Resource and Precedence Constraints in a Hard Real-time System

Luis Gutiérrez, Raúl Jacinto, Julio Martínez, José Barrios

Universidad Autónoma de Aguascalientes, Av. Universidad s/n,
Aguascalientes, Ags., 20100 México
lgutierr@cencar.udg.mx, rjacinto@cucea.udg.mx, jucemaro@yahoo.com,
jmbarrio@hotmail.com

Abstract. This work addresses the problem of resource allocation and precedence constraints in a Hard Real-time system with Earliest Deadline First policy (EDF): the concurrency control and the precedence constraints in periodic tasks. It is an improved variant of Stack Resource Policy (SRP) [1], which strictly bounds priority inversion using dynamic scheduling policies. The task deadlines are reduced to manage precedence constraints; a modified SRP-based resource access control approach is proposed. This paper uses resource constraints as in the SRP policy along EDF; the proposed algorithm produces a well-formed policy to handle both precedence-constrained periodic tasks and shared resource access in a Hard Real-Time System.

Keywords: deadline, precedence-constraints, real-time, resource-constraints.

1 Introduction

Real-time system activities are called process, tasks or jobs. A real-time system is characterized by a timing constraint in its tasks called deadline, which represents the time before a process should complete its execution without causing a catastrophic result to the system. A system is considered to be a hard real-time system, if a failure caused by a missed deadline may lead to disastrous effects.

Handling tasks with precedence relationships and resources contention is an issue which has not yet fully explored. In this paper this issue is addressed taking basis on the proposed solution in the SRP algorithm [1] where task priorities define the order of resource access using a stack mechanism. In designing a Real Time System restrictions (other than those inherent to the operating environment) need to be considered. Three main inherent restrictions of a real-time task can be usually found [2]: Time restrictions, determined by its deadline and its execution frequency or period, precedence restrictions and resource access restriction, meaning resource access management to guarantee the fair use of available resources.

The unpredictability caused by shared resource access in conflict with other concurrent tasks needs to be controlled and it is the main subject of this paper. Determining the feasibility of scheduling a set of time-restricted tasks with shared-resource access and with precedence constraints is a NP-hard problem [3]. In order to

Received 01/07/07
Accepted 19/10/07
Final version 24/10/07

reduce the complexity, this paper adopts a preemptive model with assumptions which are defined below.

The paper presents a variant of the Stack Resource Policy (SRP) proposed by Baker of a policy to include handling of precedence constraints. The restrictions mentioned above are solved by the conjunction of Earliest Deadline First policy [4] and SRP.

In section 2 the main concepts on Real-time Systems and concurrency control are presented. In section 3 the proposal's model and the proposed resource allocation policy are discussed. Section 4 is devoted to conclusions and to sketch future work.

2 Concurrency Control and the Stack Resource Policy

Concurrency control is the management of concurrent conflicting task operations that access shared resources or data [5] to avoid conflicts among them. Real-time concept adds time requirements satisfaction – deadlines – to this definition [6]. A conflict occurs when two tasks try to access the same resource at the same time and, at least, one of the tasks operations being a write operation.

Concurrency control problem can be summarized in two main functions: conflict detection and conflict resolution [7]. The standard conflict detection is accomplished by using locks (usually mutex semaphores) over resources. When a resource is unlocked, the resource is free; otherwise is busy. Usually the lock is considered to be exclusive, but in more advanced systems, the type of operation determines the type of lock – either read (non exclusive or weak) or writes (exclusive or strong) locks. Conflict resolution often is achieved by choosing one of the conflicting tasks for abortion, forcing it to release its locks.

In real-time systems, the concurrency problem is even harder than in conventional systems. Priority inversions, blocking, and deadlocks, are problems that should be considered to keep the schedulability in the processes involved. To deal with concurrency, a natural approach consists in conflict avoidance rather than the conventional detection and resolution approach.

If the problem is avoided, the time used in detection and resolution (time expended in maintaining wait-for graphs, task election for abortion and resource releases) can be dismissed allowing for a more efficient system execution. However none of possible solutions, conflict avoidance or conflict detection and resolution can avoid the Blocking problem. Specialized locks for reading and for writing can reduce this problem but at the end, there is a need for locks to avoid inconsistencies in the system due to concurrency problems.

Blocking is a source of unpredictability in real-time systems in several forms: Undefined waiting, priority inversion, deadlock, etc. When undefined waiting is potentially caused by priorities, this can be avoided by *priority inheritance*: when a higher priority task is blocked by a lower priority task, the lower priority task inherits priority from the higher one so it cannot be blocked by medium priority tasks.

Unhappily this technique can lead to *priority inversion*, however, when properly used, reduces blocking time and chained blocking generation [8]. Even if it is not clear whether to use conflict detection and resolution or conflict avoidance, the work in concurrency control in real time systems seems to be biased to avoidance mechanisms. Among this work, Stack Resource Policy (SRP) proposed by Baker in [1] offers improvements over others strategies. SRP can be applied directly to some dynamic scheduling policies like EDF, which can support stronger schedulability test.

SRP reduces the number of context switches in the execution jobs.

2.1 Stack Resource Policy

The Stack Resource Policy (SRP) is a technique proposed by Baker, It was proposed for accessing shared resources. SRP works with priority and preemption level in each task. The priority indicates the importance of a task with respect to another in the system, and it can be assigned either statically or dynamically.

In SRP each tasks τ_i was characterized by a preemption level π_i, which is a static parameter assigned to each task before running the system. With the preemption level, a task τ_a can preempt to another task τ_b only if $\pi_a > \pi_b$. A fixed parameter makes easier the prediction of potential blocking in spite of dynamic priority schemes like EDF.

In SRP, each resource is required to have a current ceiling C_R which is a dynamic value computed as a function of the units of R (the system resource set) that are currently available, n_R denotes the number of units of R that are currently available; $\mu_R(J)$ indicates the maximum requirement of job J for R. The current ceiling of R is defined by

$C_R(n_R) = \max[\{0\} \text{ U } \{\pi(J): n_R < \mu_R(J)\}]$

The system ceiling π_s is defined as the maximum of the current ceilings of all resources, it is $\pi_S = \max(C_{Ri}: i=1,\ldots,m)$. When a job needs a resource that is not available, it is blocked at the time it attempts to preempt, rather tan later. In SRP, a job is not allowed to begin until the resources currently available are sufficient to meet the maximum requirement of every job that could preempt it, so SRP prevents multiple priority inversions. A job could preempt only if its priority is the highest among all the tasks in the ready state, and its preemption level is higher than the system ceiling.

3. System Model and a Resource Allocation Policy

The system model consists of a periodic task set τ. For τ the restrictions considered are: time restrictions (execution time, deadline and period), precedence restrictions and resource restrictions (resource requests by each task).

3.1 Symbols

τ_i	A generic periodic task
r_i	The release time of an instance of a generic task

C_i	The computation time of a task (periodic or aperiodic)
d_i	Absolute deadline of a task
D_i	Relative deadline of a task
s_i	Start time of a task
Φ_i	The phase of a periodic task.
f_i	Finish time
T_i	Period of τ_I
U_p	Processor utilization factor
U_s	Processor available factor
τ	A periodic tasks set
G_i	A directed acyclic graph. It describes the precedence constraints between a task subset with causal relations.
R	A system resources set
R_i	A generic resource

3.2 Assumptions

- A single-processor system
- A set of periodic task τ.
- Each periodic task τ_i has a period T_i, a computation time C_i, and a relative deadline D_i.
- Each Deadline D_i may be different to the period $T_{i.}$
- Periodic tasks are scheduled using dynamic-priority assignment, namely Earliest Deadline First (EDF);
- Periodic tasks can start at any time and not only at time t=0.
- Tasks are preemptive (i.e. they may be suspended and inserted into the ready queue to service ready tasks with major priority).
- All periodic tasks have hard deadlines.
- The precedence relations are described by directed acyclic graphs.
- The whole solution is based on EDF, mainly because it allows a maximum utilization of the available computing resources but also because it allows a dynamic behavior in arriving tasks.

3.3 Problem Definition

The particular problem in this paper is to create a policy with EDF which assigns resources from R to the tasks from τ and keeps the precedence constraints in the tasks which are defined by directed acyclic graph. The goal is to create a feasible schedule that adheres to the policy.

3.4 Proposed Solution

The proposal considers two objectives: Solving the precedence constraints and after that, solving the resource contention problems in the tasks. The final solution ensures schedulability at same time it keeps resource and precedence restrictions. The proposal works in three steps: defining an execution order considering the precedence constraints, setting the execution order, and assigning preemption levels.

Defining an Execution Order considering the Precedence Constraints. For each graph (fig. 1) is necessary to generate a serialized schedule which determines the execution order of the task with precedence constraints of G_i. This schedule is obtained by applying the preorder algorithm on the tree graph. An example is showed in fig. 2.

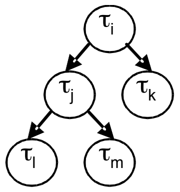

Fig. 1. Periodic tasks with precedence.

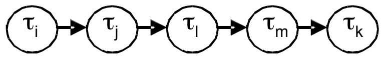

Fig. 2. An execution serialized schedule.

Setting the Execution Order. After the execution order has been established, the next step is to adjust the parameters for the tasks in order to guarantee precedence constraints. The parameters to be modified are the relative deadlines and the phase in periodic tasks; the phase Φ_i is the first activation time of τ_i. Clearly the root must be activated first. So, the tasks phases must be set in the growing order taking as reference the execution order of the first step of the algorithm. Considering the example in fig. 2, the phases must be $\Phi_i<\Phi_j<\Phi_l<\Phi_m<\Phi_k$, however it can lead to a priority inversion anomaly due to the fact that not necessarily the deadlines are also in a decreasing order; taking into account the example of the established order in fig. 1 and fig. 2, here is showed an example of the precedence anomaly in fig. 3 where the task τ_l starts the execution before the conclusion of τ_j.

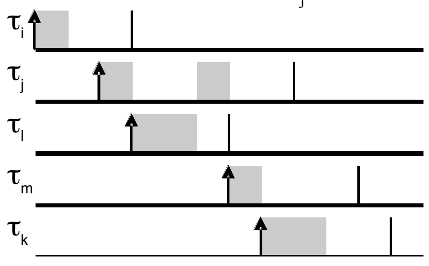

Fig. 3. A violation in the precedence constraints occurs with the tasks τ_2 and τ_3.

In order to cope with this problem, the relative deadlines need to be modified to create a schedule with increasing deadlines. To handle the modification, the "execution breadth" is defined as the maximum time that a task could wait to start its execution in order to meet the deadline.

The execution breadth in a periodic task τ_i is defined as the difference between the relative deadline and the computation time, it is expressed by: $H_{j,i}=D_i-C_i$, where H_j is the set of all execution breadth in the tasks that belongs to G_j. For all periodic tasks in G_i, in order to keep the precedence constraints assignment of the same execution breadth is needed. The new execution breadth is determined by: $H'_{j,i}=\min(H_j)$.

In this way, the new relative deadlines of the task in G_i are defined by $D_{j,i}= H'_{j,i}+C_i$. With this modification the relative deadlines produce a schedule with increasing deadlines which leads to the generation of a schedule which keeps the precedence constraints. This is proved by the next theorem.

Theorem 1. Let τ be a set with n periodic task: $\tau_1, \tau_2, \tau_3,.., ,\tau_n$. Assume for all τ_i : $C_i \leq D_i \leq T_{i.}$ Now, the precedence constraints are represented by a directed acyclic graph called G. H is the set of all execution breadth in the tasks that belongs to G_j, H>0. σ is a schedule of the entire task in τ with an order keeping the precedence relations.

If it is applied the modification in the parameters as above is defined then the real execution with EDF will be executed in the order established by σ.

Proof. It is proceed by Induction and is based on demonstrating that in all time the deadlines will be increasing congruently with the serialization; this proof is enough to keep the causal or precedence relations.

For τ_1: The task τ_1 keeps the order, because it is the first task in execution and is activated in $r_1^*=0$

For τ_2: The task τ_2 also keeps the order, because it is activated C_1 time units after the activation of τ_1 and $d_2>d_1$, because $\forall C_i :C_i>0$ and $d_2 = C_1+C_2+ H > d_1 = C_1+ H$. Now, suppose the theorem is maintained for τ_k, this is:

$$d_k>d_{k-1} . \qquad (1)$$

Now, it is needed to prove that the theorem is maintained for $\tau_{k+1.}$ Taking (1) and adding C_k+H in the left part and C_{k+1}+H in the right part, the relation is kept and produces

$$d_k + C_{k+1}+H > d_{k-1}+ C_k +H . \qquad (2)$$

It is kept because $d_{k-1}+ C_k +H = d_k$ and $H+C_{k+1}>0$, and $\forall_i :C_i>0$ and $H \geq 0$

Now, substituting d_{k+1} by $d_k+C_{k+1}+H$ and d_k by $d_{k-1}+C_k+H$ in (2), the result is $d_{k+1} > d_k$ □

With this modification in the deadlines, a schedule is obtained that keeps the precedence relationship and shown in the fig. 4.

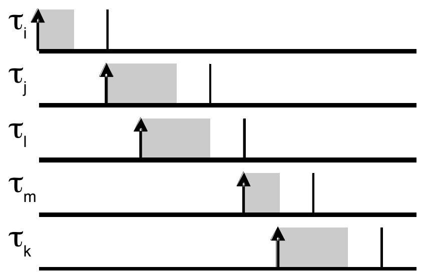

Fig. 4. Schedule that keeps the precedence relationship and time restriction (deadline)

Assigning Preemption Levels for Considering the Integration between Resource Allocations and the Precedence Handling. The next and last step is considering the resource request in tasks. The original job from [1] does not consider precedence constraints. This step modifies the Baker proposal to consider the precedence constraints.

Actually, the simple integration of EDF + the actual proposal + Baker proposal, may lead to the wrong impression that the problem is solved because time restriction is considered by EDF, precedence relations are covered by the first part of this proposal, while resource handling is covered via SRP.

This is a mistake because direct application of SRP may lead to situations where the precedence relationships are not respected as it is shown later.

To show the problem lets include a new independent task τ_v which requires a resource R in order to be executed, and τ_j request the same resource in the first computation time, then the scenario showed in fig. 5 can arise, where τ_v is executed before than τ_i although $d_v > d_i$.

This is because τ_v executes a lock (R) that produces a blocking of the task τ_i, after which τ_l arrives and preempts τ_v because $d_v > d_l$.

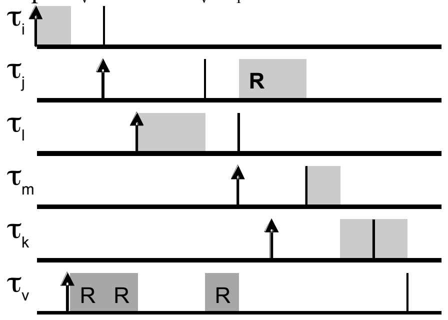

Fig. 5. The precedence constraints were broken because τ_l was executed before than τ_j, and three tasks failed to meet the deadlines.

SRP could work with EDF if preemption levels are ordered inversely with respect to the order of relative deadlines; that is $\pi_i > \pi_j \Leftrightarrow D_i < D_j$. This is because of the general definition in order to keep all the properties is required that if J_a arrives after J_b and J_a has higher priority than J_b, then J_a must have a higher preemption level than J_b. SRP by itself does not consider the precedence constraints. To achieve this, the proposal consists in changing the preemption level assignation to integrate SRP to this work.

Each graph Gi generates an execution serialized schedule σ_i. Before assigning the preemption level, for each G_i, should be considered that the preemption level assignation must be in an inverse order with respect to the deadline order presented above. From the example presented in fig. 1 and fig. 2, it is if $\Phi_i<\Phi_j<\Phi_l<\Phi_m<\Phi_k$ and $d_i<d_j<d_l<d_m<d_k$ then must be $\pi_i> \pi_j>\pi_l>\pi_m>\pi_k$ in order to keep the precedence constraints. To consider two or more directed acyclic graphs, the preemption level assignation must be combined in competetion with the best element of each graph. This is in congruence with the original strategy of SRP because it considers the relatives deadlines.

Because schedule σ_i is ordered in congruence with the internal preemption levels, the first element of σ_i is the element with the smallest preemption level. Initially a counter called pl and with a value in "1" is set, it assign its value as the preemption level to the selected element (task). Let get(σ_1,σ_2,..,σ_{n-1},σ_n) be an instruction which compares the first element in each schedule, takes the element or task with the smallest relative deadline, assigns the pl as preemption level in the selected task (this task won't be considered in the next steps) and increments pl in "1". The get instruction must be executed until that all task in τ have received a preemption level. It can be seen like a general schedule construction as is showed in fig. 6. This way of preemption levels assignment integrates SRP with the precedence management policy presented in this paper. So the execution with SRP keeps the established order in the tasks, as proved in the Theorem 2.

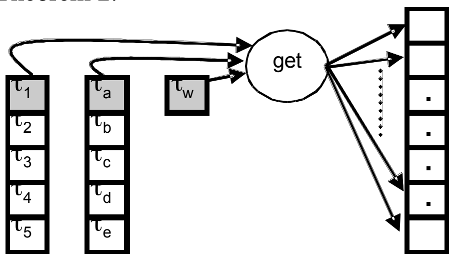

Fig. 6. The get instruction: functionality.

Lemma 1. Let τ be a set with n periodic task: τ_1, τ_2, τ_3,.., ,τ_n. Assume for all τ_i : $C_i \leq D_i \leq T_i$. Now, the precedence constraints are represented by a directed acyclic graph called G. Let be σ an ordered schedule of the entire task (τ_a-> τ_b-> τ_c-> τ_d-> τ_e) in τ with an order keeping the precedence relations, τ_a is predecessor of τ_b, τ_b is predecessor of τ_c, etc. If a decreasing order of preemption levels with respect to the graph is assigned, where the time parameters were modified to ensure precedence relationships, then a feasible schedule is produced, which considers precedence constraints and resource allocations.

Proof. In order to prove this lemma, is enough to demonstrate that the modification in the preemption levels keeps the precedence constraints. The other aspects do not change the properties of SRP. It is proceed by contradiction. Let be two tasks with precedence constraints, being τ_a the task that must be executed first and τ_b the second one, and $\pi_a > \pi_b$. Suppose that the assignation produces an execution inversion, it is τ_b is executed before than τ_a, it means that τ_a was blocked by another task, it is $\pi_a < \pi_s$. So, τ_b was executed before than τ_a, then it means than $\pi_b > \pi_a$, which leads to a contradiction.

Theorem 2. Let G_1, G_2, ..,G_n be the directed acyclic graphs that represents the precedence constraints between all tasks in τ. Each G_i is defined by at least one task and for all i.j: 1..n, $G_i \cap G_j = \{\}$. The get instruction defined above produces an assignation of preemption levels to all task of τ which produces a feasible schedule considering SRP and the proposal for managing precedence constraints.

Proof. In order to prove this theorem, it is only necessary to exhibit that the assigned preemption levels maintain the precedence rules established. The get instruction in order to construct the general assignment, for each σ_i always considers the element with the time more distant to be activated as candidate, it for assigning the preemption level; which is incremented after an assignment. The candidates of σ_1, σ_2 ,.., σ_{n-1}, σ_n competes as in SRP, by considering the relative deadline, which was proved in [1] that maintains the properties for SRP with independent tasks. This is true also in this work, because the tasks between two graphs are independent, $G_i \cap G_j = \{\}$. Now, the last thing is to demonstrate that the order in each graph keeps the precedence constraints, which was proved in Lemma 1. So, this theorem remains.

4. Conclusions and Future Work

In this paper, a policy for considering precedence constraints and resource allocation in a system with EDF has been proposed, it works in three steps: defining an execution order considering the precedence constraints, setting the execution order with modifications in the deadlines and phases of the tasks, and finally, assigning preemption levels. The last part is a refinement of SRP with a modification in the form that preemption levels are assigned. The modification consists in considering a serialized execution, which keeps the causal relationships. The proposed algorithm correctness has been demonstrated with analytical proofs; in the sense that modifying tasks deadlines and phases is enough to keep the precedence constraints with EDF. It was proved that with the proposed assignment in the preemption levels, SRP could consider precedence constraints after that a serialized schedule has been defined. This work is part of a local scheduler in a distributed scheduling mechanism. Actually a distributed simulator is in place; in this simulator the main parts of the architecture are implemented trying to simulate real conditions. The proposed algorithm is being compared with a proposal that uses resource reservation [9], [10]; the results will be presented in a future work.

References

1. Baker, T.P.: Stack-based scheduling of real-time processes. Journal of Real-Time Systems, 3 (1991)
2. Liu, J.W.S.: Issues in distributed real-time systems: Workshop on Large, distributed, parallel architecture real-time systems (1993)
3. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems, Addison Wesley Publishing (1987)
4. Horn, W.: Some simple scheduling algorithms. Naval Research Logistics Quarterly, 21, (1974)
5. DiPippo, L., Wolfe, V.F.: Real-time databases, University of Rhode Island (1995)
6. Ramamritham, K.: Real-time databases. International Journal of Distributed and Parallel Databases (1996)
7. Yu, P.S., Wu, K.L., Lin, K.J., Son, S.H.: On real-time databases: concurrency control and scheduling. IBM Watson research center, University of Illinois, University of Virginia (1994)
8. Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers (1990)
9. Abeni, L., Lipari, G., Buttazzo, G.: Constant bandwidth vs proportional share resource allocation. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, (1999)
10. Saowanee, S., Rajkumar, R.: Hierarchical reservation in resource kernels. Technical report, Electrical and Computer Engineering CM, (2001)