Incorporating the Deadline Floor Protocol in Ada

Mario Aldea[1], Alan Burns[2], Marina Gutiérrez[1] and Michael González Harbour[1]

[1]*Universidad de Cantabria*
{aldeam, gutierrezlm, mgh}@unican.es
[2]*University of York*
alan.burns@york.ac.uk

Abstract

The Ada 2005 standard introduced "Earliest Deadline First" (EDF) as one of the supported dispatching policies. The standard specifies the "Stack Resource Protocol" (SRP) as the protocol for resource sharing among EDF tasks. During the time the SRP has been in the standard it has shown to be a relatively complex protocol. Recently, a new protocol has been proposed for resource sharing in EDF. This new protocol, called "Deadline Floor inheritance Protocol" (DFP), is simpler and more efficient than SRP while keeping all its good properties. In this paper we briefly describe both protocols and compare them from the complexity point of view. In light of its simplicity, we propose to change the language standard to include DFP instead of SRP. Some alternative modifications of the Ada Reference Manual are pointed out in order to include DFP in the most straightforward way.

1. Introduction

The Ada 2005 standard introduced "Earliest Deadline First" (EDF) as one of the supported dispatching policies and the "Stack Resource Policy" (SRP) was specified as the protocol for resource sharing among EDF tasks. SRP is a complex protocol as has been shown by the initial difficulties with its specification and implementation. Recently, Alan Burns has defined a new protocol for resource sharing in EDF [3]. This new protocol, called "Deadline Floor Protocol" (DFP), is simpler and more efficient than SRP while keeping all its good properties.

MaRTE OS [6] [7] is a real-time operating system developed by the Computers and Real-time group at the University of Cantabria. Most of its code is written in Ada with some C and assembler parts. It provides the applications with a POSIX/C interface to be used for concurrent C/C++ applications. From the Ada point of view, the POSIX version of the GNAT runtime system has been adapted to run on top of the POSIX/C interface of MaRTE OS. MaRTE OS provides support for most of the new real-time functionalities defined in the Ada 2012 standard; in particular it supports EDF dispatching (including SRP) [8].

In [9], we have implemented the DFP in the kernel of MaRTE OS and defined a POSIX-like interface to manage the specific parameters required for this protocol. We compared the performance of SRP and DFP and found that DFP outperforms SRP in both less implementation complexity and smaller runtime overheads.

In this paper we extend the performance analysis of DFP vs SRP to the Ada context. Then, we propose some changes that would be needed to incorporate the DFP into the Ada language, as a replacement for SRP. We analyze the required changes to the protected object locking policies and to the EDF task dispatching policy itself. We describe different alternatives for their discussion at the Real-Time Ada Workshop.

The paper is organized as follows. Section 2 briefly describes both resource sharing protocols: SRP and DFP. In Section 3 we give an overview of the EDF dispatching policy defined in Ada 2012 together with the specification of the SRP for protected objects. Section 4 describes the implementation of the new DFP in MaRTE OS, followed in Section 5 by a comparison of both resource sharing protocols in the context of the Ada specification. In Section 6 we propose some alternatives to incorporate the DFP to the Ada language, while Section 7 contains a discussion of the interactions between the new policy and the other policies defined in Ada. Finally, Section 8 gives our conclusions.

2. Resource sharing policies

For Ada tasks scheduled under fixed priorities, i.e., with the `Fifo_Within_Priorities` task dispatching policy, the `Ceiling_Locking` locking policy is used to manage access to shared resources through protected objects. When EDF was added in the Ada 2005 version of the language, tasks scheduled under the `EDF_Across_Priorities` task dispatching policy also used the `Ceiling_Locking` policy with a set of scheduling rules that made the scheduling of tasks using protected

objects equivalent to the Stack Resource Protocol (SRP). We will now review the SRP rules and introduce the Deadline Floor Protocol (DFP).

2.1. The SRP algorithm

Baker presents in [1] the Stack Resource Policy (SRP) for bounding priority inversion when accessing resources in real-time systems scheduled under EDF. SRP is a generalization of the Priority Inheritance Protocol (PIP) [2] and the Priority Ceiling Protocol (PCP) [2]. On a single processor, SRP presents the good properties of PCP: mutual exclusion ensured by the protocol itself (without needing an actual lock), deadlock avoidance, and at most a single blocking effect from any task with a longer relative deadline. These properties hold as long as there is no suspension while holding a lock.

With SRP, each task is assigned a number called the "preemption level" that correlates inversely to its relative deadline: the shorter the deadline the higher the preemption level. Shared resources are also assigned a preemption level that is the highest of the preemption levels of all the tasks that may use that resource.

The use of SRP imposes a new rule to the base EDF scheduling: "a thread can only become ready for execution if its preemption level is strictly higher than the preemption levels of the resources currently locked in the system". This rule is the cause of most of the SRP complexity as will be discussed later on.

2.2. Deadline Floor inheritance Protocol

Recently, Burns introduced a new protocol for resource sharing in EDF, called Deadline Floor Protocol (DFP) [3]. The DFP has all the key properties of SRP, specially causing at most a single blocking effect from any task with a longer relative deadline, which leads to the same worst-case blocking in both protocols. In an EDF-scheduled system, DFP is structurally equivalent to PCP in a system scheduled under fixed priorities.

Under DFP every resource has a relative deadline equal to the shortest relative deadline of any task that uses it. The relative deadline of a resource is called "deadline floor", making it clear the symmetry with the "priority ceiling" defined for the resources in PCP.

The key idea of the DFP is that the absolute deadline of a task could be temporarily shortened while accessing a resource. Given a task with absolute deadline `d` that accesses a resource with deadline floor `D` at time `t`, its absolute deadline is (potentially) reduced according to `d=min{d, t+D}` while holding the resource.

DFP does not add any new rule to the EDF scheduling, thus its leads to simpler and more efficient implementations than SRP as it will be shown later on.

3. Earliest Deadline First Dispatching and SRP in Ada 2012

Support for the Earliest Deadline First (EDF) dispatching is in the language from the Ada 2005 standard. The definition of EDF has been maintained unchanged in Ada 2012 (apart from the replacement of "pragmas" for the new Ada 2012 "aspects").

EDF dispatching is defined in the Ada Reference Manual (ARM) D.2.6 where a policy identifier (`EDF_Across_Priorities`) is defined along with the language-defined library package `Ada.Dispatching.EDF`. This package provides operations to manage the absolute deadlines of the tasks.

```
with Ada.Real_Time;
with Ada.Task_Identification;
package Ada.Dispatching.EDF is
   subtype Deadline is Ada.Real_Time.Time;
   Default_Deadline : constant Deadline := Ada.Real_Time.Time_Last;
   procedure Set_Deadline
             (D : in Deadline;
              T : in Ada.Task_Identification.Task_Id :=
                  Ada.Task_Identification.Current_Task);
   procedure Delay_Until_And_Set_Deadline (
             Delay_Until_Time : in Ada.Real_Time.Time;
             Deadline_Offset  : in Ada.Real_Time.Time_Span);
   function Get_Deadline (T : Ada.Task_Identification.Task_Id :=
                  Ada.Task_Identification.Current_Task) return Deadline;
end Ada.Dispatching.EDF;
```

The most complex part of the EDF dispatching definition in Ada is the integration of the base Ada dispatching model (based on fixed priorities for the tasks and priority ceilings for the protected objects) with the SRP rules and the "preemption level" concept. EDF is defined to work in a given band of priority levels, which may cover the whole range of system priorities, or a specific interval. The Reference Manual defines a clever integration of preemption levels and priorities: preemption levels of tasks and protected objects are mapped to priorities in the EDF priority band.

```
task T with Priority => 20;
-- if priority 20 is in an EDF range it represents the task's preemption level

protected Object with Priority => 24 is ...
-- if priority 24 is in an EDF range it represents the protected object's preemption level
```

The sources of priority inheritance are redefined for EDF tasks. The RM defines that, by default, the active priority of an EDF task is the lowest priority in its EDF priority band. The task will inherit priorities as any other Ada task, in particular, when an EDF task executes a protected operation it will inherit the priority (preemption level) of the protected object. But, for EDF tasks, the RM defines a third source of priority inheritance: "*the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks; and furthermore T has an earlier deadline than all other tasks on ready queues with priorities in the given EDF_Across_Priorities range that are strictly less than P*". This latter rule added to the base EDF scheduling is required to obey the SRP algorithm and is the major source of the complexity of this policy.

An example of the ordering of the EDF tasks is shown in Figure 1. Figure 1 (a) shows the schedule of three EDF tasks T_a, T_b and T_c with relative deadlines d_a=9, d_b=6 and d_c=3 and preemption levels pl_a=1, pl_b=2 and pl_c=3. There is a protected object PO_1 with preemption level pl_1=2 that is accessed by T_a at t=1.

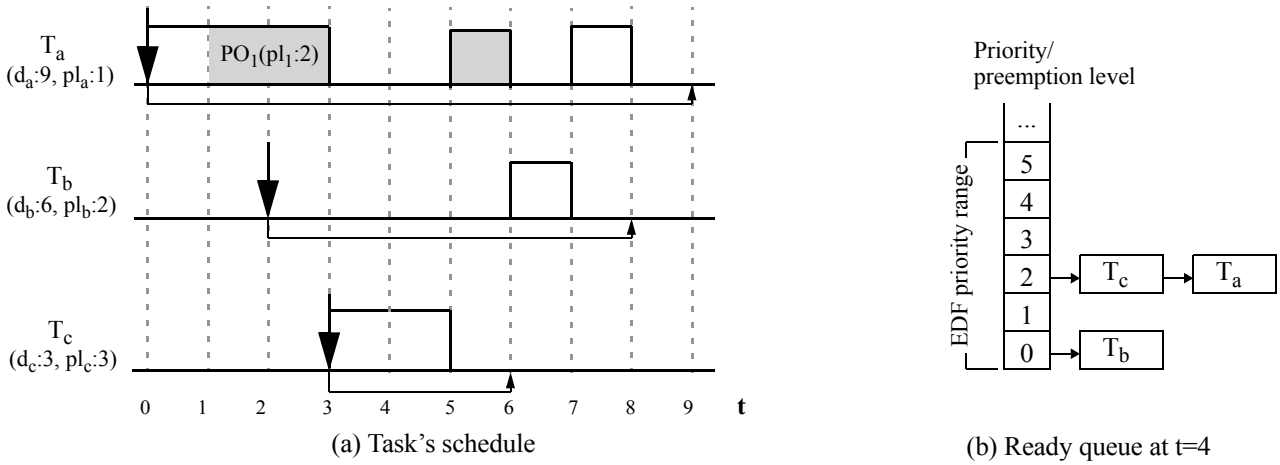


Figure 1. Example of EDF ordering in the ready queue

Figure 1 (b) shows the configuration of the ready queue at t=4. At that point in time T_a has inherited the preemption level of PO_1 and, consequently it is placed in the queue of priority 2. T_b is executing at the lowest priority in the range since it does not inherit T_a's priority because, although it has a more urgent deadline than T_a, its preemption level is not strictly higher than T_a's active priority as required by the third priority inheritance rule of SRP. T_c has inherited T_a's priority because it verifies the conditions imposed by the third priority inheritance rule of SRP: it has a shorter deadline that any task at priorities 2 and below, and its preemption level is strictly higher than T_a's priority.

The Ada dispatching model suggests a direct implementation of the ready queue as an array of queues, one for each priority level (Figure 1 (b)). Using this implementation of the ready queue and the rules for the EDF dispatching, the algorithm to find the right place for a newly activated EDF task T in the ready queue would be the one shown in Figure 2.

This section has shown the inherent complexity of the SRP. It is important to notice that this complexity is not due to the particular implementation of the SRP used in Ada (based on priority queues). Other alternative implementations, such as the one used in MaRTE OS that is based on only one queue [9], suffer from this complexity. Further evidence of this inherent

```
procedure Add_To_Ready_Queue (Task) is
begin
   Prio_Max := Lower_Priority_In_EDF_Range;
   for Prio in Lower_Priority_In_EDF_Range+1 .. T.Preempion_Level-1 loop
      if not Queue(Prio).Empty then
         if T.Deadline < Queue(Prio).Head.Deadline then
            Prio_Max := Prio;
         else
            exit;
         end if;
      end if;
   end loop;
   if Prio_Max = Lower_Priority_In_EDF_Range then
      Queue(Lower_Priority_In_EDF_Range).Add_In_Deadline_Order (T);
   else
      Queue(Prio_Max).Add_Head(T);
   end if;
end Add_To_Ready_Queue;
```

Figure 2. Algorithm to find the right place for a newly activated task in the ready queue with SRP

complexity of the SRP was the mistake in the original definition of SRP rules in Ada 2005 [4] and the original erroneous implementation in MaRTE OS [5].

There is one drawback that is specific to the Ada definition of SRP: the limited number of distinct preemption levels. The number of distinct preemption levels that can be used for tasks in an EDF priority range is the size of the range minus one. In a system with few priority levels or in a narrow EDF range this limitation could jeopardize the schedulability of the system by causing more blocking than is necessary. It could be argued that the implementation could provide more priority levels, but priority levels are expensive because they affect the size and performance of many of the run-time data structures such as the ready queues and the entry queues.

4. DFP implementation in MaRTE OS

Support for EDF and SRP was included in the kernel of MaRTE OS some years ago [8]. The operating system provides a POSIX-like interface to these services that is used by the adapted GNAT run-time system used by MaRTE OS to provide support to the EDF dispatching functionality defined in the Ada Reference Manual.

Recently, we have implemented the DFP in the MaRTE OS kernel. Implementing DFP is quite straightforward since it does not impose any additional scheduling rule to an EDF scheduled system, so the ordering criterion of the ready queue remains the same as in any other EDF system. Consequently, the ready queue in EDF & DFP is much simpler than the array of queues defined in Ada for EDF & SRP. It is enough with a single queue where tasks are ordered according to their absolute deadlines (shorter deadlines first).

The only specific parameter required by the DFP protocol is the deadline floor that, in the MaRTE OS implementation, has been added as a new mutex parameter (MaRTE uses POSIX mutexes as the base primitive for exclusive access to shared resources).

Our implementation of DFP has been simplified by adding the limitation to supporting just nested but not arbitrarily overlapped critical sections. This limitation has no effects for Ada, which does not allow overlapped critical sections. This simplification eases the implementation of the mutex lock and unlock operations. In the case of the lock operation, the old absolute deadline of the task is stored in a field of the mutex and the task is assigned a (possibly) new absolute deadline according to the deadline floor of the mutex. In the unlock operation the task's deadline is returned to the value stored during the lock operation and then the task has to be reordered in the ready queue according to its (possibly) longer deadline, which could lead to a context switch. Note that the possible deadline change in the lock operation cannot lead to a context switch since it could only shorten the deadline of the running task which, by definition, is the task with the earliest deadline in the system.

A small drawback of the DFP is the need for reading the clock in the lock operation. However, the time consumed by this operation is not an impediment for EDF & DFP to be more efficient than EDF & SRP as we will see in the next section.

Further details of the implementation of DFP in MaRTE OS can be found in [9].

5. Comparative analysis

The MaRTE OS implementation of SRP does not use an array of queues as the ARM suggests. Instead, it uses a single queue at the expense of adding complexity to the function used to order tasks in the queue. Of course the MaRTE OS implementation is functionally equivalent to the SRP algorithm defined by Ada.

The work in [9] presents a comparison of the DFP and SRP implementations in MaRTE OS. The results obtained show that the DFP implementation is simpler and much more efficient than the implementation of SRP. However, since the MaRTE OS implementation of SRP is different from the model described in the Ada standard, it could be argued that an implementation according to the Ada model could be more efficient than the one used in MaRTE. To analyze that model we have done an experiment creating the data structures used for both the multiple-queue implementation described by the ARM for EDF & SRP, and the single binary heap ready queue used for EDF & DFP in [9]. Such structures are shown in Figure 3.

(a) Ready queue for EDF & DFP

(b) Ready queue for EDF & SRP

Figure 3. Structure of the ready queue for EDF & DFP and EDF & SRP

The doubly-linked list is a linked data structure that consists of a set of sequentially linked nodes, each containing a ready task in this case. In addition, each node contains two links that are references to the previous and to the next node in the sequence of nodes. Removing and inserting the head of the list are operations that can be performed in constant time.

The binary heap [10] is a heap data structure created using a binary tree, with two additional constraints: all levels of the tree, except possibly the last one, i.e. the deepest one, are fully filled and each node is less than or equal to each of its children. The binary heap is an efficient implementation of an ordered queue: insert and remove operations take logarithmic time while peeking the head takes constant time.

Figure 3-b shows that in the implementation of the ready queue for EDF & SRP, the lower priority level of the EDF range can be implemented with a binary heap, because only the absolute deadlines are used to order the tasks at this priority level, which only has tasks that do not hold shared resources. For the other priority levels we use doubly-linked lists because elements are only added, removed or read at the head of the list.

Table 1 gives the pseudocode of the parts of the lock and unlock operations that are specific to the DFP and SRP implementations. Procedure `Reorder()` places the task in its correct position according to its new scheduling parameters. Procedure `Reorder_And_Dispatch()` calls `Reorder()` and, in the case the most urgent task had changed, it performs a context switch.

Let us consider the simpler case of the lock and unlock operations: a task, with no previous resources held, locks and immediately unlocks a mutex. Locking an SRP mutex involves a call to `Reorder()`, which dequeues the task from the head of the heap and adds it to the head of the doubly-linked list of the ceiling priority of the mutex. Unlocking the SRP mutex involves a call to `Reorder_And_Dispatch()` which removes the task from the head of the doubly-linked list and enqueues it into the binary heap. Note that now the task could possibly not be the most urgent task, because other tasks could have been activated in the meanwhile, implying the need for a context switch.

Table 1 Lock and unlock operations for DFP and SRP

	DFP	SRP

lock	`procedure Task_Locks_Mutex (Task, Mutex)`
	`is`
	`  ...`
	`  Mutex.Owner_Deadline := Task.Deadline;`
	`  Heir_Deadline := Clock +`
	`  Mutex.Deadlinefloor;`
	`  if Task.Deadline > Heir_Deadline then`
	`  Task.Deadline := Heir_Deadline;`
	`  end if;`
	`  ...`
	`end Task_Locks_Mutex;`
		`procedure Task_Locks_Mutex (Mutex)`
		`is`
		`  ...`
		`  Task.Num_Mutex_Owned ++;`
		`  Mutex.Owner_Preemption_Level :=`
		`  Task.Preemption_Level;`
		`  if Task.Preemption_Level <`
		`  Mutex.Preemption_Level`
		`  then`
		`  Task.Preemption_Level :=`
		`  Mutex.Preemption_Level;`
		`  Reorder (Task);`
		`  end if;`
		`  ...`
		`end Task_Locks_Mutex;`

unlock	`procedure Task_Unlocks_Mutex`
	`  (Task, Mutex)`
	`is`
	`  ...`
	`  Task.Deadline := Mutex.Owner_Deadline;`
	`  Reorder_and_Dispatch (Task);`
	`  ...`
	`end Task_Unlocks_Mutex;`
		`procedure Task_Unlocks_Mutex`
		`  (Task, Mutex)`
		`is`
		`  ...`
		`  Task.Num_Mutex_Owned --;`
		`  Task.Preemption_Level :=`
		`  Mutex.Owner_Preemption_Level;`
		`  Reorder_and_Dispatch (Task);`
		`  ...`
		`end Task_Unlocks_Mutex;`

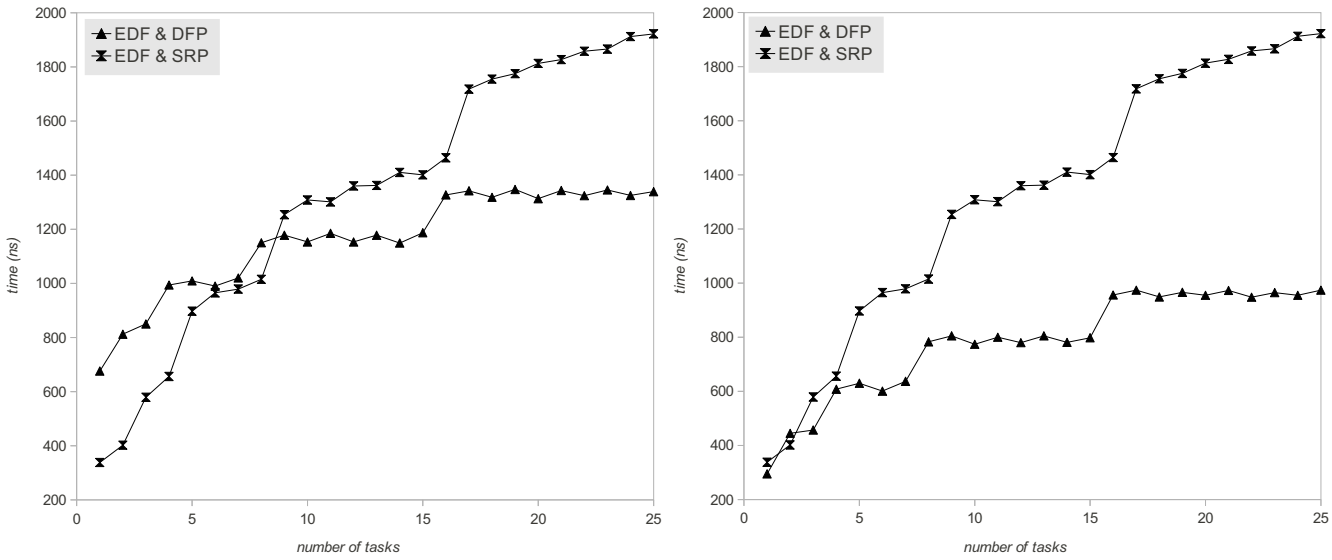
For a DFP mutex the sequence of calls is the following: locking the mutex requires reading the clock but no reordering since the task continues to be the most urgent one; unlocking the mutex involves a reordering of the task in the queue, since the end of the deadline inheritance could make the task less urgent than some other task activated while the mutex was being held.

To summarize, on the one hand, the higher overhead operations needed to lock and unlock an SRP mutex are the queue and dequeue operations in the binary heap. On the other hand, locking and unlocking a DFP mutex requires reading the clock and a requeue into the heap. In our tests we have found that reading the clock is faster than a requeue operation in a heap larger than just two elements. Figure 4 shows the execution times measured for a lock operation followed by an unlock as a function of the number of tasks under a worst-case situation, in a Pentium III at 800MHz. Figure 4a shows the performance comparison when using the heavier `Clock` function included in package `Ada.Realtime`, while Figure 4b shows the same comparison but using an internal MaRTE OS clock. This latter comparison is more accurate, because the internal implementation of the mutex operations would use internal OS resources. In consequence, the DFP outperforms the SRP even in the context of the Ada task dispatching model.

6. Alternatives to incorporate the DFP into the Ada Reference Manual

In order to include the DFP protocol in the Ada language the most decisive issue is how to deal with the current definition of the EDF dispatching. We can think of several alternatives:

- **a.** Replace the current definition of `EDF_Across_Priorities` (RM D.2.6) by a new (simpler) definition that uses DFP instead of SRP.
- **b.** Add a new dispatching policy (`EDF_With_Deadline_Floor`) to define the EDF & DFP dispatching and keep the current definition of `EDF_Across_Priorities` in the standard but declare it obsolescent.
- **c.** Add a new dispatching policy (`EDF_With_Deadline_Floor`) to define the EDF & DFP dispatching and keep the definition of `EDF_Across_Priorities` in the standard. An implementation could choose to implement both, one or none of these two dispatching policies.



a) Performance using Ada.Realtime.Clock

b) Performance using an internal OS clock

Figure 4. Performance comparison of DFP vs SRP

The proposed new dispatching policy identifier (`EDF_With_Deadline_Floor`) is intended to be used in the `Task_Dispatching_Policy` and the `Priority_Specific_Dispatching` pragmas:

```
pragma Task_Dispatching_Policy (EDF_With_Deadline_Floor);
pragma Priority_Specific_Dispatching (EDF_With_Deadline_Floor,
                                      first_priority, last_priority);
```

Pragma `Priority_Specific_Dispatching` specifies the task dispatching policy for the specified range of priorities. Since `EDF_With_Deadline_Floor` dispatching only requires one priority level it could be considered an error to provide a range with more than one priority levels. However, as discussed in Subection 7.1, for backwards compatibility of previous applications and also for analogy with the other dispatching policies, ranges of any size should be allowed. In this latter case the ARM should state that the active priority of the tasks in the `EDF_With_Deadline_Floor` band would be collapsed to the lower level of that band.

Another issue is related to the `Ceiling_Locking` policy. Locking policies are applied to the whole partition using the `Locking_Policy` pragma, so it could be argued that the identifier name `Ceiling_Locking` is inappropriate since its use for a partition implies that a protocol different from the "Ceiling Locking" is going to be used in `EDF_With_Deadline_Floor` priority ranges. On the other hand, it could be considered that for EDF scheduling the DFP is the equivalent to the "Ceiling Locking" concept and then the identifier would be appropriate.

More relevant than the identifier name are the deep modifications that would be required in the definition of the ceiling locking policy (RM D.3). Currently this policy is only defined in terms of priorities and, with the incorporation of the DFP, it would be necessary to define it also in terms of deadlines.

A new aspect is required to assign deadline floors to the protected objects:

```
protected Object with Deadline_Floor => Ada.Real_Time.Milliseconds(24) is ...
```

If a new aspect is not deemed necessary, it would be possible to reuse the `Relative_Deadline` aspect, much in the way the priority ceiling of the PCP protected objects is named `Priority` instead of `Priority_Ceiling`.

From Ada 95 the language allows the dynamic change of the base priority of a task with the operations in the `Ada.Dynamic_Priorities` package and in Ada 2005 the picture was completed with the definition of the `Priority` attribute that allows applications to dynamically change the ceiling priority of a protected object. In the same way, for reconfigurable systems that use the DFP protocol we would need primitives to change the relative deadline of tasks and protected

objects. In the case of the tasks, a new package `Ada.Dispatching.EDF.Dynamic_Relative_Deadlines` should be provided:

```
with Ada.Real_Time;
with Ada.Task_Identification;
package Ada.Dispatching.EDF.Dynamic_Relative_Deadlines is

   procedure Set_Relative_Deadline
      (D : in Real_Time.Time_Span;
       T : in Ada.Task_Identification.Task_Id :=
       Ada.Task_Identification.Current_Task);

   function Get_Relative_Deadline
      (T : Ada.Task_Identification.Task_Id :=
       Ada.Task_Identification.Current_Task)
      return Real_Time.Time_Span;

end Ada.Dispatching.EDF.Dynamic_Relative_Deadlines;
```

Currently the relative deadline is only used for setting the first absolute deadline after the task's activation, so it could be argued that this facility is not needed. However, the DFP specification should require that there are no violations of the deadline floor protocol by checking that tasks only use protected objects with a deadline floor that is not larger than the relative deadline of the task. If the deadline floors are allowed to change dynamically, so should the task relative deadlines.

To dynamically change the deadline floor of a protected object a new `Deadline` attribute should be provided. This new attribute would behave very much like the `Priority` attribute:

```
protected body PO is

   procedure Change_Relative_Deadline (D: in Real_Time.Time_Span) is
   begin
      ... -- PO'Deadline has old value here
      PO'Deadline := D;
      ... -- PO'Deadline has new value here
   end Change_Relative_Deadline; -- relative deadline is changed here
   ...
end PO;
```

Detecting ceiling violations could require a new check: when the absolute deadline of an EDF task is changed the implementation could check that the new absolute deadline is not sooner than the current time plus the relative deadline minus the real-time clock jitter.

Another check that might be necessary is detecting incoherent behaviors when changing the relative deadline. If the relative deadline is changed to a value that is greater than the interval from now until the absolute deadline there is an interval during which the absolute deadline is sooner than anticipated by the new relative deadline, which might imply glitches with some degree of priority inversion during the mode change. The main implication of this problem is that the protocol might not be able to ensure mutual exclusion just by itself, and an actual lock might be needed.

7. Integration of DFP in the current Ada dispatching model

Two main issues have to be solved related to the integration of the new `EDF_With_Deadline_Floor` dispatching with the other policies defined in the standard:

- Backwards compatibility: it is desirable that old EDF applications would need minor changes in their code (or no changes at all) to run with the new EDF & DFP dispatching policy.
- Interaction with the other dispatching policies: in applications with priority-specific dispatching policies it is usual that tasks with different policies interact using protected objects. Effects of this interaction should be well defined in the standard and should not lead to priority inversions nor deadlocks.

7.1. Backwards compatibility

The DFP rules allow an EDF application written for Ada 2005 to execute with no changes for the new EDF & DFP policy. The only difference would be the possible change of dispatching policy identifier in the configuration pragma `Task_Dispatching_Policy` or `Priority_Specific_Dispatching`. For this purpose, the new `EDF_With_Deadline_Floor` dispatching policy should be allowed to have priority bands with multiple levels, although the language would specify that those priorities would be collapsed to the lowest priority in the EDF band when calculating active priorities of tasks.

Assigning deadline floors to the protected objects would be desirable but not required since the default deadline for a protected object would be `Ada.Real_Time.Time_Span_Zero`, in the same way that the default ceiling priority for a protected object is `System.Priority'Last` (RM D.3 11/3). Of course assigning deadline floors to protected objects would be very important to improve system schedulability.

7.2. Interaction between `EDF_With_Deadline_Floor` and `FIFO_Within_Priorities` tasks

The first situation to consider is when the EDF range is at a higher priority level than the `FIFO_Within_Priorities` range. In such situation a FIFO task could use protected objects in the EDF range to interact with EDF tasks. The FIFO task would inherit both the priority ceiling and the deadline floor of the protected object.

Deadline floors of protected objects in the EDF priority band are assigned according to the deadlines of the EDF tasks that access them. To avoid a deadline floor violation when they are used by a FIFO task, it is enough to assume that the relative deadline of the FIFO tasks is infinite (`Ada.Real_Time.Time_Span_Last`). Consequently, a FIFO task accessing an EDF protected object would inherit its deadline floor and could only be preempted by EDF tasks with shorter relative deadlines. This behavior is the expected since, by definition, those tasks with shorter relative deadlines are not going to access the protected object.

In the opposite situation, when the EDF range is below the `FIFO_Within_Priorities` range, only the basic ceiling locking policy rules go into action: the EDF task that accesses a protected object in the FIFO range inherits the priority ceiling of the protected object and, consequently, can only be preempted by tasks with higher priorities.

7.3. Interaction between `EDF_With_Deadline_Floor` and `Round_Robin_Within_Priorities` tasks

It is exactly the same situation as for the `FIFO_Within_Priorities` policy since the round robin tasks behave like FIFO tasks while executing a protected action (the quantum does not expire until the protected action finishes).

7.4. Interaction between two `EDF_With_Deadline_Floor` tasks in different priority ranges

Although the utility of having more than one EDF priority range could be arguable, the language must be complete and should take into account that possibility.

It is the user's responsibility to avoid deadline floor violations when a task from the lower priority range uses a protected object in the higher priority range. The deadline floors assigned to the protected objects should consider the deadlines of all the tasks that access them in both priority ranges.

7.5. Interaction between `EDF_With_Deadline_Floor` and `EDF_Across_Priorities` tasks

In the assumption that both versions of EDF are allowed to coexist in the same system, synchronization among tasks in different EDF priority ranges would also work according to the expected properties of the protocols.

In the case that the `EDF_With_Deadline_Floor` is the higher priority range, the only issue to be taken into account is the correct assignment of deadline floors as discussed in Subection 7.4. If the `EDF_Across_Priorities` range is the higher priority range, no extra considerations are required since the ceiling locking policy is enough to ensure the correct behavior of the system.

8. Conclusions

The Deadline Floor inheritance Protocol (DFP) has been proposed as an alternative to the Stack Resource Protocol defined in Ada. In this paper we have shown that the DFP is simpler to understand, describe, and implement. DFP also has better performance than SRP. We have also discussed different alternatives for including the protocol in Ada. As a conclu-

sion we can state that the DFP is a good alternative to SRP in Ada and its standardization should be addressed in a future version of the language.

Acknowledgments

This work has been funded in part by the Spanish Government and FEDER funds under grant TIN2011-28567-C03-02 (HI-PARTES).

References

[1] Baker T.P., “Stack-Based Scheduling of Realtime Processes”, Journal of Real-Time Systems, Volume 3, Issue 1 (March 1991), pp. 67–99.

[2] L. Sha, R. Rajkumar, and J.P. Lehoczky. “Priority inheritance protocols: An approach to real-time synchronisation”. IEEE Transactions on Computers, 39(9):1175–1185, 1990.

[3] A. Burns. “A Deadline-Floor Inheritance Protocol for EDF Scheduled Real-Time Systems with Resource Sharing”. Technical Report YCS-2012-476, Department of Computer Science, University of York, UK, 2012.

[4] A. Zerzelidis, A. Burns, A. J. Wellings. “Correcting the EDF protocol in Ada 2005”. ACM Ada Letters – IRTAW '07: Proceedings of the 13th international workshop on Real-time Ada, 2007.

[5] M.L. Fairbairn and A. Burns. “Implementing and Verifying EDF Preemption-Level Resource Control”. LNCS 7308 – Proceedings Reliable Software Technology - Ada-Europe, Springer, 2012.

[6] M. Aldea and M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2001, Leuven, Belgium, Lecture Notes in Computer Science, LNCS 2043, May, 2001.

[7] MaRTE OS home page: http://marte.unican.es

[8] M. Aldea, M. González and J.F. Ruiz. “Implementation of the Ada 2005 Task Dispatching Model in MaRTE OS and GNAT”. LNCS 5570 – Proceedings of the Reliable Software Technologies - Ada-Europe, Springer, 2009.

[9] M. Gutiérrez López, A. Burns, M. Aldea Rivas, M. González Harbour. “Performance comparison between the SRP and DFP synchronization protocols in MaRTE OS”. Technical Report. University of Cantabria. Jan, 2013.
URI: http://hdl.handle.net/10902/1509

[10]Michael T. Goodrich, Roberto Tamassia. “Data Structures & Algorithms in Java”, John Wiley & Sons, Inc, 2006, pages 320-321.