Flexible Scheduling by Deadline Inheritance in Soft Real Time Kernels

Pierre G. Jansen

Dept. of Computer Science
University of Twente
Enschede, Netherlands
jansen@cs.utwente.nl

Emiel Wijgerink

Dept. of Math. and Comp. Science
Eindhoven University
Eindhoven, Netherlands
emiel@win.tue.nl

January 9, 1996

Abstract

Current Hard Real Time (HRT) kernels have their timely behaviour *guaranteed* on the cost of a rather restrictive use of the available resources. This makes HRT scheduling techniques inadequate for use in Soft Real Time (SRT) environment where we can make a considerable profit by a better and more *flexible* use of the resources. We will show that we can also improve the *responsiveness* and *fairness* of SRT kernels by combining *Real Time Transactions* and the *Earliest Deadline First* rule with *Deadline Inheritance* policies. In particular we present a policy where each task can request its own private Scheduling Quality of Service from the kernel. The kernel can offer scheduling services with varying levels of service ranging from "not attentive" to "very attentive". We claim that the proposed techniques (1) can be combined with HRT techniques such that HRT guarantees are not affected, (2) that they are particularly useful to multimedia environments, (3) that they are easy to use, and (4) that the required administration overhead is low. A currently running multimedia experiment shows a scheduling overhead of less than one percent[1].

Keywords: Real Time kernel, transactions, deadline inheritance, pre-announcements, multimedia

1 Introduction

Recent developments in the field of multimedia and communication architectures, open an exiting range of new applications. This is particularly true when combining them to new powerful systems that deal with vision, sound and control. In multimedia we have the possibility of manipulating sound and vision, robotics deals with control, while recent developments on computer and communication architectures open the possibility to distribute these functions. This opens a range of possibilities among which we find *video on demand*, *video conferencing*, *tele-teaching*, etc.. Moreover, multimedia can and will be used also in other than these typical applications. For instance in the process control environment we also see the deployment of the combination of

[1]The research is executed at the University of Twente in the Multimedia "Tukker" subproject.

remote viewing, remote hearing and remote control, mainly composed from "of the shelf" components.

All these applications have in common that they require some degree of timeliness ranging from Soft Real Time (SRT) for most multimedia applications to Hard Real Time (HRT) for process control applications. Consequently this requires an operating system kernel that can handle timeliness, so we need a Real Time (RT) kernel that can give adequate support. There are already numerous kernels on the market. None of them is a standard or a *de facto* standard for RT purposes. The reason for this is that none of them is flexible enough to support a wide range of RT applications. The basic problem is twofold:

1. When used for HRT tasks, these kernels are inevitably inflexible. HRT tasks either require off-line reservation of resources or some other off-line guarantee for schedulability of tasks. New tasks may require a time consuming re-scheduling of all the existing tasks.

2. When used for SRT tasks, scheduling of tasks and reservation of resources is far from optimal. Moving pre-calculated offline scheduling to flexible online scheduling causes phenomena like priority inversion, late reactions, and unnecessary reservation of resources.

This paper investigates the possibilities of a SRT scheduler that is flexible, fair and responsive enough to

- support multimedia continuous streams, and multimedia processes, and

- to be integrated in a HRT environment such that HRT requirements can be maintained.

In this paper we leave the HRT kernels as they are and we focus on the SRT kernels. Typical SRT applications are found in the field of multimedia, where continuous video and audio streams have to be supported. These do not require HRT requirements. Occasionally we can tolerate the loss of an image or a sound frame, although we do not like this and we will always try to avoid it. This tolerance opens the possibility of introducing flexibility in process scheduling and resource usage.

Because multimedia applications are typical applications for our SRT kernel we will give a closer look into this environment. This could be a workstation where an user is looking at a window in which a movie is playing or video conferencing is proceeding. We may expect the Continuous Media (CM) stream to be received and unpacked by the ATM AAL5 interface [22]. The unpacked but still compressed video (moving JPEG or MPEG [ISO /IEC91]) is sent to the decompression unit and from there, after optional processing, to a video processor that displays the images in time in its window. For these applications we need a SRT kernel.

RT kernels can be classified as dynamic or static [12].

- A *dynamic* kernel has no, or few information about the arrival of tasks apriori. Therefore a considerable part of the scheduling decisions have to be made at the moment a task arrives.

- A *static* kernel however has prior information about arrival times and resource usage such that scheduling decisions can be made off-line.

Static kernels are mainly used for HRT applications, while dynamic kernels more refer to SRT applications. This paper mainly concentrates on dynamic kernels. We will first give an

overview of the existing techniques in the dynamic kernels in section 2. We describe our task model in section 3, introduce the new scheduling protocols in section 4, consider performance aspects in 5 and describe the implementation in 6.

2 Existing dynamic kernels

In the dynamic kernel jobs arrive without a priori knowledge. They are scheduled according to a priority, which is typically derived from parameters as used in classical scheduling policies such as Static Priority Scheduling, Shortest Process Time, Shortest Slack Time (time to deadline minus runtime), or Earliest Deadline First (EDF). Among others Liu and Layland [25] proved that a given set of processes, scheduled by any satisfying scheduling algorithm, could also be scheduled by a deadline driven algorithm. However, without any precautions these techniques may lead to the phenomenon of *priority inversion* when other resources then the processor come into view. Priority inversion could happen when a high priority task is blocked for a resource that is held by a low priority task. The latter may not proceed due to its low priority, consequently blocking the high priority task.

Depending on the synchronisation policy such as the Fixed Priority Protocol, the Basic Inheritance Protocol (BIP), the Priority Ceiling Protocol (PCP) [35] or the Real Time Transaction Protocol (RTTP) [13] a dispatcher assigns processes to the processor(s). The Fixed Priority Protocol generally suffers from priority inversion, whereas the Basic Inheritance Protocol, the Priority Ceiling Protocol and the Real Time Transaction Protocol provide methods to bound the duration of priority inversion.

The Basic Inheritance Protocol realises this by inheritance of priority. Low priority processes, owning shared resources that are also requested by high priority processes, inherit the high priority from the waiting processes. A primary disadvantage of this scheme is the impossibility of avoiding transitive waiting and no timing guarantees whatsoever can be given.

The Priority Ceiling Protocol (PCP) avoids priority inversion and also transitive waiting. The basic idea is to make way for high priority jobs, even if it is not certain that they will become active. The rule is that a medium priority job may not preempt a low priority job if the low priority job holds resources that could be claimed by a high priority job. The priority ceiling associated with a resource is the highest priority of a job that ever can claim this resource. PCP is rather strict in preserving resources for processes of which it is not known whether they will become active. This prohibits preemption of a low priority job by a medium priority jobs, even when the high priority job seldom or never shows up.

The Real Time Transaction Protocol (RTTP) also avoids priority inversion and transitive waiting. It roughly works as follows:

When a transaction starts, it simultaneously acquires all resources it needs to complete the transaction. During the transaction resources can only be released. A transaction has completed when it has released all of them and becomes free running. A transaction is assigned a processor if it has the highest priority and when it can acquire all its requested resources. Priority inheritance is used when a high priority transaction is waiting for a low priority transaction in order to avoid preemption of the low priority transaction. In contrast to PCP, RTTP is rather generous for the lower level priority processes with respect to the preservation of resources. Resource usage of a process, say P, is only taken into account after P is released. The consequence is that a high priority process might have to wait longer than wanted before it can start.

We will now introduce a task model that can deal with continuous media and with which our scheduling policy can be described.

3 Tasks

This section will discuss a variety of protocols all based on the Earliest Deadline First rule and which are enhanced by Deadline Inheritance refinements. The protocols have been designed in such a way that:

- They are *fair* in strategy. Tasks are served in accordance to the best knowledge of the system. A system can serve a task better if it has adequate information of the task.
- They are *flexible* with respect to the use of resources. Reservations are only done when necessary.
- Tasks are served with the desired *responsiveness* and they can request their own quality of service. Resources are preserved accordingly.
- They have a low administration overhead which is necessary in order to limit consuming scarce "real" time.

We will now introduce our task model.

3.1 Task model

A task is a sequence of "free running" or "resource using" *transactions*, similar to the Real Time Transaction Protocol [13]. A free running transaction is not subject to mutex scheduling constraints since it does not use shared resources. This makes it easy to schedule these transactions. (This is discussed in 3.4 with more detail). An invocation of a "resource using transaction" can only be run if it can acquire all its resources *simultaneously.* This condition guarantees that a transaction always runs to completion[2]. Unbounded priority inversion, transitive waiting and deadlock[3] are impossible. A transaction may release its shared resources at any time. However, we assume that a transactions releases its resources when it runs to completion. Dealing with early release times is possible, however, a little more complicated and beyond the scope of this paper.

In fact a transaction is a subtask. From here on we are only referring to transactions and every time when we write task, or subtask we mean a transaction unless stated otherwise.

3.2 Task states

A task may be in one of the following states: *sleeping/pre-announced*, *ready*, or *flushed.* The ready state is split up in *new released*, *running* or *preempted.* States and state transitions are shown in figure 1.

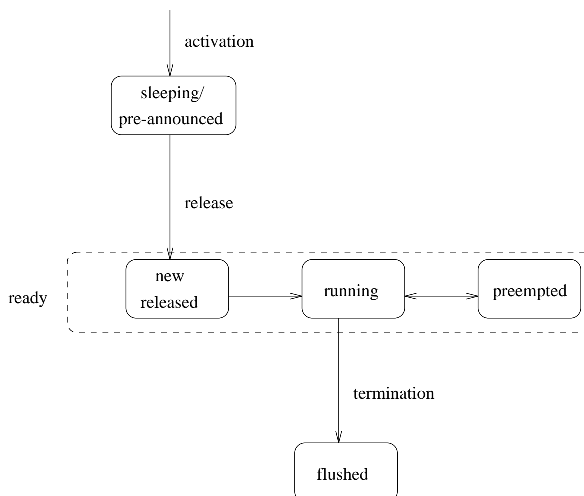

Figure 1: Task invocation states and transitions

A task is put in the administration after the oc-

[2]A consequence is that, waiting time for external events has to be bounded such that the transaction can finish before its deadline.

[3]Deadlock is a special form of transitive waiting.

currence of an (aperiodic) event from outside, e.g. an interrupt, or from an internal (periodic) event generator. A task is put in the sleeping - or in the pre-announcement administration where it is waiting for its release time. A task is only put in the pre-announcement administration when it is pre-announced, otherwise it is put in the sleeping administration. In both administrations the task is waiting for its release, after which it leaves the administrations and it enters the ready queue.

Pre-announcement is a special form of sleeping. The task is not yet active but the system takes already into account that the task will show up within a finite time. Pre-announcement will be explained in detail in section 4.3.

After its release a task will go to the *ready* state and when it is terminated to the *flushed* state. In the *ready* state a task can be running when it has the processor or *new-released/preempted* when not.

When a task is finished it is withdrawn from the administration. Periodic tasks are put into the administration by the periodic event generator.

In general the scheduling of tasks is subject to constraints. We will define the constraints and show in the following sections that our proposed scheduling algorithms will keep to the constraints.

3.3 Timing attributes

The following task timing attributes of a task T are event times. They are used for scheduling decisions.

- A pre-announcement time $p(T)$ is the absolute time at which a task can make itself known to the scheduler administration.
- A release time $r(T)$ is the absolute time from which a task may run.
- A deadline $d(T)$ is the absolute time at which a task has to be completed.

A RD-interval of a task T is the interval between release time and deadline: $d(T) - r(T)$. A PD-interval of a task T is the interval between pre-announcement time and deadline: $p(T) - r(T)$.
The set of resources which are in use by a task T is denoted by $s(T)$.

A run time interval[4] is the upper-bound of processor time which the invocatio of the task is supposed to consume.

3.4 Scheduling Constraints

The invocation of a task is subject to constraints resulting from the communications links, release times, deadlines, and resource requirements. Denote the release time and deadline of a task T as $r(T)$ and $d(T)$ respectively. When a task T_a is executed before T_b then there exists a precedence relation between them, denoted by $T_a \prec T_b$. The five types of scheduling constraints are:

Precedence: If T_a must precede T_b then there exist a precedence constraint between them and $T_a \prec T_b$ must be enforced.

Release: A task T_a cannot start before its release time $r(T_a)$.

Deadline: A task T_a is supposed to be finished before its deadline $d(T_a)$.

Mutex: If two tasks T_a and T_b require the same resource ($s(T_a) \cap s(T_b) \neq \emptyset$), then they are not allowed to preempt each other.

[4]Run times are only used in conjunction with HRT requirements. They are not needed in a SRT system and henceforth are not used in this paper.

Equality: A task with a deadline equal to a running task may not preempt the running task.

Note that the *mutex constraint* enforces mutual exclusion and that if a scheduling algorithm keeps to mutex constraints *no* additional synchronisation is required.

A mutex constraint can also be defined in terms of precedence constraints: the invocation of T_a must precede the invocation of T_b *or vice versa.* The invocations of both tasks have to execute in some sequential order that is not specified by the constraints. This can however only be used in a static environment where both tasks are known to exist and have known timing parameters.

Note that the following two relations always must hold:
$T_a \prec T_b$ implies $r(T_a) \leq r(T_b)$ (release time ordering)
$T_a \prec T_b$ implies $d(T_a) \leq d(T_b)$ (deadline ordering)

4 The proposed scheduling scheme

In this section we present the design of an integrated scheduling scheme to improve

Responsiveness towards tasks with short deadlines,

Fairness towards all the supported applications: Tasks are scheduled according to best knowledge of the system. Timing parameters and resource usage are taken into account.

Flexibility by offering a scheduling service of choice per task at any desired moment[5].

[5]This can be offline (static), always (dynamic), at pre-announcement time or at release time.

The scheduling scheme basically uses the combination of RT transactions, the Earliest Deadline First (EDF) rule and deadline inheritance protocols, with or without pre-announcements. Before discussing the scheme we first present the EDF rules and the inheritance protocols.

4.1 Earliest Deadline First (EDF)

The EDF strategy is bound to the following rules:

1. The invocations are subject to the release, mutex and equality constraints.

2. The algorithm executes one invocation when runnable invocations are available. The executing invocation has the minimum deadline of all runnable invocations, which is enforced by preemption if this were not the case.

We emphasize that the EDF algorithm maintains the precedence constraints according to given deadlines.

A direct consequence of our EDF scheduling strategy is the following: an invocation of a task T_a can only preempt another invocation of a task T_b if the RD-interval of the former completely comprises the RD-interval of the latter (i.e. the invocation T_a has a greater release time and a shorter deadline than the invocation of T_b).

Note that if the sizes of the two RD-intervals are equal, preemption is never possible. A special case is that two invocations of a single task can never preempt each other.

EDF does not solve the problem of deadline inversion when shared resources come into play. The following two protocols will reduce the deadline inversion time by deadline inheritance.

4.2 Inheritance protocols

This subsection presents two new inheritance protocols on top of EDF: the Basic Deadline Inheritance Protocol (BDIP) and the Deadline Ceiling Inheritance Protocol (DCIP). The first one has similarity with the Basic Priority Inheritance Protocol (BPIP) and the second one with the Priority Ceiling Protocol (PCP) as described in section 2. In combination with EDF these inheritance protocols maintain all predefined constraints.
The main differences with the original protocols are:

1. The original BPIP and PCP are defined in terms of static priorities, while the new protocols are based on dynamic deadlines.
2. The new task invocations are treated as *transactions* as introduced in subsection 3.1.
3. The protocols are a property of a task and not of the scheduler. Consequently each task is treated by its protocol of choice. This will be explained in subsection 4.4

4.2.1 Basic Deadline Inheritance Protocol

Definition 1 (BDIP) *BDIP is defined by the following rules for the set of runnable tasks:*

1. *According to EDF the runnable task with the shortest deadline is selected to run.*
2. *Deadline inheritance occurs at the release time of each invocation of a task, say T_a.*
3. *Each preempted or running invocation of, say T_b, that has overlapping resource requirements ($s(T_a) \cap s(T_b) \neq \emptyset$) is subject to the inheritance of the deadline of T_a.*
4. *The invocation of T_b is assigned a deadline that is equal to the minimum of the current deadline of T_b and the inherited deadline of T_a.*

Note that a task has an *original* deadline. In first instance an invocation will run under this deadline unless it inherits a smaller one. The deadline under which an invocation runs determines its preemptability and can be indicated as its *preemption deadline.* It determines the order in which the invocations are scheduled by the EDF rule. In particular, a shorter *preemption deadline* reduces the preemptability of an invocation.

Figure 2 shows an example where two tasks require the same resource R. The shortest deadline is shown by the darker shading. At time $e_1 + r_e(T_1)$ the invocation of task T_2 inherits the deadline $e_1 + d_e(T_1)$ from the invocation of T_1. This is the preemption deadline of T_2 and is shown by a dotted line on the time axis of T_2. Note that the release times $r_e(T_i)$ and deadlines $d_e(T_i)$ are relative to the respective event e_i. However $e_i+r_e(T_i)$ and $e_i+d_e(T_i)$ are absolute values.

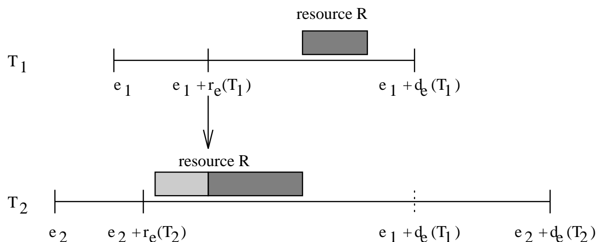

Figure 2: Basic deadline inheritance

Note that the combination of EDF and BDIP maintains the mutex constraint by the equality constraint automatically: after the deadline inheritance, invocations that have overlapping resource requirements have equal preemption deadlines. This makes the mutex constraint redundant and the EDF/BDIP algorithm straightforward: after deadline inheritance the only scheduling and preemption parameters are the deadlines.

EDF/BDIP is rather generous to other tasks in handling resources. This is because the resource needs for future tasks are only taken into account after their release.

We will now show the EDF/DCI protocol, which is more reserved to other tasks in handling resources.

4.2.2 Dynamic Ceiling Inheritance Protocol

DCIP is based on the notions of the *ceiling* of a resource and the related *execution deadline* of a task.

Definition 2 (ceiling) *The ceiling $c(R)$ of a resource R is defined as the size of the shortest RD-interval $d(T) - r(T)$ of all tasks T that require the resource.*

Note that the difference of absolute values $d(T) - r(T)$ is equal to the difference of their relative representations $d_e(T) - r_e(T)$. Note also that the difference $c(R)$ is a relative value.

Definition 3 (execution deadline) *The execution deadline of a task T is $t + mc(T)$ where t is the start time of a task T and $mc(T)$ is the minimum of all ceilings of resources that are required by T.*

Note that the start time t and the execution deadline $t + mc(T)$ are absolute values.

Definition 4 (DCIP) *DCIP is defined by the following rules:*

1. *According to EDF the runnable task T with the shortest deadline is selected to run.*

2. *When an invocation of a task T starts to execute at time t it computes the execution deadline $t + mc(T)$.*

3. *The invocation of T is assigned a deadline that is equal to the minimum of the original deadline and the execution deadline.*

Note that these rules put heavy preemption restrictions on other tasks. If all these tasks were activated immediately after time t, then BDIP would compute the same new preemption deadlines as DCIP.

In the example of figure 3 an invocation of task T_2 starts to execute when it has the smallest deadline at time t. At this point the execution deadline is determined. In this example T_2 inherits via (the only) resource R the ceiling $c(R)$ which in this case is equal to the RD-interval of its only user $d_e(T_1) - r_e(T_1)$. Consequently the execution deadline of T_2 is $t + c(R)$. An invocation of T_1 that is released later, will not preempt the invocation of task T_2 by EDF, because T_1 has a later deadline $e_1 + d_e(T_1)$ than the execution deadline of T_2.

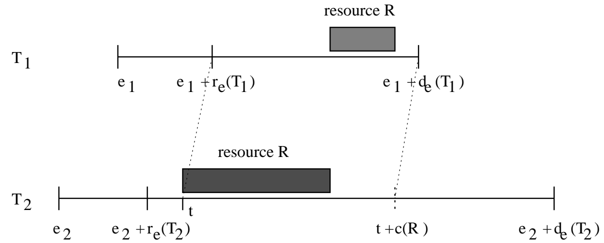

Figure 3: Dynamic ceiling inheritance

We emphasize that the combination of EDF and DCIP maintains the mutex constraint indirectly. Assume that an invocation T_b that uses resource R starts to execute at time t. In addition, task T_a has the shortest RD-interval of all tasks that use resource R. This determines the deadline of T_b which is $t + mc(T_b)$ with $mc(T_b) = c(R) = d_e(T_a) - r_e(T_a)$. If an invocation of T_a is released at time $s \geq t$, the deadline of this invocation is equal to $s + mc(T_a)$ with $mc(T_a) = c(R)$. Since this makes $s + mc(T_a) \geq t + mc(T_b)$ EDF will not preempt T_b. As in BDIP this makes the mutex

constraint redundant and the scheduling algorithm becomes straightforward: after deadline inheritance the only scheduling and preemption parameters are the deadlines.

Under particular circumstances[6] EDF/DCIP restricts other processes more than the original PCP.

A policy similar to EDF/DCIP is also used in the YARTOS kernel [16, 17]. However the task model of YARTOS is more restricted than our model.

4.2.3 Comparing BDIP and DCIP

Both protocols maintain the mutex constraint by implication from the inheritance rule and the equality constraint.

DCIP's ceilings are static and they may cause tasks that have inherited a ceiling to be non-preemptable, even in case that the task for which the non-preemptability has been organised does not show up. One may also consider this as an improper since it gives the inheriting tasks processing rights which are in fact not deserved.

BDIP has a dynamic character and takes only information of tasks into account after they have been released. This may cause longer waiting times for newly released tasks, even if they have short deadlines.

DCIP has a smaller *run-time overhead* than BDIP because the ceilings and execution deadlines can be computed off-line. The inheritance of the execution deadline in DCIP only requires one assignment statement per invocation. Deadline inheritance in BDIP includes a search for overlapping resource requirements over the preempted and running invocations.

Figure 4 shows an example of a situation where BDIP can schedule a task while DCIP cannot. DCIP anticipates to an invocation of a task T_3 that requires resource R and has a RD-interval of size $mc(T_2)$ and therefore T_2 is not preemptable by T_1. In BDIP T_1 may preempt T_2 because T_1 has the shortest deadline and T_1 and T_2 have no common resource needs.

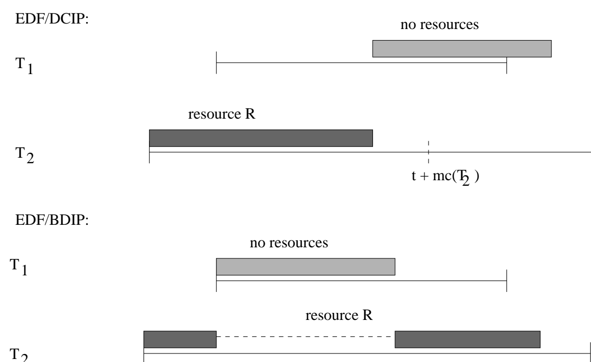

Figure 4: EDF/DCIP versus EDF/BDIP

4.3 Pre-announcements

Both BDIP and DCIP do not use all the resource and timing information of sleeping tasks. With pre-announcements this information can be made available at an earlier time and the scheduler can exploit this to make more reasonable schedules. Pre-announcements can be used in combination with BDIP as well as with DCIP, resulting in two new protocols: BDIP&PA and DCIP&PA. Pre-announcement occurs at activation-time, at which timing - and resource information of the invocation is put in a PA-database.

Under BDIP&PA an invocation requests the scheduler to be aware of its future resource usage. The BDIP&PA protocol will offer a better response then BDIP for the requesting invocation and a lesser preemptability for other invocations.

Under DCIP&PA tasks contribute the *Pre-announce-Deadline* (PD) interval instead of the RD interval to the deadline ceilings of the used resources. Note that since $RD \leq PD$,

[6] A task that is free running and does not (yet) use resources may preempt in PCP when it has a higher priority.

higher deadline ceilings are produced than in plain DCIP and consequently higher execution deadlines, resulting in a better preemptability.

Definition 5 (BDIP&PA) *Pre-announcement in BDIP&PA is defined by the following rules:*

1. *While an invocation is sleeping, the absolute deadline of the invocation is recorded in the PA database. The entry in the PA database is created at activation time and removed at release time of an invocation.*
2. *When an invocation starts to execute, it determines the minimum, absolute deadline of all invocations in the database that have competing resource requirements.*
3. *The new deadline of the invocation is set to the minimum of the* original *deadline and the deadline read in the database.*

For DCIP&PA it turns out that the set of rules is quite similar, only rule 3 differs:

Definition 6 (DCIP&PA) *Pre-announcement in DCIP&PA is defined by the following rules:*

1. *While an invocation is sleeping, the absolute deadline of the invocation is recorded in the PA database. The entry in the PA database is created at activation time and removed at release time of an invocation.*
2. *When an invocation starts to execute, it determines the minimum, absolute deadline of all invocations in the database that have competing resource requirements.*
3. *The new deadline of the invocation is set to the minimum of the* original *deadline, the* execution *deadline and the deadline read in the database.*

We will illustrate the behaviour of the protocols by the following examples (where the PD-interval of a task T_i is denoted by $d_e(T_i)$).

BDIP&PA: In figure 5, an invocation of task T_2 is inheriting a shorter deadline from the PA database when it starts to execute. Under BDIP the invocation of T_2 would not have inherited the deadline of T_1 until the release time of the invocation of T_1. In the BDIP&PA scheme, invocations that have a deadline in the interval $[e_1 + d_e(T_1), e_2 + d_e(T_2)]$ cannot preempt the invocation of task T_2 which improves the responsiveness of the system to T_1.

DCIP&PA: In figure 5, under DCIP, the invocation of T_2 would have inherited an execution deadline $t + mc(T_2)$ at the start of execution (time t). The invocation T_2 then becomes not preemptable for invocations that have a deadline in the interval $[t + mc(T_2), e_1 + d_e(T_1)\ ]$. DCIP&PA tolerates preemptions of invocations with deadlines in this interval.

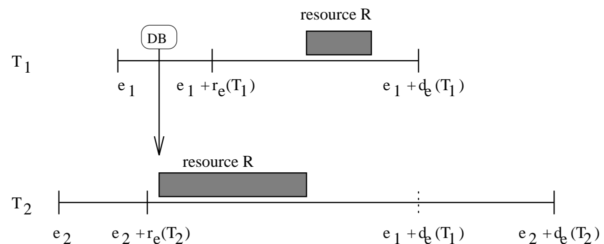

Figure 5: Pre-announcement

The following examples show what happens if the pre-announcement of T_1 comes later than the activation of T_2 at time t.

BDIP&PA: Figure 6 shows how T_2 inherits a deadline $e_1 + d_e(T_1)$ at time e_1 from T_1. Note also that the mutex constraints are not violated since T_1 cannot preempt T_2 because of equal deadlines.

DCIP&PA: Figure 7 shows what happens if T_1 is announced after T_2 has become active. At time t, T_2 inherits the execution deadline of the (only) resource R, which is $t + c(R)$. R got its ceiling from its most critical user T_1, so $c(R) = d_e(T_1)$.

The invocation of task T_1 cannot preempt the invocation of task T_2 by EDF because the deadline of T_2 is smaller than the deadline of T_1 $(t + c(R) < e_1 + d_e(T_1)$.

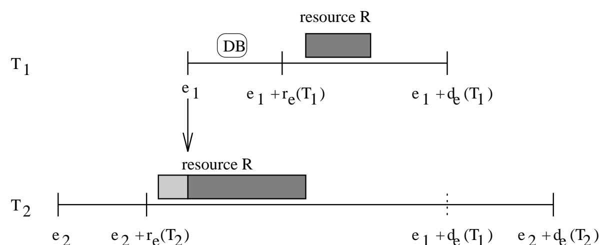

Figure 6: Pre-announcement and deadline inheritance

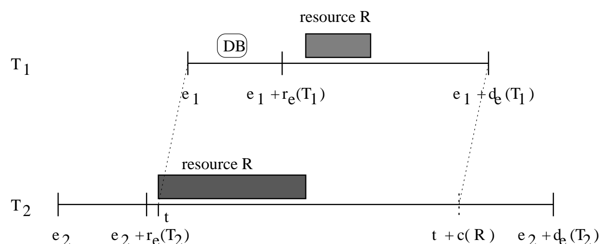

Figure 7: Pre-announcement and ceiling inheritance

We have to make a few remarks about a running invocations with increasing deadlines.

1. In the case of DCIP&PA as given in figure 7 it would be reasonable to increase the deadline. At time e_1 the information about T_1 becomes known and the preemption deadline of T_2 could increase from $t + d_e(T_1)$ to $e_1 + d_e(T_1)$.

2. Another case of deadline increase of an invocation T_1 would occur if $mc(T_1) = c(R)$ and when T_1 releases a resource R before the end of the transaction (see also 3.1).

When an invocation increases its deadline, preemption in favour of an invocation with a shorter deadline becomes possible. We are aware of this phenomenon but have not yet investigated the implementation consequences thoroughly. In the context of this paper we do not increase preemption deadlines of ready or running invocations and we will leave it as a research subject for the near future.

4.4 The integrated scheduling scheme

It is possible to make a kernel that can offer any task a protocol of choice. Four scheduling types can be offered simultaneously: BDIP, DCIP, BDIP&PA, and DCIP&PA.

The following rules are used to schedule a task set of mixed scheduling types:

1. All tasks will have an execution deadline. The overhead of assigning the execution deadline to an invocation is small and can be done off-line.

2. The execution deadline of a BDIP or BDIP&PA task is set equal to its deadline. Both types do not contribute to the computation of the ceilings and execution deadlines.

3. Only tasks of type BDIP&PA or DCIP&PA write the PA database, but all tasks read the database.

4. Only DCIP and DCIP&PA tasks contribute to the ceiling of the resources: DCIP contributes its RD interval and DCIP&PA contributes its PD interval.

Note that only BDIP and BDIP&PA tasks perform basic deadline inheritance. This involves

a small searching overhead which is only required for these tasks.

It is beyond the scope of this paper to describe in detail the combined protocol. Tasks that have a small RD-interval can be given the type DCIP&PA or BDIP to increase the resource ceilings and consequently the preemptability. We conclude this section with the remark that the ceiling protocols have the smallest administration overhead. They are however also the least flexible since they require static knowledge of the involved tasks.

4.5 Readers-Writers synchronisation

All proposed protocols can be refined by adding readers-writers synchronisation to the scheduling policy. This is possible by partitioning the set of required resources per task into two sets: the resources that have to be read and resources that have to be written. Deadline inheritance occurs in case of a read-write or a write-write conflict. Its beyond the scope of this paper to explain the details of these refinements and we conclude by the remark that readers-writers synchronisation will improve the quality of the system without violating the mutex constraint.

5 Performance

In this section we discuss the performance aspects of our kernel as flexibility, fairness, response and speed. We will neither give a formal description of these aspects give a formal proof of our statements. Both are beyond the context of this paper. Below follows an informal justification of our claims about scheduling algorithms in the introduction.

They are fair.
They are fair in the following sense: Due to the application of the EDF rule the scheduler will try to serve the tasks with the shortest deadline best. Resource needs and timing requirements of other tasks can and will be taken into account if these have handed this information to the kernel. By using priority inheritance the kernel avoids *priority inversion* to its best knowledge and in accordance with the requested scheduling protocol[7].

They offer a flexible response.
A task with a short deadline expects a fast response, that is the disposal of the processor, from the kernel. Transitive waiting and consequently deadlocks are impossible due to the introduction of RT transactions. The system will deliver a fast response if the requesting task has (1) a short deadline, (2) it gives its information early enough and (3) if the system is not burdened with a heavy administration overhead. (1) and (2) are covered by EDF/inheritance and pre-announcements, while (3) is covered by the transaction-form of tasks. It is in particular the RT transaction that makes the system administration overhead low. There are however still some possibilities to improve the response times. These are (1) to relax the behaviour of the RT transactions and take the early release of resources into account (see 4.3) and (2) to use (maximum) run-time information of the tasks. These possibilities make the algorithm more complicated and the question is whether the costs weigh against the profits. We will leave this question to future research.

They are fast.
As stated already in the previous paragraph, this is mainly due to the nature of the RT transaction in combination with EDF/inheritance. This combination guar-

[7]There is however (still) one exception, described in section 4.3 under DCIP&PA, where the kernel could do better.

antees mutual exclusion, which has not to be implemented explicitly anymore. The DCIP-algorithms are the fastest. They only[8] require ordering to deadline of ready tasks and determination of the execution deadline when they are selected. The BDIP-algorithms also require ordering to deadline of ready tasks and in addition a $O(n)$ scan over the ready tasks for determination of the inherited deadlines. The PA variants require in addition an $O(n)$ search over the administration entries for writing the PA-administration. Our first implementation, described in the following section, showed an overhead below 1 percent for the executed multimedia experiments.

6 Implementation and tests

Scheduling strategies have been simulated in Python [33] first. Then they have been added to the Nemo kernel [12], where they run at the highest priority level. The original scheduler runs in the background. The assembly code for interrupt handling and context switching is borrowed from a previous project [31].

In this section we mainly refer to continuous media applications. In [41], a mapping from a continuous media application to a task-transaction model is described in detail. It roughly comprises the following items.

- A continuous media application can be implemented *quickly* by using the building blocks of tasks, connections, resources, and devices.
- Continuous media streams can be mapped to connection oriented communication.
- Periodic and aperiodic tasks are supported in a uniform way. This enables the use of the kernel in a general purpose environment.
- Computational jitter and delay of the tasks can be mapped to release and deadline constraints.
- Tasks can share global resources. Tasks can access a resource read-only or read-write.
- All constraints are explicit and can be visualised in a task graph. This increases the predictability and reliability of CM applications.

Two applications have been written to test the scheduler and determine the administrations overhead.

The "oscilloscope" application consists of four task that draw horizontal lines on the screen. It displays when the tasks run and how they preempt each other.

The "jumping tea can" applications presents four video streams of a bouncing tea can. The stream consists of 25 frames of 256x256 pixels in 256 colours at a rate of 25 frames per second. One task computes the frame number and the position and a second task draws the picture. Four additional empty dummy tasks that are activated every 50 ms increase the workload of the scheduler.

The system is running on a MIPS 3000. Tests have run under DCIP, BDIP and under DCIP&PA. Combination of multiple task types has not yet been tested. Each invocation requires two task switches (call and return). An empty invocation (two task switches) is timed 3-4 μs. The average computational load of the tasks was 90.31%.

The overhead is 0.42% with DCIP, 0.45% with BDIP, and 0.71% with DCIP&PA. The overhead of BDIP is relatively small because the small average length of the ready queue (including the running invocation) is 1-2 invocations.

[8] Determination of resource ceilings is done offdline.

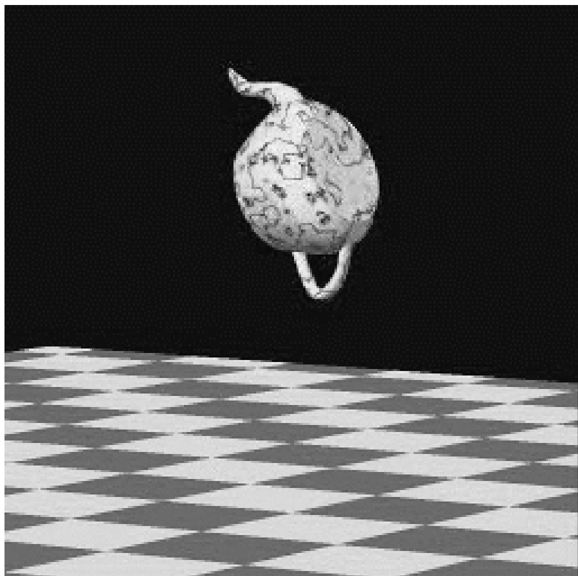

Figure 8: The 6th frame of the jumping tea can

7 Conclusions

We have proposed a class of Soft Real Time (SRT) scheduling techniques with properties that are powerful enough to be used for a general purpose Real Time kernel. By experiment we have shown that the used techniques are particularly appropriate for continuous media where SRT *flexibility*, *fairness* and *responsiveness* are as worthwhile as a statistical guarantee for *schedulability*.

All offered scheduling policies (1) decrease the priority inversion time according to the best knowledge of the scheduler and (2) avoid transitive waiting for resources and consequently deadlock.

We presented four scheduling policies, of which the basic ingredients are: (1) Real Time Transactions, (2) The Earliest Deadline First rule (3) Deadline Inheritance and (4) Pre-announcements. With these ingredients we are able to build a flexible scheduler offering a wide variety of scheduling services. The basic services offered are (1) Basic Deadline Inheritance Protocol and (2) Deadline Ceiling Inheritance Protocol. The first one offers a high - and the second one a moderate quality of service. A task can, as a service requester, give additional pre-announcement information to the scheduler. The scheduler can use this information to further refine the delivered quality of service for both protocols. It is possible to integrate the services in the kernel such that each task can request any protocol of choice.

HRT schedulability guarantees are beyond the scope of this paper. However we state that the proposed SRT policies can easily be integrated in an environment that can support HRT guarantees. The basic idea is to apply SRT policies, to resources left after satisfying HRT guarantees, without affecting the latter ones.

Experiments with a first prototype have shown that the scheduling overhead is low. This low overhead is due to the orthogonality of the ingredients which enable a systematic implementation. Among others, mutual exclusion is guaranteed by the aforementioned ingredients (1) to (3). No additional synchronisation primitives are needed.

Measurements have shown that the scheduling overhead is less then one (1) percent for all multimedia applications. This is really a low price for the offered services.

Acknowledgments

Acknowledgments are due to Eelco Klaver, Michiel Pelt and Koos Wiegman for there contributions during the Wednesday afternoon sessions, and to Pallapa Venkataram, Berend Tel and Rob van de Wetering for comments on the manuscript.

References

[1] D.P. Anderson, R. Govindan, et al., "Integrated Digital Continuous Media: A

Framework Based on Mach, X11, and TCP/IP", *Tech. Rep. 566,* Univ. of Calif. Berkeley, 1990.

[2] N.C. Audsley, A. Burns, et al., "Hard Real-Time Scheduling: The Deadline Monotonic Approach," *Internal Rep,* Dep. of Comp. Science, Univ. of York, 1991.

[3] N.C. Audsley, A. Burns, et al., "Integrating Best Effort and Fixed Priority Scheduling," *Internal Rep,* Dept. of Comp. Science, Univ. of York, 1994.

[4] A. Burns and A.J. Wellings, "Real-Time Systems and their Programming Languages," *Addison-Wesley*, 1990.

[5] A. Burns, K. Tindell and A.J. Wellings, "Fixed Priority Scheduling with Deadlines Prior to Completion," *Internal Rep,* Dep. of Comp. Science, Univ. of York, 1994.

[6] G.C. Buttazzo, "HARTIK: A Real-Time Kernel for Robotics Applications," *Proc. of the 14th IEEE Real-Time Sys. Symp,* pp. 201-205, 1993.

[7] M.I. Chen and K.J. Lin, "Dynamic Priority Ceilings: A concurrency Control Protocol for Real-Time Systems", *Real-Time Systems,* Vol.2, pp. 325-346, 1990.

[8] R.B. Dannenberg, D.B. Anderson, et al., "Performance Measurements of the Multimedia Testbed on Real-Time Mach." *Tech. Rep. CMU-CS-94-141,* Carnegie Mellon Univ, 1994.

[9] R.I. Davis, K.W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed Priority Preemptive Systems," *Proc. of the 14th IEEE Real-Time Sys. Symp,* pp. 222-231, 1993.

[10] D.B. Golub, "Operating System Support for Coexistence of Real-Time and Conventional Scheduling," *Internal Rep. CMU-CS-94,* Carnegie Mellon Univ., 1994.

[11] R. Govindan and D.P. Anderson, Scheduling and IPC Mechanisms for Continuous Media. *13th ACM Symp. on OS Principles,* pp. 68-80, 1991.

[12] E.A. Hyden, "Operating System Support for Quality of Service", *Ph.D. Thesis,* University of Cambridge, Febr. 1994.

[13] P.G. Jansen and F. Gansevles, "Priority Inheritance in Real-Time Micro Kernels", *International Symp. on Comp. and Inf. Sciences VII,* Antalya, Turkey, Nov. 1992, pp. 211-220.

[14] P. G. Jansen and P. Sijben, "Real-Time in Multimedia: Opportunistic Scheduling or Quality of Service Contracts?" *ISMM Conf. on Distributed Multimedia Systems and Applications,* Honolulu, Hawaii, Aug. 1994, pp. 194-197.

[15] K.Jeffay, D.F. Stanat, C.U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks" Proc. of the 12th IEEE Real-Time Sys. Symp, pp. 129-139, 1991.

[16] K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard Real-Time Systems," *Proc. of the* 13th *IEEE Real-Time Sys. Symp.,* pp. 89-99, 1992.

[17] K. Jeffay, D.L. Stone and F.D. Smith, "Kernel support for live digital audio and video," *Computer Communications,* Vol. 15, No. 6, pp. 388-395, 1992.

[18] E.D. Jensen, "The Kernel Computational Model of the Alpha Real-Time Distributed OS," *Mission Critical Oper. Sys.,* IOS Press, pp. 179-207, 1992.

[19] C. Kim, and S.W. Kang, "A Media Synchronization Scheme for Distributed Multimedia Systems," *Multimedia Symp. Hawaii,* pp. 163-166, 1994.

[20] G. Koren and D. Shasha, "An Optimal Scheduling Algorithm with a Competitive Factor for Real-Time Systems," *Techn.Rep. 572,* NYU, 1991.

[21] G. Koren and D. Shasha, "Dover: An Optimal On-line Scheduling Algorithm for Overloaded Real-Time Systems," *Proc. of the IEEE Real-Time Sys. Symp,* pp. 290-, 1992.

[22] J.Y. Le Boudec, "The Asynchronous Transfer Mode: A tutorial" *Comp. Networks and ISDN Systems,* Vol. 24, 1992, pp. 279-309.

[23] J.P. Lehoczky, L. Sha, and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour," *Proc. IEEE Real-Time Syst. Symp.,* pp. 166-171, 1989.

[24] J.P. Lehoczky, and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. of the IEEE Real-Time Sys. Symp,* pp. 110-123, 1992.

[25] C.L. Liu and J.W. Layland,"Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* Vol. 20, pp. 40-61, 1973.

[26] S.L.A. Lo, N.C. Hutchinson and S.T. Chanson, "Architectural Considerations in the Design of Real-Time Kernels," *Proc. of the IEEE Real-Time Sys. Symp,* pp. 138-147, 1993.

[27] H. Massalin, "Synthesis: An efficient implementation of fundamental operating system services," *Ph.D. Thesis,* Columbia University, 1992.

[28] H. Mei, Scheduling Multimedia Tasks with Timing Constraints on Distributed Systems. *Multimedia Symposium Hawaii,* pp. 180-185, 1994.

[29] J. M. Nakajima Yazaki, H. Matsumoto, "Multimedia/Realtime Extensions for the Mach Operating System," *USENIX Workshop on Micro-Kernels and other Kernel Arch.,* pp. 183-197, Summer 1991.

[30] D. Niehaus, "Program Representation and Translation for Predictable Real-Time Systems," *Proc. of the 12th IEEE Real-Time Sys. Symp,* pp. 53-63, 1991.

[31] M. Pelt and K. Wiegman, "Design and Implementation of a Soft Realtime Dispatcher," *Master's thesis,* Univ. of Twente, May 1995.

[32] T. Roscoe, "The Structure of a Multi-Service Operating System" *PhD thesis,* Cambridge, April 1995.

[33] G. van Rossum, "Python Reference Manual" *Dept. CST,* CWI, Amsterdam, 11 Oct 1994.

[34] L. Sha and J.P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *J. of Real-Time Sys.,* Vol. 1, pp. 27-69, 1989.

[35] L. Sha, R. Rajkumar, J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Comp.,* Vol. 39, No. 9, pp. 1175-1185, 1990.

[36] P. Sijben, "Description of the Scheduling Algorithm" *Internal Report, Comp. Sc.,* Univ. Twente, Dec 1994.

[37] B. Sprunt and L. Sha, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System," *Internal Rep. CMU/SEI-89-TR-11,* Carnegie Mellon Univ., 1989.

[38] J.A. Stankovic and K. Ramaritham, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software,* May 1991, pp. 62-72, 1991.

[39] J.A. Stankovic and K. Ramaritham, "The Spring Kernel," *Mission Critical Oper. Sys.,* IOS Press 1992, pp 86-117, 1992.

[40] H. Tokuda and C.W. Mercer, "The ARTS Kernel: Toward Predictable Distributed Real-Time Systems," *Mission Critical Oper. Sys.,* IOS Press, pp. 118-130, 1992.

[41] E.M.G. Wijgerink, "Flexible Scheduling by Deadline Inheritance in Soft Real Time Kernels," *Master's thesis,* Univ. of Twente, Aug. 1995.