How to Integrate Precedence Constraints and Shared Resources in Real-Time Scheduling*

Marco Spuri†
Scuola Superiore “S.Anna”
via Carducci, 40 - 56100 Pisa (Italy)
E-mail: spuri@sssup1.sssup.it

John A. Stankovic‡
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
E-mail: stankovic@sssup2.sssup.it

Abstract

Formal results for precedence constrained, real-time scheduling of unit time tasks are extended to arbitrary timed tasks with preemption. An exact characterisation of the EDF-like schedulers that can be used to transparently enforce precedence constraints among tasks is shown. These extended results are then integrated with a well-known protocol that handles real-time scheduling of tasks with shared resources, but does not consider precedence constraints. This results in schedulability formulas for task sets which allow preemption, shared

---

*This paper has been submitted as a concise paper.

†This work has been supported, in part, by the IRI of Italy under a research grant.

‡This work has been written while this author was visiting and supported by the Scuola Superiore “S.Anna” of Pisa, and also supported, in part, by NSF under grants IRI 9208920 and CDA 8922572, and by ONR under grant N00014-92-J-1048.

resources, and precedence constraints, and a practical algorithm for many real-time uniprocessor systems.

1 Introduction

In many hard real-time systems, due to the strict deadlines that must be met, communications among tasks are implemented in a completely deterministic manner. The usual approach followed to achieve this, is to model communication requirements as precedence constraints among tasks, that is, if a task T_i has to communicate the result of its computation to another task T_j, we introduce the pair (T_i, T_j) in a partial order $\prec$, and we schedule the tasks in such a way that if $T_i \prec T_j$ the execution of T_i precedes the execution of T_j.

Good examples of this modeling can be found in the MARS operating system [8, 9], in which the basic concept of a *real-time transaction* is described exactly in this way, and in Mok's *kernelized monitor* [12], in which a *rendezvous* construct is used to handle similar situations. In both cases, shared resources among tasks are also considered. However, in the former work the whole schedule is statically generated, that is, is produced in advance before the system can run. The schedule is then stored in a table that, at run-time, is consulted by a dispatcher to actually schedule the tasks without any other computational effort. In the latter, instead, even if generated a bit more dynamically, the schedule is basically nonpreemptive, or at least we can say the preemption points are chosen very carefully, since the processor is assigned in quantums of time of fixed length equal to the size of the largest critical section.

Because preemptive systems are generally much more efficient than nonpreemptive ones, our goal is to present a simple technique with a formal basis for integrating precedence constraints and shared resources in the task model of dynamic uniprocessor systems, in which preemption is allowed.

Few protocols that handle shared resources have appeared so far [13, 3, 1, 14]. Both the Priority Ceiling Protocol [13, 3] and the Stack Resource Policy [1] will be considered in this paper. They are both well studied and characterised with respect to sufficient conditions for the schedulability of a set of tasks. However, they have been described using a simple independent task model, while we believe a more complex model including precedence constraints would be valuable.

Vice versa, several papers dealing with precedence constraints, but not with shared resources have appeared. Blazewicz [2] shows the optimality of a preemptive earliest deadline first (EDF) scheduler assuming the release times and the deadlines are modified according to the partial order among the tasks. The same technique is used by Garey *et al.* [6] to optimally schedule unit-time tasks. In [4], Chetto *et al.* show sufficient conditions for the EDF schedulability of a set of tasks, assuming the release times and the deadlines are modified as above.

Our main contributions are: an exact characterisation of EDF-like schedulers that can be used to correctly schedule precedence constrained tasks, and showing how preemptive algorithms, even those that deal with shared resources, can be easily extended to deal with precedence constraints, too. We do this by inventing the notion of *quasi–normality*, which is an extension to [6]. Furthermore, while the formal results are general, we also present a straightforward application of these results to the Priority Ceiling Protocol (PCP) and the Stack Resource Policy (SRP), developing schedulability formulas that are valid when the SRP is extended to handle both shared resources and precedence constraints.

The paper is organized as follows. In section 2, a brief description of the PCP and the SRP protocols is given. In section 3, the general results on precedence constrained tasks scheduling are presented. In section 4, as an example, we apply the general results to the PCP and the SRP. Finally, in section 5, we conclude with a brief summary.

2 Protocols Handling Shared Resources

In [13], Sha *et al.* introduce the *Priority Ceiling Protocol*(PCP), an allocation policy for shared resources which works with a Rate Monotonic scheduler [11]. Chen and Lin [3] extend the utilization of the protocol to an EDF (earliest deadline first) scheduler.

The main goal of these protocols, as other similar protocols, is to bound the usually uncontrolled priority inversion, a situation in which a higher priority task is blocked by lower priority tasks for an indefinite period of time (a block can occur if a task tries to enter a critical section already locked by some other task). Finding a bound to priority inversion allows us to evaluate the worst case blocking times eventually experienced by the tasks, so that they can be accounted for in the schedulability guaranteeing formulas. In other words, this means we can evaluate the worst case loss of performance due to blocking.

The key idea behind the PCP is to prevent multiple priority inversions by means of early blocking of tasks that could cause priority inversion, and to minimize as much as possible the length of the same priority inversion by allowing a temporary rise of the priority of the blocking task. Following the description given in [3], the PCP has two parts which define the priority ceiling of a semaphore and the handling of lock requests:

"*Ceiling Protocol.* At any time, the priority ceiling of a semaphore S, $c(S)$, is equal to the original priority of the highest priority task that currently locks or will lock the semaphore.

Locking Protocol. A task T_j requesting to lock a semaphore S can get the lock only if $pr_j > c(S_H)$, where pr_j is the priority of T_j and S_H is the semaphore with the highest priority ceiling among the semaphores currently locked by tasks other than T_j. Otherwise, T_j waits and the task T_l which has the lock on S_H inherits the priority of T_j until it unlocks S_H."

Furthermore, assuming an EDF priority assignment, a task receives a higher priority, the earlier is its deadline.

Note that a task can be blocked even if the critical section it requests is free, when there are other critical sections already locked. This is necessary to prevent a high priority task from being blocked two or more times if it wants to enter several critical sections.

The protocol has been shown to have the following properties:

- A task can be blocked at most once before it enters its first critical section.
- The PCP prevents the occurrence of deadlocks.

Of course, the former property is used to evaluate the worst case blocking times of the tasks. In particular, the schedulability formula of Liu and Layland [11] has been extended by Chen and Lin [3] to obtain the following condition.

Theorem 2.1 *A set of n periodic tasks can be scheduled by EDF using the dynamic priority ceiling protocol if the following condition is satisfied:*

$$\sum_{i=1}^{n} \frac{c_i + b_i}{p_i} \leq 1,$$

where c_i is the worst case execution time, b_i is the worst case blocking length and p_i is the period of the task T_i. □

Baker [1] describes a similar protocol, the Stack Resource Policy (SRP), that handles a more general situation in which multiunit resources, both static and dynamic priority schemes, and sharing of runtime stacks are all allowed. The protocol relies on the following two conditions:

(2.1) "To prevent deadlocks, a task should not be permitted to start until the resources currently available are sufficient to meet its maximum requirements.

(2.2) To prevent multiple priority inversion, a task should not be permitted to start until the resources currently available are sufficient to meet the maximum requirement of any single task that might preempt it."

The key idea is that when a task needs a resource which is not available, it is blocked at the time it attempts to preempt, rather than later, when it actually may need the shared resource. The main advantages of this earlier blocking are to save unnecessary context switches and the possibility of a simple and efficient implementation of the SRP by means of a stack.

The SRP has been shown to have properties similar to those of the PCP. Furthermore, assuming n tasks ordered by increasing relative deadlines, Baker [1] develops a tighter formula for a sufficient schedulability condition (a task, periodic or sporadic, has a *relative deadline* d if whenever it is released at time t it must be completed before time $t + d$; of course, it must be $d \leq p$).

Theorem 2.2 *A set of n tasks (periodic and sporadic) is schedulable by EDF scheduling with SRP semaphore locking if*

$$\forall k = 1, \ldots, n \qquad \left(\sum_{i=1}^{k} \frac{c_i}{d_i}\right) + \frac{b_k}{d_k} \leq 1.$$

□

In the rest of this paper we will assume an implementation of the SRP in which priorities are assigned to tasks using an EDF rule.

3 Basis For Precedence Constraints – Quasi-Normality

A nice analytical result concerning the integration of precedence constraints and real-time scheduling can be found in [6]. In this paper, Garey *et al.* describe a scheduling algorithm for unit-time tasks with arbitrary release times and deadlines, and precedence constraints using the concept of normality. Here, we extend their idea to more general dynamic systems using preemptive EDF schedulers without unit time constraints.

Definition 3.1 *Given a partial order $\prec$ on the tasks, we say the release times and the deadlines are* consistent *with the partial order if*

$$T_i \prec T_j \quad \Rightarrow \quad r_i \leq r_j \text{ and } d_i < d_j.$$

Note that the idea behind the consistency with a partial order is to enforce a precedence constraint by using an earlier deadline.

The following definition formalizes the concept of a preemptive EDF schedule.

Definition 3.2 *Given any schedule of a task set, we say it is* normal *(with respect to EDF) if for all portions δ_i and δ_j of two tasks T_i and T_j, respectively,*

$$s_{\delta_j} < s_{\delta_i} \quad \Rightarrow \quad d_j \leq d_i \text{ or } r_i > s_{\delta_j},$$

where s_δ is the start time of the portion δ.

What this definition says is that at any time among all those tasks eligible to execute (a task T_i is eligible for execution only if the current time t is greater than or equal to the release time r_i), we always schedule the task with the earliest deadline.

In [6] Garey *et al.* show that we can use the consistency of release times and deadlines to integrate precedence constraints into our task model; just use an algorithm that produces normal schedules. This result is proven only

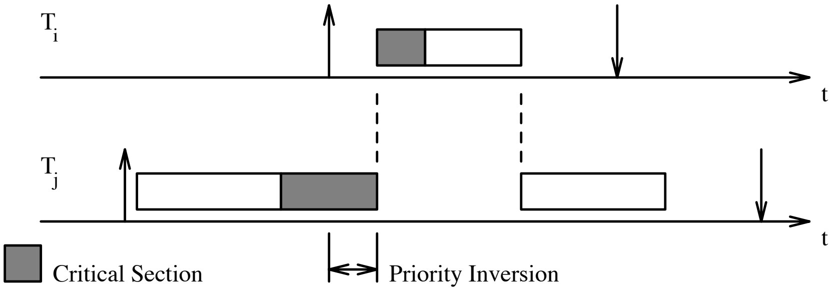

Figure 1: Example of a not normal schedule produced by PCP and SRP.

for unit-time tasks. We now extend their result to tasks of arbitrary length and running on a preemptive system.

Lemma 3.1 *If the release times and deadlines are consistent with a partial order, then any normal schedule that satisfies the release times and deadlines must also obey the partial order.*

Proof. Consider any normal one-processor schedule and suppose that $T_i \prec T_j$ but that $s_j < f_i$, where f_i is the completion time of T_i. The last expression implies that there are two portions δ_j and δ_i of T_j and T_i, respectively, such that $s_{\delta_j} < s_{\delta_i}$. Since the schedule is normal, this means that $d_j \leq d_i$ or $r_i > s_{\delta_j}$ (recall that for the feasibility assumption we have $s_{\delta_j} \geq s_j \geq r_j$). However, by the consistency assumption, we have $r_i \leq r_j$ and $d_i < d_j$; hence, in both cases we have a contradiction. □

Now the question is whether we can extend this result in order to handle the more general situation in which we have shared resources among tasks, too. Unfortunately, a direct generalization to an EDF-like scheduling algorithm, using some protocol like PCP or SRP, does not hold. In fact, in both cases, the produced schedules are not necessarily normal (see Figure 1 for an example).

The motivation is very simple: even if bounded, all these protocols allow *priority inversion*; that is, during the evolution of the system, there may be a lower priority task blocking another higher priority one. In this case, the condition for the schedule to be normal is violated.

Hence, our conclusion is that as long as shared resources are used, the normality must be weakened in some way. That is, we want a less restricting policy, with respect to scheduling decisions, but that still preserves the property of normality shown in Lemma 3.1.

Definition 3.3 *Given any schedule of a task set, we say it is* quasi–normal *(with respect to EDF) if for all portions* δ_i *and* δ_j *of two tasks* T_i *and* T_j*, respectively,*

$$r_i \leq r_j \ \ and \ \ s_{\delta_j} < s_{\delta_i} \quad \Rightarrow \quad d_j \leq d_i.$$

In other words, the definition establishes that in a quasi–normal schedule the decision of preempting a task is left to the scheduler (recall that in a normal schedule whenever there is an eligible task with an earlier deadline you are forced to preempt). However, if the scheduler chooses to preempt a task T_i and assigns the processor to a task T_j, the deadline of T_j must be earlier than the deadline of T_i (without loss of generality, we can assume that tasks with equal deadlines are scheduled in FIFO order). So with quasi–normality, we give more freedom to the scheduler (so that it can obey shared resource requirements) and we obtain a bit weaker condition, as established by the following lemma.

Lemma 3.2 *If a feasible schedule is* normal *then it is also* quasi–normal.

Proof. Consider two portions δ_i and δ_j of the tasks T_i and T_j, respectively, with $r_i \leq r_j$. If $s_{\delta_j} < s_{\delta_i}$, for the normality of the schedule we have $d_j \leq d_i$ or $r_i > s_{\delta_j}$. Since the schedule is feasible $s_{\delta_j} \geq r_j$, hence, we cannot have $r_i > \delta_j$. It follows that $d_j \leq d_i$, that is the schedule is quasi–normal. □

Note that the opposite is not true (see again figure 1 for an example of a quasi–normal but not normal schedule).

At this point, we are able to generalize the result of Lemma 3.1.

Theorem 3.1 *Given a set of tasks with release times and deadlines consistent with a partial order* $\prec$*, any feasible schedule (i.e., that satisfies both the release times and the deadlines) obeys the partial order* $\prec$ if *and* only if *is quasi–normal.*

Proof. "If". Consider any quasi–normal schedule and suppose that $T_i \prec T_j$, but $s_j < f_i$, where s_j is the start time of T_j. By the consistency assumption we have $r_i \leq r_j$ and $d_i < d_j$. Being that the schedule is quasi–normal, we have also $d_j \leq d_i$, a contradiction.

"Only if". Suppose now that the schedule obeys the partial order $\prec$ and that there are two portions δ_i and δ_j of the tasks T_i and T_j, respectively, with $r_i \leq r_j$, whose start times are $s_{\delta_j} < s_{\delta_i}$. If the condition of quasi–normality is violated, we have $d_j > d_i$. This means that the release times and the deadlines of T_i and T_j are consistent with a partial order in which T_i precedes T_j. Hence, even if $\prec$ does not contain the relation $T_i \prec T_j$, we can force it without changing the problem. But this is a contradiction to the fact that a portion of T_j precedes a portion of T_i in the given schedule. □

4 Integration of Shared Resources and Precedence

In this section we show how the PCP and the SRP can be used with an extended task model, in which precedence constraints between tasks can be specified, as well as shared resources. We start by showing that quasi–normality is the essential property of a certain EDF schedulers class. This, together with the results shown in the previous section, gives us an analytical basis for our extended protocol.

Theorem 4.1 *Any schedule produced by a policy or protocol that uses an EDF priority assignment is quasi–normal* if *and* only if *at any time t the executing task is in the set*

$$\mathcal{S}_t = \{T_j : \ r_j \leq t \ \ and \ \ pr_j \geq pr_i \ \forall T_i \ \ with \ \ r_i \leq r_j\},$$

where pr_j is the priority of task T_j.

Proof. "If". Consider two tasks T_i and T_j, with $r_i \leq r_j$ and $s_{\delta_j} < s_{\delta_i}$. At time $t = s_{\delta_j}$, by assumption $pr_j \geq pr_i$, i.e., $d_j \leq d_i$. Hence, the schedule is quasi–normal.

"Only if". At any time t consider the executing task T_j. Let $\mathcal{R}_t$ be the set of all tasks with release time less than or equal to r_j, i.e., for any task $T_i \in \mathcal{R}_t$ we have $r_i \leq r_j$. Being T_i still present in the system, at least a portion δ_i will be executed later than the portion δ_j of T_j currently executing, that is, $s_{\delta_j} < s_{\delta_i}$. For the quasi–normality of the schedule we have $d_j \leq d_i$. Hence, T_j is in $\mathcal{S}_t$. □

Note that in case of priority inversion, the condition for the schedule to be quasi–normal is not violated since the blocking task, even if it does not have the highest priority in the system, is in $\mathcal{S}_t$. Furthermore, whenever a task has entered $\mathcal{S}_t$, it does not leave the set until it completes its execution. This lets us to prove the condition of Theorem 4.1 only testing it at the beginning of each task execution.

Theorem 4.1 states a general result that together with Theorem 3.1 lets us always model precedence constraints among tasks by just enforcing consistency with respect to Definition 3.1, even in complex systems with shared resources. In what follows, we show how these considerations can be applied to a couple of well-known protocols, like the PCP and the SRP (note that the results will not change even considering a simpler protocol as the Priority Inheritance Protocol [13]).

Corollary 4.1 *Any schedule produced by the PCP, used with an EDF priority assignment, is quasi–normal.*

Proof. It is sufficient to prove that at any time the executing task is in $\mathcal{S}_t$ and then applying Theorem 4.1 we have the result. The condition is always true whenever a task begins its execution because, at this time, the task has the highest priority in the system (each task executes at a priority different from its original one only if it is blocking a higher priority task, but this cannot occur at the beginning of its execution). From that instant on, the task will always be in $\mathcal{S}_t$, until it completes its execution. □

Note that some form of priority inheritance, by lower priority tasks blocking higher priority ones, is necessary. Otherwise, we could have a situation like that shown in Figure 2, in which quasi–normality and a precedence constraint are violated because the medium priority task, which is not in $\mathcal{S}_t$, is allowed to start when the higher blocks. So, by deadline modification and some form of inheritance, we can obtain the integration of precedence constraints and shared resources.

Corollary 4.2 *Any schedule produced by the SRP, used with an EDF priority assignment, is quasi–normal.*

Proof. Again, it is sufficient to prove that at any time the executing task is in $\mathcal{S}_t$ and then applying Theorem 4.1, we have the result. For the definition of the SRP, each task execution request is blocked from starting execution until it is the oldest highest priority pending request, and both the conditions

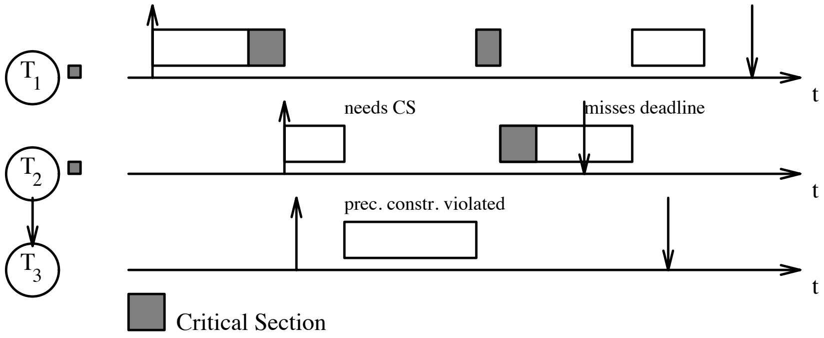

Figure 2: A situation in which an EDF scheduler without priority inheritance violates quasi–normality and precedence constraints.

2.1 and 2.2 are verified. Hence, the condition above is always true whenever a task begins its execution. The same task will leave the set S_t only at the end of its computation. □

Note that even in this case, we have a form of priority inheritance; that is, "an executing task holding a resource resists preemption as though it inherits the priority of any task that might need that resource" [1].

Finally, we show that consistency can be used with the PCP or the SRP and an EDF priority assignment to enforce precedence constraints.

Corollary 4.3 [1] *If the release times and the deadlines are consistent with a partial order, any schedule produced by both the PCP and the SRP, used with an EDF priority assignment, obeys the partial order.*

Proof. Follows directly from Corollary 4.1, Corollary 4.2 and from Theorem 3.1. □

Corollary 4.3 allows us to extend our programming model with a partial order among tasks, we only need to use a consistent assignment for release times and deadlines.

We now assume that our system is a uniprocessor and allows preemption. Priorities are assigned to tasks according to the EDF algorithm and accesses

---

[1] Note that there is a way of showing this result directly, using the properties of priority inheritance and deadline modification, as pointed out by Chia Shen in an informal correspondence with us, but here we obtain it as a simple consequence of the general results shown above.

to shared resources are controlled by the SRP (the same extended model with a slightly different analysis can be used with the PCP). The activities of the system are modelled by means of processes. We define a process $\mathcal{P}_i$ (periodic or sporadic) as a 6-*uple* $(\mathcal{T}_i, \mathcal{G}_i, P_i, D_i, C_i, \mu_i)$, where:

- $\mathcal{T}_i$ is a set of tasks that form the process,
- $\mathcal{G}_i$ is a directed acyclic graph that models a partial order among tasks in $\mathcal{T}_i$ (there is an arc from node j to node k if and only if $T_j \prec T_k$),
- P_i is the period of the process (if the process is sporadic, it is the minimum interval of time between two successive execution requests of the same process),
- D_i is the relative deadline of the process,
- C_i is its worst case computation time, that is, $C_i = \sum_{T_j \in \mathcal{T}_i} c_j$, and
- μ_i is a function that represents the maximum shared resource requirements of each task in $\mathcal{T}_i$.

Furthermore, we assume the processes arrive dynamically in the system and are dynamically scheduled.

In order to make use of the previous results, we have to enforce the consistency of the release times and the deadlines with the partial order. We can use a technique similar to those which have already appeared in several papers [2, 6, 12, 4]. Two different assignments of deadlines to tasks are proposed in this paper. They both guarantee consistency with the given partial order, but they have a different impact in terms of schedulability analysis. In the first solution, we start by assigning off-line to each task of the process $\mathcal{P}_i$ a relative deadline equal to D_i, that is,

$$d_j \leftarrow D_i \quad \forall T_j \in \mathcal{T}_i$$

and then we modify the deadlines by processing the tasks in reverse topological order:

$$d_j \leftarrow \min(\{d_j\} \cup \{d_k - c_k : T_j \prec_{\mathcal{G}_i} T_k\}),$$

where c_k is the worst case computation time of the task T_k. Note that this can be done in $O\left(\sum_{i=1}^{n} m_i + n_i\right)$, where m_i is the number of arcs in $\mathcal{G}_i$, n_i is the number of tasks in $\mathcal{T}_i$ and n is the number of processes in the system.

Then at run-time, whenever a request of execution for the process $\mathcal{P}_i$ arrives at time t, we only have to assign

$$\underline{\mathrm{r}}_j \leftarrow t,\ \underline{\mathrm{d}}_j \leftarrow t + d_j \quad \forall T_j \in \mathcal{T}_i,$$

where $\underline{\mathrm{d}}_j$ is the absolute deadline of task T_j.

Now, considering that each task T_j can be blocked if it makes use of shared resources, we have to estimate, as usual, the value b_j of its worst case blocking time. Hence, assuming we have ordered all the tasks in the system by increasing relative deadlines, we can use the formula proposed by Baker [1] to check the schedulability of the whole set:

$$\forall k = 1, \ldots, N \qquad \left(\sum_{j=1}^{k} \frac{c_j}{d_j} \right) + \frac{b_k}{d_k} \leq 1,$$

where $N = \sum_{i=1}^{n} \mid \mathcal{T}_i \mid$. Note that in this approach, the schedulability check is performed on a task basis using the modified deadlines without considering the process as a whole. If the schedulability test is positive, the formula works correctly. However, if the test is negative, it is pessimistic because of the following anomaly. When modifying deadlines of tasks on a per process basis, it is possible that tasks from different processes are interleaved. This means that a task from a process with a late deadline might execute before tasks from a process with an earlier deadline, possibly causing unnecessary missed deadlines.

We can get a tighter set of conditions using an alternative deadline assignment. We always start by assigning to each task of the process $\mathcal{P}_i$ a relative deadline equal to D_i. We then modify these deadlines according to the following argument: make the tasks within a process consistent with the given partial order, and ensure that deadlines of tasks pertaining to different processes are not interleaved. In effect, this approach uses EDF scheduling for the process as a whole, and uses modified deadlines to ensure the partial order among the tasks of the process itself. This can be easily implemented as follows.

We can avoid the mentioned interleaving, assuming that the original deadlines are expressed in terms of integer numbers. Then, it is quite simple to find for each process $\mathcal{P}_i$ a sufficiently small positive number $\delta_i < 1$ such that, modifying the deadlines by processing the tasks in reverse topological order as follows

$$d_j \leftarrow \min(\{d_j\} \ \cup \ \{d_k - \delta_i : T_j \prec_{\mathcal{G}_i} T_k\}),$$

The smallest deadline of any task of this process is greater than $D_i - 1$; and even with equal deadlines between two or more processes, there will not be interleaving between the deadlines of their tasks.

Now, during the estimation of the blocking times and the evaluation of the schedulability of the system, we can consider each process as a whole. That is, the blocking time of a process $\mathcal{P}_i$ is at most

$$B_i = \max_{T_j \in \mathcal{T}_i} b_j,$$

and, assuming again that the processes are ordered by increasing relative deadlines, the set of schedulability conditions becomes

$$\forall k = 1, \ldots, n \quad \left(\sum_{i=1}^{k} \frac{C_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1. \tag{1}$$

This formula is very similar to that proposed by Baker [1] in his schedulability analysis of the SRP. However, this one is tighter and accounts for groups of tasks with precedence constraints. Note that even though processes consist of sets of tasks with precedence constraints, the internal details of a process are kept hidden in the schedulability conditions (1).

5 Conclusions

Previous results such as the PCP and SRP protocols have been very useful for real-time systems. However, their use has been limited to situations without precedence constraints. Similarly, formal results existed for showing how to modify deadlines in a consistent manner so that a run time algorithm, such as earliest deadline scheduling, could be used without violating the precedence constraints.

In this paper, we have extended these formal results to more general dynamic systems, in which more freedom is left to the scheduler, allowing, for instance, priority inversion. As an application of these results, we have shown how to simply extend the task model used by the SRP protocol. This produces valuable results in that analytical formulas for the schedulability of task sets subject to preemption, shared resources and precedence constraints are obtained, and an algorithm that can be applied in more real-time system situations than previously is developed.

6 Acknowledgements

We would like to thank Krithi Ramamritham, Chia Shen, Fuxing Wang and Marco Di Natale for their valuable comments on this paper.

References

[1] T.P. Baker, "Stack-Based Scheduling of Realtime Processes," *Journal of Real-Time Systems*, 3, 1991.

[2] J. Blazewicz, "Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines," in E. Gelembe, H. Beilner (*eds*), "Modelling and Performance Evaluation of Computer Systems," North-Holland, Amsterdam, 1976.

[3] M. Chen and K. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," *Journal of Real-Time Systems*, 2, 1990.

[4] H. Chetto, M. Silly and T. Bouchentouf, "Dynamic Scheduling of Real-Time Tasks under Precedence Constraints," *Journal of Real-Time Systems*, 2, 1990.

[5] M.L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes," *Information Processing 74*, North-Holland Publishing Company, 1974.

[6] M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan, "Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines," *SIAM Journal Comput.*, 10(2), May 1981.

[7] J.R. Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

[8] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft and R. Zainlinger, "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach," *IEEE Micro*, February 1989.

[9] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner and W. Schütz, "The Design of Real-Time Systems: From Specification to Implementation and Verification," *Software Engineering Journal*, May 1991.

[10] E.L. Lawler, "Recent Results in the Theory of Machine Scheduling," Mathematical Programming: the State of the Art, A. Bachen *et al.* (eds.), Springer-Verlag, New York, 1983.

[11] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, 20(1), 1973.

[12] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983.

[13] L. Sha, R. Rajkumar and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Computers*, 39(9), 1990.

[14] W. Zhao, K. Ramamritham and J. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," *IEEE Trans. on Computers*, Vol. C-36, No. 8, pp. 949-960, August 1987.