variance and therefore more predictable (distribution with QNX is 0.14%, compared to 1.85% with PREEMPT_RT, and 3.14% with Linux).

Further conclusions about the predictability regarding access latency are founded on the standard deviation. Figures 14 and 15 show the standard deviation of the time necessary to access a resource using SHARB in comparison to native access, which is also in relation to the repetitions and the amount of data respectively. Nevertheless, the measurements also show that the latency is scattered only slightly more with activated SHARB and is independent of size and repetitions. In particular, the standard deviation built from the mean of different sizes for the different number of reads (as shown in Figure 14) illustrates a predictable temporal behavior that is independent of the repetitions of a call to read(). For example, both the distribution of the relative standard deviation with SHARB on QNX and also the standard deviation itself is smaller than the measures of SHARB on Linux. However, Linux patched with PREEMPT_RT still performs comparably to the measures achieved with QNX.

Figures 16 and 17 provide details regarding the absolute overhead (mean duration arbitrated less mean duration native). They correspond to Figures 12 and 13 for the observations regarding the comparison of QNX, PREEMPT_RT and Linux: On QNX SHARB behaves more deterministically, however introduces more overhead.

Although the additional costs may appear high considering the efficiency of the targeted system, the predictability of the access order prevails with respect to the deterministic behavior of the overall system. This is the main objective of SHARB.

V. CONCLUSION AND OUTLOOK

The integration of different SW components into a common platform implies demanding challenges. MC HW platforms can provide support, but also create new challenges due to
the additional parallelism. In contrast to the concurrent multi-tasking (quasi-parallelism), concurrent resource access on a MC HW platform is not predictable in terms of temporal order. This article motivates the necessity to arbitrate the access to shared I/O resources. Therefore, the example of component based ICM systems is used, which are developed with a high division of labor. Requirements were defined, which found the basis for an arbiter: SHARB. It introduces a thin architectural layer in between application components and system libraries for accessing particular resources. SHARB appears transparent to both the applicational layer and the libraries underneath, which obviates the need to change either the applications or libraries. Furthermore, SHARB resides within user-space and therefore does not require any changes to the OS. A prototypical implementation which supports the verification of the presented approach for arbitration. The applicability and introduced overhead of the prototypical implementation were theoretically and quantitatively evaluated. For the latter, different target OSs are compared.

Although SHARB is targeted for use in SMP based ICM systems utilizing a single OS, the approach is also portable to other usage contexts and architectures with similar requirements. This includes architectures focusing on a more strict isolation of components. These may also require the management of concurrent access to shared resources, based on their density of integration or grade of parallelism. This refers, for example, to AMP systems [8, p.168] and architectures based upon virtualization using hypervisors [22]. For certain I/O resources within the context of multimedia, including video and audio, arbitration is still subject of research [23].

REFERENCES


Andreas Knirsch is currently a PhD student in the Centre for Security, Communications and Network Research at Plymouth University, UK. He is further a researcher with the In-Car Multimedia Labs at h_da Hochschule Darmstadt, DE where he received his MSc in Computer Science in 2009. Prior to academia, he gained several years industrial experience in SW development. His research interests include SW frameworks, system partitioning, and operating systems for ICM.

Pierre Schnarz is currently a PhD student in the Centre for Security, Communications and Network Research at Plymouth University, UK. He is further a researcher with the In-Car Multimedia Labs at h_da Hochschule Darmstadt, DE where he received his MSc in Computer Science in 2012. His research interests include multi-operating systems, trusted inter-OS communication and synchronization, and security requirements of ICM environments.

Joachim Wietzke is a Professor of Informatics and heads the In-Car Multimedia Labs at h_da Hochschule Darmstadt, DE. Prof. Wietzke has been involved in a number of industrial projects related to SW project task forces as well as next generation ICM systems. Prior to his current position, he gained extensive industrial experience in head positions of SW development units at Bosch, Balaupunkt, and Harman/Becker. His research interests include next generation ICM systems and HMIs.