INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

SCHEDULING ALGORITHMS OF REAL TIME SYSTEM AN OVERVIEW

Er. Jaitsri Kaur[1], Er.Sukhvinder Singh[2], Er. Mandeep Singh

BBSBEC, Fatehgarhsahib[1,], RIEBT[2,3]

jaitsri24@gmail.com[1], sukhaish@gmail.com[2]

Abstract:- The problem of real-time scheduling spans a broad spectrum of algorithms from simple uniprocessor to highly sophisticated multiprocessor scheduling algorithms. Real-time software must satisfy not only functional correctness requirements but also timeliness requirements. A lot of real-time researches were focused on analysis rather than testing recently, observes that real-time testing is still a "lost-world" compared to "civilization" developed in other areas of software, reflecting the little work done in the area. In this paper, we have studied the various scheduling algorithm of the real time system.

Keywords:- Real time System, Uniprocessor, Multiprocessor etc.

.

1. Real Time System

A system is defined as being real-time if it is required to respond to input stimuli within a finite and specified time interval. The stimuli being either an event at the interface to the system or some internal clock tick that is, at least notionally, coordinated with the passage of time in the system's environment. Real-time systems are found in a wide range of applications areas, from simple domestic appliances to multi-media systems, large scale process control and safety critical avionics. In some systems the required response times are measured in milliseconds, in others it is seconds or even minutes. Nevertheless they all have dead- lines that must be satisfied. Real-time systems are systems where timeliness is essential to correctness. In a real-time system the correctness of the system depends not only the logical results of the computations, but also on the physical instant at which these results are produced.Consider the airbag in a car. It does not suffice to establish that after a collision it will expand it is also crucial that this happens neither too early nor too late. Such systems play a major role in industrial design and development. In particular embedded systems, whose primary purpose is to offer some service other than computation itself, outnumber full-grown processors by an order of magnitude,: most computations today are done on small, weak, and cheap hardware. These systems are found in everyday use and range from simple appliances like wristwatches and remote controls to complex designs, such as mobile phones, cars, and airplanes.

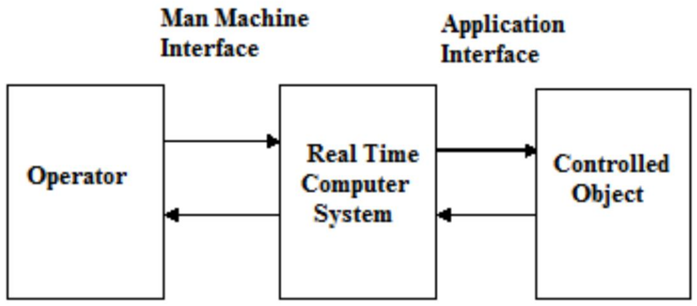


Fig1: Example of Real Time System

A generic real-time system can be described as consisting of three principal subsystems, as shown above. The *controlled object* represents the application, or environment (e.g. an industrial plant, a computer controlled vehicle), which dictates the real-time requirements. The *real time computer system* controls some computing and communication equipment for use from the controlled object. The *operator* initiates and monitors the entire system activity. The interface (application interface) between the controlled object and the real time computer system consists of such devices as sensors and actuators. The interface between the control subsystem and the operator consists of a man-machine interface.

Real-time systems originated with the need to solve two main types of applications: event response, and closed loop control systems. Event response applications require a response to a stimulus in a determined amount of time, an example of such a system is an automotive airbag system. Closed loop control systems continuously process feedback in order to adjust an output; an automotive cruise control system is an example of a closed-loop control system. Both of these types of systems require the completion of an operation within a specific deadline. Definition of "real time system" is *"Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified delay"*. Now, we can say that a **real-time system** is a system whose specification includes both logical and temporal correctness requirements.

- **Logical Correctness:** Produces correct outputs.

- **Temporal Correctness:** Produces output at the right time.

It is mandatory to define embedded system with real time system. To most people, embedded systems are not recognizable as computers. Instead, they are hidden inside everyday objects that surround us and help us in our lives. Embedded systems typically do not interface with the outside world through familiar personal computer interface devices such as a mouse, keyboard and graphic user interface. Instead, they interface with the outside world through unusual interfaces such as sensors, actuators and specialized communication links. Real-time and embedded systems operate in constrained environments in which computer memory and processing power are limited. They often need to provide their services within strict time deadlines to their users and to the surrounding world. It is these memory, speed and timing constraints that dictate the use of real-time operating systems in embedded software. Here are some definitions of embedded system. *"An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, usually with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts."* For example, many embedded systems are referred to as real-time systems. Cruise control, telecommunications, flight control and electronic engines are some of the popular real-time system applications where as computer simulation, user interface and Internet video are categorized as non-real time applications.

2. Characteristics of Real Time Systems

Real-time software systems have several characteristics that distinguish them from other software systems: **Timeliness:** The results of a computer operation, i.e. control signals or responses, must be evaluated according to certain rules and within fixed periods of time. Due to physical characteristics of the environment late results can be useless or even dangerous. The conclusion is that the correctness of real-time systems depends not only on the processing results, but also on the time when these results become available. Correct timing is determined by the needs of the environment and not by the computer's processing speed. **Simultaneity:** The environment of a real-time system is usually composed of a number of partially independent objects, e.g. sensors and actuators. The objects operate concurrently, producing data and requesting services from the computer system. To respond appropriately, the computer system should possess the ability for simultaneous processing of concurrent events. **Interaction with External Environment:** A real-time system typically interacts with an external environment, which is, to a large extent, non - human. For example, the real-time system may be controlling machines or a manufacturing processes or it may be monitoring chemical processes and reporting alarm conditions. This situation often necessitates a sensory interface for receiving data from the external environment and actuators for outputting data to and controlling the external environment. **Continuous operation.** The definition of real time system does not refer to any “start" or “stop" points for the system activity. On the contrary, the system should at least in principle run forever, synchronously with the events which occur within the external process.

Predictability and analysability. Data to be processed may arrive randomly distributed in time. The simultaneous occurrence of several events can lead to a competition for service. Such an indeterminism arises particularly in the case of transient overload and other error situations. Despite this unpredictability, the reactions to be carried out by the computer must be precisely planned and fully predictable. This raises the requirement for the analyzability of a real-time system description. **Real-Time Constraints:** Real-time systems have timing constraints, *i.e.*, they must process events within a given time frame. These real-time constraints are specified in the soft- ware requirements. Whereas, in an interactive system, a human may be inconvenienced if the system response is delayed, in a real-time system, a delay may be catastrophic. For example, inadequate response in an air traffic control system could result in a midair collision of two aircraft. The required response time will vary by application, ranging from milliseconds in some cases, to seconds, or even minutes, in others. **Embedded Systems:** A real-time system is often an embedded system, *i.e.*, the real-time software system is a component of a larger hardware/software system. They often contain devices that act as the senses (e.g., heat sensors, or light sensors), and devices that act as (actuater) the effect of physical changes (e.g., mechanical, electromechanical, and electronic actuators). An example of this is a robot controller that is a component of a robot system consisting of one or more mechanical arms, servo-mechanisms controlling axis motion, processing, *i.e.*, and sensors and actuators for interfacing to the external environment. A computerized automobile cruise control system is embedded in the automobile. **Reliability and Safety:** Real time systems have higher reliability and safety requirements than that required by other systems. The failure of a system involved in automatic fund transfer between banks can lead to millions of dollars being lost; failure in an embedded system could result in the failure of a vital life-support system. **Reactive Systems:** Many real-time systems are reactive systems. They are event-driven and must respond to external stimuli. It is usually the case in reactive systems that the response made by the system to an input stimulus is state dependent, *i.e.*, the response depends not only on the stimulus itself, but also on, what has previously happened in the system.
Fault detection: Fault processing in a real-time context demands special techniques to detect them, recover them and evaluate their impact on the global behavior of the application.

3. Classification of Scheduling Algorithms

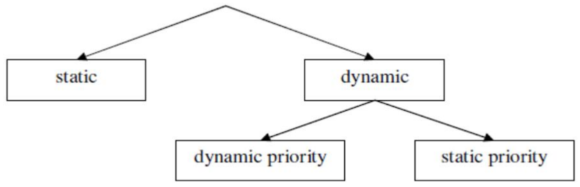
A schedule for a given task set is called feasible if the execution of all tasks meets given constraints, such as deadlines, precedence relations, and mutual exclusion. A set of tasks is schedulable or feasible if there exists at least one algorithm that can produce a feasible schedule. An optimal scheduling algorithm is the best one according to some criteria. For example, a performance measure can be defined over the task set and the scheduling algorithm is optimal if it minimizes this performance measure. If the only concern is to obtain a feasible schedule, then an algorithm is optimal if it will produce a feasible schedule for all feasible task sets. If an optimal algorithm fails to schedule a task set, then any algorithm from the same class will also fail. Common performance measures for evaluating schedulers include the average response time, maximum lateness, or the number of tardy tasks. The lateness of a task instance Li,j is defined as the time difference between the finishing time and the deadline, measuring how late a task execution is: Li,j = fi,j − di. If the task instance completes before the deadline, its lateness is negative. In general, finding optimal algorithms can be computationally intractable.

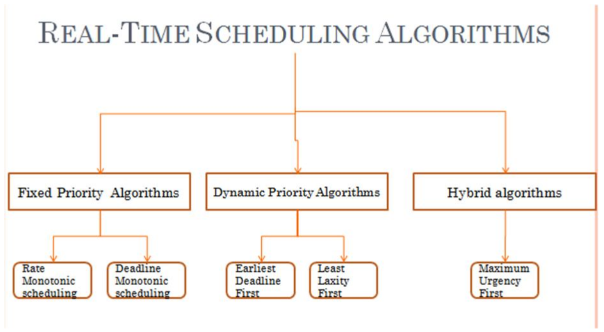
The reason is that scheduling belongs to the class of combinatorial search problems. The number of possible combinations can grow exponentially with the number of tasks. Optimal polynomial algorithms exist for particular scheduling problems, where tasks have fewer constraints.

approximation or heuristic algorithms. These do not guarantee a solution but provide a feasible schedule in most cases.

Real-time scheduling algorithms have been an active topic of research since the late 1960's. Real-time scheduling algorithms work either dynamically or statically. Static scheduling algorithms and techniques have more computer time to devote to scheduling because scheduling is completed before the target system begins. As you might expect, unless all information about all tasks is known before the system is brought online, the system may not use a static scheduling algorithm. The dynamic scheduling algorithms are further divided into static priority and dynamic priority. As you might expect, unless all information about all tasks stays unchanged throughout each of the tasks. lifetimes, the system may not use a static priority algorithm. This classification is not exhaustive in depth or detail.

Real Time Scheduling Algorithms




REAL-TIME SCHEDULING ALGORITHMS
Fixed Priority Algorithms
Dynamic Priority Algorithms
Hybrid algorithms
Rate Monotonic scheduling
Deadline Monotonic scheduling
Earliest Deadline First
Least Laxity First
Maximum Urgency First

A scheduling algorithm is non-idling if the processor is never inactive when there are tasks ready to execute. Scheduling algorithms may be further classified as follows: • **Off-line vs. On-line**: In off-line scheduling, all decisions are computed at compile time and stored in a dispatch table; at run-time no scheduler is needed, but only a dispatcher which takes the next entry from the table. In on-line scheduling, all decisions are taken at run-time, when a new task is released or when a task terminates its execution. Off-line scheduling needs a priori knowledge of all task attributes, including all release times; whereas in on-line scheduling, the attributes of a task become available only when the task is released. Off-line scheduling is less flexible but it has small run-time overhead, being actually a simple table lookup. On-line scheduling is also called process-based scheduling.

• **Preemptive vs. non-preemptive**: A preemptive scheduler can interrupt a task execution, when some higher-priority task needs to be executed; and resume it later, when the higher-priority task terminates. The executions of tasks are interleaved. In contrast, a non-preemptive scheduler will execute a task until completion, regardless of which requests became enabled in the meantime. Some tasks require explicitly non-preemptive scheduling because of their functionality, for example an interrupt handler that saves the state of the processor. If the application has no restrictions on preemption, it is not trivial to answer which alternative is better. Typically, the maximum response time from all tasks is smaller for an optimal preemptive algorithm than for an optimal non-preemptive one, but the cost of preemption is most of the times ignored in the analysis. The question remains if the benefit of preemption compensates for the context-switch overhead. • **Static vs. dynamic:** In static scheduling, all scheduling decisions are based on fixed parameters, assigned to tasks before their activation; whereas in dynamic scheduling, all decisions are based on dynamic parameters that might change at run-time (Buttazzo, 1997). Static scheduling needs a priori knowledge of all task attributes; therefore it is less flexible. Dynamic scheduling can provide a better processor utilization and supports non-predicted events, but it has a higher runtime overhead than static scheduling. Static and dynamic scheduling has a further meaning in multiprocessor and distributed systems, depending on how these systems are configured. In a static system, the tasks are partitioned into subsystems and they are statically allocated to processors. The tasks on one processor are scheduled independently, except the cases when they must be synchronized. In a dynamic system, the tasks are dynamically dispatched to processors. There is a common queue for all processors and the task in the head of the queue is dispatched to the first processor that becomes idle. If preempted, a task can migrate from one processor to another in order to resume its execution. Both off-line and on-line scheduling can use a preemptive or a non-preemptive algorithm, but on-line schedulers avoid computationally expensive algorithms because the run-time overhead should be minimized. Off-line scheduling can use complex algorithms for optimizing the schedule; for example, scheduling can be implemented as a search tree, using branch-and-bound algorithms.
Off-line scheduling implies static scheduling, On-line scheduling may be either static or dynamic, in the sense that the scheduler can assign static priorities (i.e., fixed priorities) or dynamic priorities to tasks. Furthermore, there are applications that need to create or delete tasks dynamically, depending on the environmental requests. The time instants of these events cannot be predicted and included in an off-line schedulability analysis. Therefore, such dynamic systems need on-line scheduling algorithms and efficient schedulability tests that can be applied on-line. An air traffic control system is an example of a hard real-time application where the dynamic creation and deletion of tasks is necessary. Each aircraft is monitored by a task and the number of tasks changes as aircrafts enter or leave the area. Each scheduling paradigm has advantages and drawbacks, being suited for different applications. Safely-critical systems need determinism and timing guarantees; therefore, they give up flexibility and they are typically implemented with static scheduling. Off-line scheduling manages well distributed applications with complex constraints. Moreover, off-line scheduling does not need a Schedulabiltiy test, because building the dispatch table guarantees the feasibility of the application. In contrast, Schedulabiltiy analysis might not be trivial for the on-line algorithms.
Real-Time systems can be classified from different perspectives. The first two classifications, hard real-time versus soft real-time, and fail-safe versus fail-operational, depend on the characteristics of the application, i.e., on factors outside the computer system. Third class of real time system is event-triggered versus time-triggered, depend on the design and implementation, i.e., on factors inside the computer system. **Fail-Safe Versus Fail-Operational** Fail-Safe system goes to the known system state when the fault occurs. On the other hand (Fail operational) when the system faults occurs system can be degraded gracefully and operational. Fail safe systems -- when in doubt, turn off • Radiation therapy machines • Car engines • Power tools • Nuclear power plant Fail operational systems -- when in doubt, keep working • Aircraft engines • Military combat systems

Event-triggered Versus Time-triggered

Event triggered systems where actions have to be performed not at particular times or time intervals but in response to same event. Ex: Turning off pump or closing a valve when the level of a liquid in a tank reaches predetermined value.
Events occur at non-deterministic intervals and event-based tasks are aperiodic tasks. Such tasks have deadlines expressed in terms of having start-time or Finish times. **Event-triggered** • Computation/communication responds to an event • Events happen whenever they want to happen • Think "interrupt-driven I/O" • High peak load -- what if all possible events happen simultaneously?
Time-triggered • Computation/communication responds to a system clock time • Events happen according to a schedule (fixed or changeable) • Think "I/O polling" • easily characterized load -- a spread sheet can schedule the whole system.
A system is defined as being real-time if it is required to respond to input stimuli within a finite and specified time interval. The stimuli being either an event at the interface to the system or some internal clock tick that is, at least notionally, coordinated with the passage of time in the system's environment. Real-time systems are found in a wide range of applications areas, from simple domestic appliances to multi-media systems, large scale process control and safety critical avionics. In some systems the required response times are measured in milliseconds, in others it is seconds or even minutes. Nevertheless they all have deadlines that must be satisfied.
The concept of time is the main characteristic that differentiates real-time systems from other computing systems. Nevertheless, there is a wide range of real-time applications and timing constraints. Depending on the consequences of timing failures, we distinguish between hard real-time and soft real-time systems.

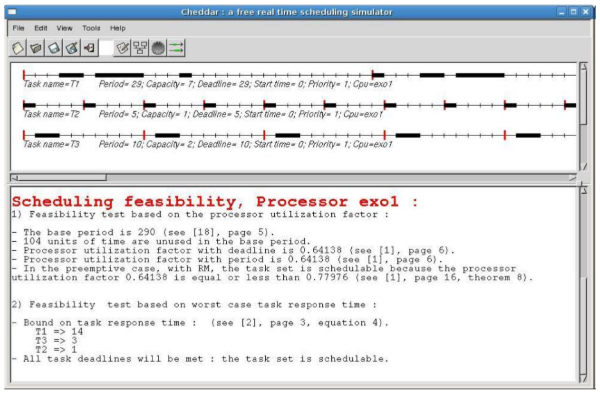
In hard real-time systems, timing constraints are of utmost importance and violating timing constraints may have catastrophic consequences on the controlled environment. Examples of hard real-time applications include industrial process control or the automotive and avionics industries. Hard real-time systems are further classified into safety-critical systems, where a failure leads to loss of human lives; and mission-critical systems, where a failure leads to major economic loss or mission failure.
In soft real-time systems, meeting timing constraints is desirable, but a few violations are tolerated as they only lead to performance degradation. Examples of soft real-time applications include multimedia systems, electronic games, and on-line transaction systems.
When activities have timing constraints, as is typical of real-time computing systems, scheduling these activities to meet their timing constraints is one major problem that comes to mind. However, in spite of an extensive literature on scheduling, scheduling algorithms that is of practical value for real-time computing, ones that take real-world considerations into account, have only begun to appear. Given the vast amount of work that has been done by both the operations research and computer science communities in the scheduling area, it is impossible to do an exhaustive survey of the field.

Clearly, a real-time operating system must be able to perform integrated CPU scheduling and resource allocation so that collections of cooperating tasks can obtain the resources they need, at the right time, in order to meet timing constraints. In addition to proper scheduling algorithms, predictability requires bounded operating system primitives. Using the current operating system paradigm of allowing arbitrary waits for resources or events, or treating a task as a random process will not be feasible in the future to meet the more complicated set of requirements. It is also important to avoid having to rewrite the operating system for each application area.

4. Cheddar: Real Time Scheduling Simulator

Cheddar is a free real time scheduling tool. Cheddar is designed for checking task temporal constraints of a real time application/system. Systems to analyze can be described with AADL or a with Cheddar specific language. It can help you for quick prototyping of real time schedulers. It can also be used for educational purpose. Cheddar is developed and maintained by the LISyC Team, University of Brest.
Cheddar relies on Ocarina to provide schedulability analysis of AADL models. The ocarina team also use the Cheddar analyzer for their research projects and some tutorial/labs. The two teams cooperate for the development of AADL

software engineering tools. Cheddar provides two kinds of features: a simulation engine and feasibility tests. Feasibility tests allow a user to study a real time application/system without computing a scheduling. In the contrary, the simulation engine can be used firstly to computing a scheduling and secondly, to automatically looking for task constraint properties in the computed scheduling. Most of the time, feasibility tests are less complex tools but they are available only for few schedulers and tasks models. To solve this problem, the Cheddar simulation engine provides tools to design specific schedulers and task models: the system is then analyzed according to a given scheduling.


Cheddar : a free real time scheduling simulator
File Edit View Tools Help

```
Task name=T1    Period= 29; Capacity= 7; Deadline= 29; Start time= 0; Priority= 1; Cpu=exo1
Task name=T2    Period= 5; Capacity= 1; Deadline= 5; Start time= 0; Priority= 1; Cpu=exo1
Task name=T3    Period= 10; Capacity= 2; Deadline= 10; Start time= 0; Priority= 1; Cpu=exo1

Scheduling feasibility, Processor exo1 :
1) Feasibility test based on the processor utilization factor :

- The base period is 290 (see [18], page 5).
- 104 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.64138 (see [1], page 6).
- Processor utilization factor with period is 0.64138 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor
utilization factor 0.64138 is equal or less than 0.77976 (see [1], page 16, theorem 8).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    T1 => 14
    T3 => 3
    T2 => 1
- All task deadlines will be met : the task set is schedulable.
```

5. RATE MONOTIC SCHEDULING ALGORITHM

The RM scheduling algorithm is one of the most widely studied and used in practice. It is a uniprocessor static-priority preemptive scheme. Liu and Layland were perhaps the first to formally study priority-driven algorithms. They focused on the problem of scheduling periodic tasks on a single processor and proposed two preemptive algorithms. The first algorithm, called the Rate-Monotonic (RM) algorithm and Earliest deadline first (EDF). The term rate monotonic derives from a method of assigning priorities to a set of processes as a monotonic function of their rates. In this algorithm, priorities are assigned according to the request rate (frequency) of tasks. RMA assigns static priorities to tasks based on their periods. The priority of a process is determined by its period time. The highest priority is assigned to the process with the shortest period time. The remaining processes get priorities according to the ascending order of their period times. A system is schedulable if all tasks meet their deadlines. Rate monotonic analysis provides a mathematical and scientific model for reasoning about schedulability. One drawback of the RM algorithm is the schedulable bound is less than 100%. Schedulable bound is the maximum value of the CPU utilization for a set of tasks up to which all tasks will be guaranteed to meet their deadline. Since *Ci/Ti* is the fraction of processor time spent in executing task ti, the total CPU utilization for *n* tasks is:

$$U = \sum_{i=1}^{n} (C_i / T_i)$$

where *Ci* is the worst-case computing time and the period is *Ti*. The worst-case schedulable bound *Wn* for *n* tasks is: *Wn* = *n* (21/n – 1) This worst case utilization bound decreases monotonically from 0.83 when *n* = 2 to 0.693 as *n* approaches infinity. This shows that any periodic task set of any size will meet deadlines all the time if the rate monotonic algorithm is used and the total utilization is not greater than 0.693. It has been observed that the RM algorithm can successfully schedule periodic task sets with total utilization around 0.90. This suggests that the average case behavior is substantially better than the worst-case behavior.

Basic Assumptions When applying the RMA we'll have to make several assumptions about the system. We assume that tasks are selected for running by priorities. If a task with higher priority comes available, then the current task is interrupted and the task with higher priority will be run. The following assumptions are made for RM algorithm.

- In RMA task can be preempted any at time and resume it later.
- In RMA all tasks are independent, no dependency.
- Task interactions are not allowed.
- Tasks become ready to execute precisely at the beginning of their periods and relinquish the CPU only when execution is complete.
- In RMA all tasks are periodic.
- Tasks with shorter periods are assigned higher priorities; the criticality of tasks is not considered.
- Task execution is always consistent with its rate monotonic priority: a lower priority task never executes when a higher priority task is ready to execute.
- In RMA deadline of a task is equal to its period
- If processor utilization(U) in RMA is less than 0.693 ($U < 0.693$), schedulability is guaranteed.
- In RMA (some cases) tasks may be schedulable even if $U > 0.693$.

6. Scheduling Approaches

The two main approaches for scheduling real-time systems fall into two categories: the use of heuristic scheduling algorithms for real-time systems in which the timing constraints of the system can change frequently or are unpredictable and the use of rigorous schedulability analysis coupled with predictable scheduling algorithms to guarantee performance for real-time systems in which the timing constraints do not change dynamically.

a) Heuristic Scheduling

Real-time scheduling algorithms for very dynamic systems, in which little can be said about task arrival patterns or expected computation load, typically use heuristics to schedule tasks. Obtaining optimal solutions for these systems is generally an NP-hard problem, so heuristics are used to create a schedule at runtime that attempts to maximize a particular scheduling metric. Examples of proposed heuristic approaches show that online scheduling can provide acceptable scheduling performance for dynamic real-time systems.

b) Predictable Scheduling

An alternative to using heuristics is to employ algorithms that attempt to capture the timing behaviour of all tasks in the system. The goal of these algorithms is to provide a high degree of schedulable resource utilization and an *a priori* verification of timing correctness for all tasks in the system. These algorithms are more restrictive than the heuristic methods in that they require more knowledge about task utilization, arrival patterns, and resource requirements. The well-developed algorithms in this class are for the priority-driven, pre-emptive scheduling of periodic tasks with hard deadlines.

Vol.2 Issue.1, Pg.:185-193
January 2014

REFERENCES

[1] F.SinghOff, J. Lengrand, L. Nana , "Cheddar : a Flexible real time scheduling framework", ACM ,SIGADA-2004.

[2] Marco Spuri, " Analysis of deadline scheduled Real Time Systems" Jan-96.

[3]. Frank SinghOff, Alain Plantec, "Can we increase usability of Real Time Scheduling Theory? The Cheddar Project"

[4]. F.Singhoff, J. Legrand,L.Nana ,"Extending Rate Monotonic Analysis when tasks share buffers". T.P. Baker. "Multiprocessor EDF and deadline monotonic schedulability analysis". In proceedings. Real-Time Systems Symposium (RTSS), pp. 120–129, 2003.

[6] S.K. Baruah, A. Burns, "Sustainable Scheduling Analysis". In proceedings Real-Time Systems Symposium, pp. 159-168, 2006.

[7] S.K. Baruah. "The limited-preemption uniprocessor scheduling of sporadic task systems". In Proceedings Euromicro Conference on Real-Time Systems, pp. 137–144, 2005.

[8] S.K. Baruah, "Techniques for Multiprocessor Global Schedulability Analysis". In proceedings Real-Time Systems Symposium, pp. 119-128, 2007.

[9] .S.K. Baruah, N. Fisher. "Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems" In proceedings International Conference on Distributed Computing and Networking, pp. 215-226, Jan 2008.

[10] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability" In Proceedings of OSPERT, , pp. 33-44, Brussels, Belgum, 2010.

[11] M. Bertogna, M. Cirinei, G. Lipari, "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors". In proceedings International Conf. on Principles of Distributed Systems, pp. 306-321, Dec. 2005.