International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue IX, September 2017- Available at www.ijraset.com

New Trends in Real Time Operating Systems

Vijaybhai Parmar[1]
[1]*C U Shah University*

Abstract: Most of today's embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Still timely responses to events have to be provided within precise timing constraints in order to guarantee a desired level of performance. Hence, embedded systems are, by nature, inherently real-time. Moreover, most of embedded systems work under several resource constraints, due to space, weight, energy, and cost limitations imposed by the specific application. As a consequence, efficient resource management is a critical aspect in embedded system that must be considered at different architecture levels.
The objective of this document is to introduce Real Time Operating System to new comers the major research trends identified. After describing the characteristics of modern embedded applications, the paper presents the problems of the current approaches and discusses the new research trends in real time operating systems and scheduling emerging to overcome them.

I. INTRODUCTION

An Operating System (OS) viewed as organized collection of software extensions of hardware, consisting of control routines for operating a computer and for providing an environment for execution of programs. The range and services provided by an operating system depend on a number of factors.

Timeliness is the single most important aspect of a real – time system. These systems respond to a series of external inputs, which arrive in an unpredictable fashion. The real-time systems process these inputs, take appropriate decisions and also generate output necessary to control the peripherals connected to them. As defined by Donald Gillies "A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time in which the result is produced. If the timing constraints are not met, system failure is said to have occurred."

It is essential that the timing constraints of the system are guaranteed to be met. Guaranteeing timing behaviour requires that the system be predictable.

Most real -time systems interface with and control hardware directly. The software for such systems is mostly custom developed. Real time Applications can be either embedded applications or non-embedded (desktop) applications.

II. DESIGN ISSUE OF REAL TIME SYSTEMS: THE TIMING REQUIREMENTS

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced as per definition. In traditional non-real-time computer systems, the performance goal is throughput: as many tasks should be processed as possible in given time period. Real-time systems have a different goal to meet: as many tasks as possible should be executed so, that they will complete and produce results before their time limit expires. In other words, the behaviour of real-time system must be predictable in all situations.

A. Predictability

To achieve predictability, all components of the real-time system must be time bounded. A predictability of the system depends on many different aspects.

1) *Hardware*: The computer hardware must not introduce unpredictable delays into program execution. For example, caching and swapping as well as DMA cycle stealing are often problematic when determining process execution timing.
2) *Software*: An operating system must have a predictable behaviour in all situations. Thus, special microkernel operating systems like the Chorus microkernel have been designed for real-time purposes.
3) *Implementation*: Also traditional programming concepts and languages are often not good for real-time programming. No language construct should take arbitrary long to execute, and all synchronization, communication, or device accessing should be expressible through time-bounded constructs.
4) *Human Factor*: However, despite all these real-time requirements could be solved, a human factor the real-time programmer can always cause unpredictability to the system. To assist the programming process, numerous methods have been produced for real-tim system design, specification, verification, and debugging.

B. Temporal consistency

Typically, a real-time system consists of controlling system and a controlled system. The controlled system can be viewed as the environment with which the computer interacts. The typical real-time system gathers information from the various sensors, process information and produce results. The environment for real-time system may be highly indeterministic. Events may have unpredictable starting time, duration and frequency. However, real-time system must react to all of these events within pre specified time and produce adequate reaction.

To guarantee, that a real-time system *has always a correct view of its environment*, a consistency must be maintained between them. The consistency is time-based. The controlling system must scan its environment fast enough to keep track changes in the system. The adequate fastness depends on application. For example, sensing a temperature needs often slower tempo than sensing a moving object.

The need to maintain consistency between the environment and the controlling system leads to the notion of temporal consistency. Temporal consistency has two components:

1) Absolute consistency between the state of the environment and the controlling system. The controlling system's view from the environment must be temporally consistent; it must not be too old.

2) Relative consistency among the data used to derive other data. Sometimes, the data items depend on each other. If a real-time system uses all dependent values, they must be temporally consistent to each other.

There are several possibilities to maintain temporal consistency. The *state of the environment* is often scanned in periodical basis and an image of the environment is maintained in the controlling system. A *timestamp methodology* is often used to figure out validity of the system's image of the environment.

a) *The periodic behaviour*: The periodic behaviour means, that a task must be executed within pre specified time intervals.

b) *The aperiodic behaviour*: When a task has an aperiodic behaviour, it will execute, when an external stimulus occurs.In typical

c) real-time systems, both types of tasks exist. However, all aperiodic tasks can always be transformed to periodic.

3) *Static table-driven approaches:* These perform static schedulability analysis and the resulting schedule (table) is used at run time to decide, when a task must begin execution. This is a highly predictable approach, but it is very inflexible, because a table must always be reconstructed, when a new task is added to the system. Due to predictability, this is often used when absolute hard deadlines are needed.

4) *Static priority-driven pre-emptive approaches:* These perform static schedulability analysis, but unlike in the previous approach, no explicit schedule is constructed. At run time, tasks are executed "highest priority first". This is a quite commonly used approach in concrete real-time systems.

5) *Dynamic planning-based approaches:* Unlike the previous two approaches, feasibility is checked at run time. A dynamically arriving task is accepted for execution only if it is found feasible.

6) *Dynamic best effort approaches:* Here no feasibility checking is done. The system tries to do its best to meet deadlines, but a task may be aborted during its execution.

Unfortunately the most scheduling problems are NP-complete. However, many good heuristic approaches have been presented. Thus, numerous algorithms have been introduced to support these scheduling paradigms. Algorithms are either based on the single scheduling paradigm or they can spread over several paradigms. These algorithms include least common multiply (LCM) method, earliest deadline first (EDF) method, rate monotonic (RM), and many others.

III. DESIGN ISSUE OF REAL TIME SYSTEMS: SCHEDULING

Scheduling decisions are made based on the current inputs from the sensors and the current status of the controlled system. The events can occur very unexpectedly. Still decisions must be made within a specified time.

A. Scheduling paradigms

The simplest real-time scheduling method is not to schedule at all as in many real-time systems, only one task exists. In real-time systems, several simultaneous tasks can be executed. Every task may have different timing constraints and they may have a periodic or an aperiodic execution behaviour.

B. Real time CPU scheduling

1) *Traditional Algorithms Rate-monotonic Scheduling:* It uses a static priority policy with preemption. Each periodic task assigns priority inversely based on its period. To assign a higher priority to task that requires the CPU more often.

2) *Earliest-Deadline-First Scheduling:* Dynamically assigns priorities according to deadline.
3) *Proportional Share Scheduling:* Scheduler operated by according T shares among all applications.
4) *Pthread Scheduling*: It is a programming API.

C Minimized latency

There many types of latencies affect performance of real time systems:

1) *Interrupt latency:* This refers to the period of time required from arrival of interrupt to start the routine that services the interrupt.
2) *Dispatch latency:* This refers to the amount of time required for scheduling dispatcher to stop one process and start another.

D Priority inversion problem

In a multitasking environment, the shared resources are often used. The usage of these resources must be protected with well-known methods, like semaphores and mutexes. However, a priority inversion problem arises, if these methods are used in real-time system with a preemptive priority-driven scheduling.

E Priority Inheritance Protocol

The basic idea of priority inversion protocols is that when a task blocks one or more high priority tasks, it ignores its original priority assignment and executes its critical section at the highest priority level of all the tasks it blocks. After exiting its critical section, the task returns its original priority. Priority inversion problem can be solved with a priority inheritance protocol. To support timeliness in a real-time system, a special real-time task scheduling methods have been introduced. Traditional real-time research has been concerned to uniprocessor scheduling. As a complexity and scale of real-time system grows, the processing power of real-time system is increased by adding new processors or by distribution. These issues introduce several new concepts to scheduling research; numerous schemes have been introduced for multiprocessor and distributed scheduling.

IV. TODAY'S REAL TIME SYSTEMS AND PROBLEMS WITH CURRENT APPROACHES

Most of today's real time embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Still timely responses to events have to be provided within precise timing constraints in order to guarantee a desired level of performance. Hence, embedded systems are, by nature, inherently real-time. Moreover, most of embedded systems work under several resource constraints, due to space, weight, energy, and cost limitations imposed by the specific application. As a consequence, efficient resource manage-ment is a critical aspect in embedded systems that must be considered at different architecture levels.

A Important properties of the environment

The use of computer controlled systems has increased dramatically in our daily life. Processors and microcontrollers are embedded in most of the devices we use every day, such as mobile phones, PDAs, TVs, DVD players, cameras, cars, dishwashers, etc. This trend is expected to continue in the future. Several research projects on ambient intelligence, pervasive systems, home automation, and ubiquitous computing, aim at integrating computers in our environment even more in a way that they are hidden. Most of these devices share the following important properties:

1) *Limited Resources:* Several embedded devices are de-signed under space, weight, and energy constraints imposed by the specific application. Often they also have cost constraints related with mass production and strong industrial competition. As a consequence, embedded applications typically run on small processing units with limited memory and computational power. In order to make these devices cost-effective, it is mandatory to make a very efficient use of the computational resources.
2) *Real-time constraints:* Most embedded devices inter-act with the environment and have demanding quality specifications, whose satisfaction requires the system to timely react to external events and execute computational activities within precise timing constraints. The operating system is responsible for ensuring a predictable execution behaviour of the application to allow an off-line guarantee of the required performance.
3) *Dynamic behaviour:* The complexity of embedded systems is constantly increasing and several applications consist of tens or hundreds of concurrent activities that interact with each other and compete for the use of shared resources. In addition, the behaviour of some activities depends on sensory data inputs, which can hardly be predicted in advance. Finally, low-level

architecture features, such as caching, pre-fetching, pipelining, DMA, and interrupts, although enhancing the average computer performance, introduce a non deterministic behaviour on tasks execution, making the estimation of worst-case computation times very unpredictable. As a consequence, the overall workload of complex real-time applications is subject to significant variations that cannot be easily predicted in advance. The combination of real-time features in tasks with dy-namic behaviour, together with cost and resource constraints, creates new problems to be addressed in the design of such systems, at different architecture levels. The classical worst-case design approach, typically adopted in hard real-time systems to guarantee timely responses in all possible scenarios, is no longer acceptable in highly dynamic envi-ronments, because it would waste the resources and pro-hibitively increase the cost.

4) Adaptive: Instead of allocating resources for the worst case, smarter techniques are needed to sense the current state of the environment and react as a consequence. This means that, to cope with dynamic environments, a real-time system must be *adaptive*; that is, it must be able to adjust its internal strategies in response to a change in the environment to keep the system performance at a desired level or, if this is not possible, degrade it in a controlled fashion.

B. Specific support at different levels for architectur

Implementing adaptive embedded systems requires spe- cific support at different levels of the software architecture. The most important component affecting adaptivity is the kernel, but some flexibility can also be introduced above the operating system, in a software layer denoted as the *middle-ware*. Some adaptation can also be done at the application level; however, it potentially incurs in low efficiency due to the higher overhead normally introduced by the application level services. Normally, for efficiency reasons, adaptation should be handled at the lower layers of the system architec-ture, as close as possible to the system resources. For those embedded systems that are distributed among several com-puting nodes, special network methodologies are needed to achieve adaptive behaviour and predictable response.

C. Novel applications requirements

The complexity of embedded systems is constantly in-creasing. While in the past embedded systems were synonym of 8-bit processors and small memory footprint, most of today's systems are developed on 32-bit processors with several megabytes of memory, and some of them include advanced multimedia features. For example, a modern mo-bile phone typically consists of several million lines of code with use-cases involving large number of concurrent activi-ties. Utilizing available hardware and software resources in an optimal fashion is crucial both to save costs and to keep the competitive edge.

In the large domain of consumer electronics, most of the application software includes at least three types of activi-ties, with different characteristics and requirements with re-spect to timing and resource sharing:

1) Control software: Control software is typically implemented by periodic tasks and uses only a small fraction of the available resources. Sensory acquisition and control tasks are subject to hard timing requirements that must be guaranteed off line in all operating conditions.

2)Media processing software: Media processing software is typically data or throughput driven, and is a major consumer of hardware resources (processor and network bandwidth). Audio/video processing and graphics applications are the main examples in this category. They are normally treated as soft real-time aperiodic tasks with qualityof-service (QoS) requirements. Due to the large re-source consumption, achieving high resource utilization is of crucial importance.

3)Interaction software: Interaction software is very complex and vastly increasing in size and complexity. Typical examples are electronic program guides, internet browsing, photo/music browsing, broadcast enhancement (for example, player info and statistics in sports games). In a high-end TV set, the total code size currently approaches 4 Mbytes. For this software, the timing requirements are interactive-response requirements.

D. Approach to achieve some level of adaptive ness today

A problem in handling such different types of activities is to manage the available resources so that each class of tasks can meet its specific quality constraints. Adding or removing features may cause a system to fail.

1) For complex and dynamic systems, exhaustive design and testing of every possible use case is not tractable. Therefore, tools and metrics for expressing and handling resource requirements are essential in future system design.

*2)*Adding and removing features would become predictable, allowing configuring the system without worrying about unpleasant surprises.

E. Problems with the current approach

1) In spite of the increased systems complexity, real-time applications are mainly configured acting on task priorities, which usually express the importance of tasks.
This is for many reasons inadequate when configuring complex dynamic systems, because there are other system constraints that cannot be mapped into a set of priority levels. As a consequence, today, systems require extensive testing and tuning to operate optimally.

2) Another problem with priorities is that activities often consist of several tasks, which may play different roles in different scenarios, making the priority assignment even more difficult. Any attempt to group tasks together fails since priority is a global property and will always break any type of encapsulation.

3) When dealing with dynamic applications with variable execution requirements, the use of dynamic priorities schemes would give higher flexibility to the system, allowing better adaptivity and full resource exploitation. However, most of today's industrial products with dynamic behaviour are still based on fixed priority schemes and have very limited flexibility.

4) The main reason is due to the fact that they are built on top of commercial components that do not offer the possibility of being reconfigured at runtime. For example, at the operating system level, most of the internal kernel mechanisms, such as scheduling, inter-rupt handling, synchronization, mutual exclusion, or communication, have a fixed behaviour dictated by a specific policy that cannot be easily replaced or modified.

F. QoS management modern operating systems

The typical approach used today at the operating system level to affect the execution behaviour and achieve some level of adaptation is to modify task priorities. However, this method does not always succeed and it is not trivial to predict how the system performance will change as a func-tion of priorities. For example, increasing the priority of a task with long execution time could lead to an overload con-dition that would degrade system performance. Even de-creasing the priority of a task could create problems, since it would implicitly raise the relative priority of other tasks, so also leading to an overload situation. If tasks interact through shared resources, task priorities also affect the delays caused by blocking on critical sections. If the kernel uses a priority inheritance protocol for accessing shared resources,
changing the priority of a task at the "wrong" time instant could interfere with the protocol and cause very undesirable effects, such as priority inversion. This examples show that today's commercial operating systems are not suited for on-line adaptation because they do not provide explicit support for quality-of-service (QoS) management.
In particular, we illustrated the characteristics of modern applications; and explained the problems of the current industrial practice; on these basis we can study Challenges reports the most innovative research areas in operating systems and scheduling to overcome these limitations in next chapter.

V. CHALLENGES AND WORK DIRECTIONS

The most important mechanism in the operating system affecting adaptiveness is scheduling. Unfortunately, how-ever, the majority of today's commercial operating systems schedule tasks based on a single parameter, the priority. Re-cent research on flexible scheduling showed that a single parameter is not enough to express all the application requirements. A framework that contains new challenges and work directions is proposed as considering accountability of each area.

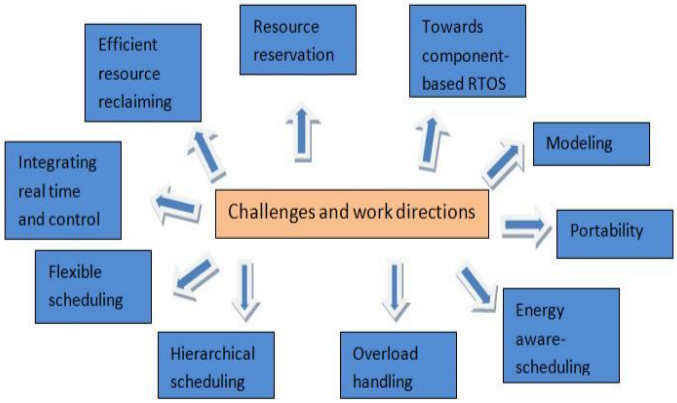

Figure: Framework for new trends in RTOS

A. Effective support to QoS management

In order to provide effective support to QoS management, modern operating systems should be:

1) Reflective. That is, they should reflect the application characteristics into a set of parameters, which can be used by appropriate scheduling algorithms to optimize system performance. For example, typical parameters that may be useful for effective task management include deadlines, periodicity constraints, importance, QoS values, computation time, and so on.

2) Resource aware. That is, they should give the possi-bility of partitioning the resources (e.g., the processor) among the existing activities based on their computational requirements. Such a partitioning would enforce a form of temporal protection that would prevent reciprocal interference among the tasks during overload conditions.

3)Informative. That is, they should provide information on the current state of execution to allow the implementation of adaptive management schemes at different levels of the software architecture. Any difference between the expected and the actual behaviour of a computation can be used to adjust system parameters and achieve a better control of the performance.

In previous chapter, we illustrated the characteristics of modern applications; and explained the problems of the current industrial practice; on these basis we can study Challenges reports the most innovative research areas in operating systems and scheduling to overcome these limitations.

To achieve these general objectives, further research is needed in several areas. They are illustrated in the following sections.

B. Resource reservation

The problems caused by priorities, mentioned in Section 3, could be solved by using a programming model that enables the designer to explicitly control the resources as-signed to a given activity at a given point in time. With reservation-based scheduling, a task or subsystem receives a real-time share of the system resources according to a given allocation policy.

Notice that, from a timing point of view, receiving a frac-tion of the processor is equivalent to executing on a private processor running at reduced speed. As a consequence, this approach isolates the execution of tasks running under re-source reservations, thus protecting the other activities from possible overruns (temporal protection).

Resource reservation can be efficiently implemented using a Constant Bandwidth Server (CBS), which is a service mechanism allocating a budget of units of time every period.

The ratio is denoted as the server bandwidth, defining the fraction of the CPU reserved to the task. If a task is handled by a CBS with bandwidth, it is guaranteed that it will never demand more than its reserved bandwidth, independently of its actual requests.

This is achieved by assigning each task a suitable (dynamic) deadline, computed as a function of the reserved bandwidth and its actual requests.

If a task requires to execute more than its expected computation time, its deadline is postponed by the server, so that its reserved bandwidth is never exceeded.

Currently, reservation-based scheduling is focused on the CPU only. However, a system wide approach is demanded, including other system resources, including memory, disk, and network. Pertinent research directions include adding support for resource reservation schemes in embedded systems. Here, a major challenge is to contain computational overhead in the implementation. One of the few existing kernels for small embedded systems supporting resource reservation is Erika Enterprise, developed by Evidence S.r.l. A further step in this area is providing a notion for real-time contracts that includes the rights and duties of the in-volved parties in a detailed way.

C. Efficient resource reclaiming

Although the resource reservation paradigm may solve many problems related to priority-based scheduling, its behaviour heavily depends on a balanced allocation of the available resources to the application tasks. In fact, if the amount of resource reserved to a task is less than required, that task will slow down too much, decreasing system performance. Conversely, if the amount of resource reserved to a task is too high, resources are wasted and the system will run with low efficiency (increasing the overall cost of the application).

However, providing an exact estimation of the resources needed by each system activity is a non trivial job, which re-quires heavy execution tests and specific tools for code anal-ysis. Nevertheless, the resulting estimations are usually af-fected by large errors (up to 20%). Therefore, an additional runtime reclaiming mechanism is required in the kernel to cope with wrong or imprecise resource reservations. The idea behind resource reclaiming is quite simple: whenever a task completes earlier than expected, so leaving part of a resource unused, this part is temporally kept in the system to be given to other tasks that may require more bandwidth.

It is worth observing, however, that resource reclaiming is an on-line mechanism that exploits early completions, hence it cannot always compensate for reservation errors. In the average, resource reclaiming is quite effective for compensating for small reservation errors, but it does not represent the solution for coping with large and systematic deviations in the execution behaviour of computational activ-ities. A few examples of reclaiming algorithms have been proposed in the literature.

D. Integrating real time and control

When application tasks have an extremely variable and unpredictable execution behaviour, feedback control theory can be used to estimate the current workload conditions and perform proper parameter tuning. Integration of real-time and control theory just begun to be studied and is a promising research area. The advantage of such as integration is twofold. From one hand, feedback control schemes can be used in the kernel to make the system more adaptive to unpredictable changes. On the other hand, real-time theory and schedulability analysis can be also considered during the design of control systems, to take into account jitter and delays introduced in control loops by resource contention and concurrent execution.

In the traditional approach to the analysis and design of computer control systems, controllers are assumed to execute in dedicated processors and these are assumed to be fast and predictable enough to meet all the application requirements. However, when resources such as processor time or network bandwidth are limited, the analysis and design of computer control systems is a challenging task: the resource limitations must be taken into account in the controller design stage, or the controlled system may exhibit unexpected behaviour.

For example, the criteria for scheduling tasks on proces-sors influences the timing of all tasks and can thus introduce timing variability (jitter) in the execution of control loops. These timing variations in the execution of control algorithms -which are allowed as long as the schedulability constraints are preserved -affect performance and possibly cause instability. This degradation appears because the controller execution violates the timing assumptions of classical discrete-time controller design theory, equidistant sampling and actuation.

On the other hand, trying to reduce jitter for control tasks by over-constraining the control task specification (e.g. by very tight deadlines) reduces the degradation of the controlled systems, but at the expense of finding feasible scheduling solutions for the entire task set.

These kinds of problems can be addressed using a com-bination of control and real-time scheduling principles. In-stead of separating the two aspects during design, control design and computer implementation have to be jointly con-sidered early in the design. A number of algorithms to com-bine real-time and control have been presented in the liter-ature in both areas of research, with different perspectives and objectives.

E. Flexible scheduling

Novel applications combine various types of tasks and constraints within the same system. The requirements on tasks may also change dynamically. While off-line guarantees are still essential for meeting minimum performance levels, different types of requirements and runtime changes are included in the system analysis, such as demands on quality of service (QoS) or acceptance probabilities. Algorithms might even change during system's runtime to better adapt to environment variations. In such a new scenario, the basic assumptions made on the classical scheduling theory are no longer valid. New approaches are needed to handle these situations in a predictable fashion. They should enforce timing constraints with a certain degree of flexibility, aimed at achieving the desired trade-off between predictable performance and efficient use of resources.

Flexible scheduling is an underlying theme to most novel scheduling trends which go beyond the standard model of completely known tasks with timing constraints expressed as periods and deadline. Many applications areas, notably for control and media processing, have timing requirements which cannot be expressed only by deadlines and periods. As a consequence, scheduling algorithms should consider more flexible ways of expressing temporal constraints, to meet the demands of application level requirements rather than system models.

Issues addressed include probabilistic parameters, handling of applications with only partially known properties, relaxed constraints, coexistence of activities with diverse properties and demands in a system, combinations of scheduling schemes, and adaptive scheduling schemes.

While individual algorithms have been proposed to adapt scheduling parameters to changes in application demands, a systematic approach is needed to identify changes in the application, distinguish them between temporary and struc-tural variations, and adjust the scheduling parameters to de-termine a proper system response.

F. Hierarchical scheduling

Today's computers are powerful enough to execute mul-tiple applications at the same time. This may require parti-tioning the processor into several "virtual" machines, each with a proper fraction of computation power and scheduling algorithm. When different scheduling schemes are demanded in the same computer, the analysis of the entire system becomes complex and more theoretical work is needed for providing guarantee tests of multiple concurrent applications.

Hierarchical scheduling means that there is not just one scheduling algorithm for a given resource, but a hierarchy of schedulers. The tasks in the system are hierarchically grouped. The root of the hierarchy is the complete system; the leaves of the hierarchy are the individual tasks. At each node in the hierarchy, a scheduling algorithm schedules the children of that node. The practical value of a two level-hierarchy is immediately obvious: intermediate nodes are applications that are independently developed and/or independently loaded. If the root scheduler provides guaranteed resource reservation and temporal isolation, the applications can (with some additional precautions) be viewed to be running on private processors. There are two very distinct real-time processing domains where some form of hierarchical scheduling is proposed: one is the area of soft real-time in personal computers; the other is the area of certified hard real-time systems.

In the first domain, several frameworks have been proposed for deterministic soft real-time scheduler composition. In the second domain, ARINC proposes a root scheduler that provides time slots in a recalculated schedule. In other proposals, the root scheduler provides some form of guaranteed bandwidth allocation.

G. Overload handling

Predictability in dynamic systems is strictly related to the capability of controlling the incoming workload to prevent overload conditions. In fact, when the computation exceeds the processor capabilities, breakdown phenomena may cause abrupt performance degradation. Computational workload can be controlled using different techniques, each requiring deeper investigation.

1) *Selection of different QoS levels*. Some computations can be performed using different algorithms with different execution requirements, hence leading to results with different quality. In other cases, the precision (hence, the quality) of a result can be enhanced by increasing the number of steps of an iterative solution. Hence, the workload can be controlled by selecting the proper quality level for each system activity.

2) *Adjustable timing constraints*. In a real-time system, the workload depends not only on the amount of com-putation arriving per each unit of time, but also on the timing constraints associated with the computations. Hence, another way to react to overloads is to relax the timing constraints of the application tasks in the presence of high computational requirements. For periodic tasks, the load can be reduced by enlarging the periods [13], but more work is needed to deal with generic task sets with arbitrary deadlines.

3) *Admission control*. A third way to control the load is to filter the incoming requests of computation. This solution is the most drastic one, because it solves the overload by rejecting one or more tasks. However, additional research is needed to evaluate the effect of a rejection on the overall system performance.

H. Energy aware-scheduling

In battery-powered devices, reducing energy consumption is crucial for increasing system lifetime. Modern processors can operate at variable voltage/frequency levels for trading performance vs. energy consumption. In real-time systems, however, decreasing voltage prolongs task execution and may cause deadline misses; hence, future scheduling algorithms must take voltage into account to meet timing constraints while minimizing energy consumption.

The majority of scheduling algorithms with energy con-siderations mainly concentrates on the CPU, integrating dy-namic voltage scaling into the scheduling problem. However, the CPU is only one resource consuming power. Other components, like memory, disk, communication devices, and input/output peripherals, are not yet considered by the theory. Hence, new approaches for a system wide energy view are needed. In particular, the "energy overhead" of the scheduling algorithm with respect to CPU and memory should also be considered.

Applying *dynamic voltage scaling* techniques to control energy consumption causes also other problems in real-time systems. In fact, in the presence of timing and resource constraints, the performance of a real-time system does not always improve as the speed of the processor is increased. Similarly, when reducing the processor speed, the quality of the delivered service may not always degrade as expected.

1) Dynamic Voltage Scaling (DVS): Saves energy by dynamically scaling down processor voltage Execution time increases thenenergy should decrease Execution time decreaseenergy should increase

2) Advantage: Efficient, ease to use, and practical

3) Disadvantage: May adversely impact the feasibility of task schedules in real-time systems To prevent these problems, new approaches need to be investigated to allow the development of real-time ap-plications whose performance can be scaled in a controlled fashion as a function of the processor speed.

I. Portability

With the constant evolution of hardware, portability is also a very important issue, necessary to run applications developed for a particular platform into new hardware plat-forms. The use of standard programming interfaces opens the door to the possibility of having several operating system providers for a single application, and promotes competition among vendors, thus increasing quality and value.

Current operating system standards mostly specify portability at the source code level, requiring the application developer to recompile the application for every different platform.

J. Modeling

Modeling plays a central role in systems engineering. The existence of modeling techniques is a basis for rigorous design and should drastically simplify validation. In current industrial practice, models are essentially used only at the early phases of system design and at a high level of abstraction. Requirements and design constraints are spread out and they do not easily carry through the entire develop-ment lifecycle. Validation of large real-time applications is mainly done by experimentation and measurement on specific platforms, in order to adjust design parameters and, hopefully, achieve conformity with requirements. Thus, experimenting with different architectures and execution platforms becomes error-prone.

The use of models can profitably replaces experimentation on actual systems with incomparable advantages, such as:

1) Enhanced modifiability of the system parameters;

2) Ease of construction by integration of models of het-erogeneous components;

3) Generality by using abstraction and behavioural non-determinism;

4) Predictability analysis by application of formal methods.

Modeling methodologies should be closely related to implementation methodologies for building correct real-time systems as a sequence of steps involving both the de-velopment of software components and their integration in an execution and communication platform. These methodologies should support end-to-end constraints at every step in the design process and provide means to automatically propagate them down to the implementation. To be useful in practice, they should be accompanied by the development of appropriate middleware, QoS management support, and validation tools.

Modeling systems in the large is an important research topic in both academia and industry. Several trends can be identified in this area: One line of research consists of the socalled model-based approaches. This research groups the study of unified frameworks for integrating different models of computation, languages and abstraction-based design methodologies.

A key issue in a modeling methodology is the use of adequate operators to compose heterogeneous schedulers (e.g., synchronous, asynchronous, event-triggered, or time-triggered). For this reason, some researchers propose model-based theories for composing scheduling policies.

Another challenge consists in adequately relating the functional and non-functional requirements of the application software with the underlying execution platform. There are two current approaches to this problem:

One relies on architecture description languages that provide means to relate software and hardware components.

The other is based on the formal verification of automata-based models automatically generated from software and appropriately annotated with timing constraint Nevertheless, building models that faithfully represent real-time systems is not a trivial problem and still requires a great amount of theoretical and experimental research.

K. Towards component-based RTOS

Another promising research area is to adopt a component-based design paradigm at the kernel level. The use of component-based operating systems would allow the designer to quickly configure the kernel for a specific appli-cation just by combining existing modules, thus speeding up the development process and optimizing efficiency for the required functionality.

Today, software modularity allows the same system to be assembled in several incremental configurations, with dif-ferent features and functionality. For example, the POSIX standard, specifies four real-time profiles:

1) Minimal Real-Time System profile (PSE51). This profile is intended for small embedded systems and can be implemented with a few thousand lines of code and with memory footprints in the tens of kilobytes range. Processes are not supported and there is not a complete file system (input/output is possible through predefined device files).

2)Real-Time Controller profile (PSE52). It is similar to the PSE51 profile, with the addition of a file system in which regular files can be created, read, or written. It is intended for systems like a robot controller, which may need support for a simplified file system.

*3)Dedicated Real-Time System profile (PSE53).*It is intended for large embedded systems (e.g., avionics) and extends the PSE52 profile with the support for multiple processes that operate with protection boundaries.

4)Multi -purpose Real-Time System profile (PSE54). It is intended for general-purpose computing systems running applications with real-time and non-real-time requirements. It requires all of the POSIX functionality for general purpose systems and, in addition, most of the real-time services.

However, this level of modularity does not allow the user to replace an internal kernel mechanism with another one
with the same interface, nor replacing a mechanism without changing the application. This happens because all kernel mechanisms have strong inter-dependencies and are usually developed on the basis of other internal features. For example, typical implementations of the Priority Inheritance Protocol strongly rely on a fixed priority scheduler and cannot be used under deadline-based scheduling algorithms.

A true component-based approach should separate mechanisms from policies in order to replace a scheduling algorithm or a resource management protocol without af-fecting the applications and the others components. In addi-tion, it should allow a safe combination of different schedul-ing disciplines to support the development of hierarchical software architectures.

5) Advantages: There would be several benefits in adopting a component-based approach at the operating system level.

First of all, it would be possible to enhance the function-ality of the kernel by adding new blocks, depending of the application requirements, so tailoring the kernel to the spe-cific system to be developed.

Secondly, it would facilitate and speed up the integration of novel research results, which could increase efficiency and/or predicability.

Finally, it would simplify the process of porting the kernel on different platforms, so reducing the time to market and the development costs on upgrades (since only small parts should be developed).

6)Limitations: However, there are several practical and theoretical problems to be solved, since most of the mechanisms implemented in a kernel (like scheduling, resource protocols, interrupt handling, aperiodic servers, synchronization and communication) heavily interact with each other and have a high degree of inter-dependencies.

Specific research in this area should provide methods for decoupling scheduling algorithms from applications, sep-arating scheduling mechanisms from scheduling policies, separating scheduling algorithms from resource management protocols, and,

Finally, guaranteeing a safe integration of resource reservation with resource management protocols.

A promising research area consists in developing hybrid methods, which integrate two complementary types of adaptation strategies: one embedded in the application and the other performed by a QoS manager. Such an integration can be done by controlling the CPU bandwidth reserved to a task, but allowing each task to change its QoS requirements if the amount of reserved resources is not sufficient to accomplish the goal within a desired deadline. Using such an integrated approach, the QoS adaptation is performed in a task-specific fashion: each task can react to overloads in a different way and use different techniques to scale down its resource requirements. On the other hand, if a task does not react adequately to a lack of resources, the scheduler will slow it down in order not to influence the other tasks

VI. CONCLUSIONS

The possibilities for embedded systems to evolve and be-come more reliable, while yet more complex, to some extent depend on what the next generation real-time operating systems and implementation tools have to offer. The challenge is how to implement applications that can execute efficiently on limited resources, to meet non-functional requirements, such as timeliness, robustness, dependability, and performance.

Proper resource and quality-of-service management would enable the implementation of embedded systems that are more flexible, yet more deterministic, than it is possible today. Since such systems would be better specified, their properties would also be verified more easily. By supporting explicit resource allocation and quality-of-service func-tionality, the system designers would regain control over the system they are set to design.

To effectively assign system resources among applica-tions and achieve predictability and flexibility, a number of issues should be further investigated. At the higher abstrac-tion level, protocols for managing quality levels and suitable architectures should be used to obtain flexible systems. At a lower level, further work on resource management algorithms, new task models, admission control, monitoring, and adaptation algorithms should be done.

Also, a promising research area consists in developing hybrid methods, which integrate two complementary types of adaptation strategies: one embedded in the application and the other performed by a QoS manager. Such an integration can be done by controlling the CPU bandwidth reserved to a task, but allowing each task to change its QoS requirements if the amount of reserved resources is not sufficient to accomplish the goal within a desired deadline. Using such an integrated approach, the QoS adaptation is performed in a task-specific fashion: each task can react to overloads in a different way and use different techniques to scale down its resource requirements. On the other hand, if a task does not react adequately to a lack of resources, the scheduler will slow it down in order not to influence the other tasks.

As a conclusion, the following recommendations for re-search emerge in the area of real-time operating systems for embedded applications: Flexible scheduling services. The complexity of modern applications requires more flexibility in scheduling than just fixed priorities. Defining APIs that could make the scheduler a pluggable and interchangeable object seems the most promising research direction. Protection. One way of managing the complexity of applications is by providing appropriate levels of protection, both in space (memory) and time. The time protection mechanisms specified in some standards, like OSEK or ARINC, are somehow too rigid, and there are needs to make this protection more flexible but still effective. Dynamicity. The complexity of applications requires moving from statically designed applications to a more dynamic environment where the application components can be changed on-line. Research is needed on methods and new APIs are needed for effective on-line admission tests and dynamic resource reservation. Quality of service. There is a need for middleware that allows the application to define quality of service re-quirements, using some contract mechanism that lets the application specify its minimum and desired requirements, so if the implementation accepts the contract it can guarantee the minimum requirements and try to provide the desired ones. To implement this middleware, there is a need to develop techniques and APIs at the operating system level to perform load adaptation, monitoring and budgeting of the system re-sources. Multiprocessor support. Predictability of the timing behaviour in multiprocessor systems is still a research issue. Most multiprocessor real-time systems today require static allocation of threads to processors.

Drivers. Portability of drivers for real-time applications is an open issue. There is a need for extending current APIs for portable drivers to support real-time requirements. Networks. There are few real-time networks and protocols, and the support for them in operating systems is very limited. There is a need to develop protocol-independent APIs that let a distributed application define its timing requirements for the network and the remote services. Modeling. There is a need to develop precise models of the timing behaviour of the operating system services, which could be used in timing analysis tools. It would be useful to have automatic procedures to obtain the timing model of any operating system on a given platform.

REFERENCES

[1] C. L. Liu And James W. Layland “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment” Project MAC, Massachusetts Institute of Technology Jet Propulsion Laboratory, California Institute of Technology IEEE Vol 20 No.1 1973
[2] GC Buttazzo, G Lipari, M Caccamo “Elastic Scheduling for Flexible Workload Management” - IEEE 2002 Vol 54 computer.org
[3] C.Steiger, H.Walder and M.Platzner, “Operating Systems for Reconfigurable Embedded Platforms:Online Scheduling of Real-Time Tasks” IEEE Nov. 2004 Vol 53 Issue: 11
[4] Krzysztof M. Sacha “Measuring the Real-Time Operating System Performance” 1068-3070/95 1995 IEEE
[5] AbouGhazaleh, N. Mosse, D. Childers, B. Melhem, R. Craven, M. “Collaborative Operating System and compiler power management for real time applications” Dept. of Comput. Sci., Pittsburgh Univ., PA, USA IEEE 2003 Vol 56 10.1109/RTTAS.2003.1203045
[6] C. Centioli, F. Iannone, G. Mazza, M. Panella, L. Pangione, V. Vitale, and L. Zaccarian “Open Source Real-Time Operating Systems for Plasma Control at FTU” IEEE VOL. 51, NO. 3, JUNE 2004
[7] Scordino, C.; Lipari, G “A Resource Reservation Algorithm for Power Aware Scheduling of Periodic and Aperiodic Real-Time Tasks” Volume: 55 , Issue: 12 Digital Object Identifier: 10.1109/TC.2006.190 Publication Year: 2006
[8] The Performance and Energy Consumption of Embedded real time operating systems
[9] Ketan Kotecha and Apurva Shah “Adaptive Scheduling Algorithm for Real-Time Operating System” 978-1-4244-1823-7/08/ 2008 IEEE

[10] Fabricio Rusu-Banu and Yingxu Wang "Formal Description Of Time Management In Real-Time Operating Systems" 1-4244-0038-4 2006 IEEE CCECE/CCGEI, Ottawa, May 2006

[11] Caccamo, M.; Buttazzo, G.C.; Thomas, D.C. "Efficient Reclaiming in Reservation-Based Real-Time Systemswith Variable Execution Times" IEEE

Volume: 54 , Issue: 2

Digital Object Identifier: 10.1109/TC.2005.25

Publication Year: 2005 , Page(s): 198 - 213

.