Ada Issue 00357 Support for Deadlines and Earliest Deadline First Scheduling

!standard D.02.6 (01) 05-03-13 AI95-00357/09
!class amendment 03-09-27
!status Amendment 200Y 04-12-02
!status ARG Approved 10-0-0 04-11-20
!status work item 03-09-27
!status received 03-09-27
!priority Medium
!difficulty Medium
!subject Support for Deadlines and Earliest Deadline First Scheduling

!summary

Direct support for deadlines is defined and a new dispatching policy for Earliest Deadline First (EDF) Scheduling is supported.

!problem

Ada does not give any direct support for deadlines (although they are the most important notion in real-time systems). A standard way of representing deadlines via a predefined attribute of a task is needed.

Once deadlines are supported it is possible to define a dispatching policy for EDF scheduling. This being as common an approach as fixed priority scheduling for real-time system. It has the advantage that higher levels of resource utilization are possible.

For many scheduling schemes, including for example EDF, the most effective locking policy for protected objects is one known as the Stack Resource Policy (or preemption level locking). This was defined by Baker in 1991 [1] and has the advantage that it does not enforce the same policy on task dispatching and PO locking; but it does lead to effective implementation of POs. Moreover when priority dispatching is used the SRP policy is the same as ceiling locking.

!proposal

To give support for EDF scheduling with the necessary protocol for sharing protected objects requires the following:

1. a means of representing a task's (absolute) deadline
2. a means of assigning a preemption level to a task
3. a means of assigning a preemption level to a PO

The basic rule for preemption is that a new task T will preempt current running task S if:

1) Deadline of T is before that of S
2) Preemption level of T is higher than preemption level of any locked PO

If preemption levels are assigned (by the programmer) using relative deadlines for each task then a tight upper bound on blocking equivalent to that of priority ceiling inheritance is obtained.

In this proposal

a) deadlines are represented by a new task 'attribute'
b) preemption levels for tasks are represented by base priorities
c) preemption levels for POs are represented by ceiling priorities

The usual behavior of priorities, when tasks execute within a PO, are followed. But the active priority of a task when executing outside a PO may be lower than its base priority (see details below).

A new pragma is provided to allow a task to set a (non-default) relative deadline to control its activation:

pragma Relative_Deadline(expression);
where the expected type of expression is Real_Time.Time_Span.

The pragma can only occur in the specification part of a task.

To support EDF scheduling a new Priority_Policy identifier is defined: EDF_Across_Priorities.

When the Priority policy EDF_Across_Priorities is in effect the following rules apply. Let Base(T) be the base priority of task T and Deadline(T) its absolute deadline.

Rule 1
All ready queues in the specified range System.Any_Priority are ordered by deadline. For two tasks on the same ready queue, S and T: Deadline(S) < Deadline(T) implies S is closer to the head of the queue than T.

Rule 2
When a task T becomes unblocked it is placed on the highest priority ready queue R such that
  A protected object with ceiling priority R is currently locked,
  Deadline(T) < Deadline of the task locking this protected object, and
  Base(T) > Priority level of R.
if no such R exists then add T to Any_Priority'first.

Rule 3
When a task is chosen for execution it runs with the active priority determined by the ready queue from which the task was taken. If preempted it returns to the ready queue for its active priority. If it inherits a higher active priority it will return to its original active priority when it no longer inherits the higher level.

Rule 4
When a task executes a delay_statement that does not result in blocking it is added to the ready queue for its active priority.

Rule 5
Priority ceiling level Priority'first is not allowed.

!wording

D.2.6 Earliest Deadline First Dispatching

The deadline of a task is an indication of the urgency of the task; it represents a point on an ideal physical time line. Unless otherwise specified, whenever tasks compete for processors or other implementation-defined resources, the resources are allocated to the task with the earliest deadline.

This clause defines a package for representing a task's deadline and a dispatching policy that defines Earliest Deadline First (EDF) dispatching. A pragma is defined to assign an initial deadline to a task.

AARM Note:
This pragma is the only way of assigning an initial deadline to a task so that its activation can be controlled by EDF scheduling. This is similar to the way pragma Priority is used to give an initial priority to a task.

Syntax

The form of a pragma Relative_Deadline is as follows:

pragma Relative_Deadline (relative_deadline_expression);

Name Resolution Rules

The expected type for relative_deadline_expression is Real_Time.Time_Span.

Legality Rules

A Relative_Deadline pragma is allowed only immediately within a task_definition or the declarative_part of a subprogram_body. At most one such pragma shall appear within a given construct.

Static Semantics

The policy_identifier EDF_Across_Priorities is a task dispatching policy.

The following language-defined library package exists:

```
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 : in Ada.Task_Identification.Task_ID :=
            Ada.Task_Identification.Current_Task) return Deadline;
end Ada.Dispatching.EDF;
```

Post-Compilation Rules

If the EDF_Across_Priorities policy is specified for a partition, then the Ceiling_Locking policy (see D.3) shall also be specified for the partition.

If the EDF_Across_Priorities policy appears in a Priority_Specific_Dispatching pragma (see D.2.2) in a partition, then the Ceiling_Locking policy (see D.3) shall also be specified for the partition.
[Note: The above assumes AI-355 is included in the Amendment.]

AARM Reason: The priority model of EDF assumes that ceiling locking is used; it wouldn't make sense otherwise. Other scheduling policies are not so tightly bound to the locking policy - they still can make sense for an alternative policy.

Dynamic Semantics

A Relative_Deadline pragma has no effect if it occurs in the declarative_part of the subprogram_body of a subprogram other than the main subprogram.

The initial absolute deadline of a task containing pragma Relative_Deadline is the value of Real_Time.Clock + relative_deadline_expression, where the call of Real_Time.Clock is made between task creation and the start of its activation. If there is no Relative_Deadline pragma then the initial absolute deadline of a task is the value of Default_Deadline.
The environment task is also given an initial deadline by this rule.

The procedure Set_Deadline changes the absolute deadline of the task to D.
The function Get_Deadline returns the absolute deadline of the task.

The procedure Delay_Until_And_Set_Deadline delays the calling task until time Delay_Until_Time. When the task becomes runnable again it will have deadline Delay_Until_Time + Deadline_Offset.

On a system with a single processor, the setting of a task's deadline to the new value occurs immediately at the first point that is outside the execution of an abort-deferred operation. If the task is currently on a ready queue it is removed and re-entered on to the ready queue determined by the rules defined below.

When EDF_Across_Priorities is specified for priority range Low..High all ready queues in this range are ordered by deadline. The task at the head of a queue is the one with the earliest deadline.

A task dispatching point occurs for the currently running task T to which policy EDF_Across_Priorities applies whenever:

- o a change to the deadline of T occurs; or
- o a decrease to the deadline of any task on a ready queue for that processor occurs and the new deadline is earlier than that of the running task; or
- o there is a non-empty ready queue for that processor with a higher priority than the priority of the running task.

In these cases, the currently running task is said to be preempted and is returned to the ready queue for its active priority.

Whenever a task T to which policy EDF_Across_Priorities applies is added to a ready queue, other than when it is preempted, it is placed on the ready queue with the highest priority P, if one exists, such that:

- a task is executing within a protected object with ceiling priority P; and
- task T has an earlier deadline than any task executing within a protected object with ceiling priority P; and
- the base priority of task T is greater than P. If no such ready queue exists the task is added to the ready queue for the lowest priority in the range specified as EDF_Across_Priorities.

When the setting of the base priority of a task takes effect and the new priority is in the range specified as EDF_Across_Priorities, the task is added to the ready queue.

When a task is chosen for execution it runs with the active priority of the ready queue from which the task was taken. If it inherits a higher active priority it will return to its original active priority when it no longer inherits the higher level.

For all the operations defined in this package, Tasking_Error is raised if the task identified by T has terminated. Program_Error is raised if the value of T is Null_Task_ID.

Bounded (Run-Time) Error

If EDF_Across_Priorities is specified for priority range Low..High, it is a bounded error to declare a protected object with ceiling priority Low or to assign the value Low to attribute 'Priority. In either case either Program_Error is raised or the ceiling of the protected object is assigned the value Low+1.

Erroneous Execution

If a value of Task_ID is passed as a parameter to any of the subprograms of this package and the corresponding task object no longer exists, the execution of the program is erroneous.

Documentation Requirements

On a multiprocessor, the implementation shall document any conditions that cause the completion of the setting of a task's deadline to be delayed later than what is specified for a single processor.

Notes:
If two adjacent priority ranges, A..B and B+1..C are specified to have policy EDF_Across_Priorities then this is not equivalent to this policy being specified for the single range, A..C.

The above rules implement the preemption-level protocol (also called Stack Resource Policy protocol) for resource sharing under EDF dispatching. The preemption-level for a task is denoted by its base priority. The definition of a ceiling preemption-level for a protected object follows the existing rules for ceiling locking.

AARM Note:
An implementation may support additional dispatching policies by replacing absolute deadline with an alternative measure of urgency.

!discussion
The addition of procedure Delay_and_Set_Relative_Deadline reflects a common need to change deadline and then delay. For example a periodic task would do this. As the change of deadline (extending it) is likely to cause a context switch (with a later switch to put the task on the delay queue) it is more efficient for the run-time to do the delay and deadline change as a single operation.

Here we show that the above rules do give the required behavior for EDF scheduling and preemption level locking (without the need for a new locking policy, i.e. just use of ceiling locking).

First if no POs are used or locked then all tasks are always placed on the ready queue for level Priority' first (which is of course ordered by deadline).

The preemption level rule is that a newly executing task (T) should preempt the current running task (S) if it has a higher preemption level than any task that has locked a PO and a shorter deadline than S. There are a few cases to look at

If S has locked the PO then T will be placed in the same ready queue as S, it has a shorter deadline and hence will execute first.

If some other task has locked the PO then S must have preempted it and hence will again be on the ready queue for this level. T added to same queue and will preempt. [What the heck does this last sentence mean? Note that I fixed many more instances of 'q' to be 'queue'. - ED]

If there are a number of locked POs then T needs to be placed on the correct ready queue. On all ready queues, apart from that at Priority'first, the tail of the queue is the task that has the lock on the PO whose ceiling priority is that of the ready queue. Rule 2 then gets the right queue: the queue is appropriate as T has the correct attributes (shorter deadline than the task holding the lock, and higher preemption level). Moreover no higher priority queue is appropriate as one of the conditions will be false.

Note that in order to be placed on a ready queue, T must have a strictly higher preemption level (base priority) than the task on that queue that is holding the lock. Hence mutual exclusion is not broken by the preemption rules.
--
During discussions an alternative way of expressing this model was considered. This did not need the 'no POs at level Priority'first' rule. But it did require that tasks be placed at one above the level in the above model, and required the rule that a task preempted while executing in a PO must always stay at the front of its ready queue even if its deadline is later than the next task on the queue.

!corrigendum D.2.6(1)

@dinsc
The deadline of a task is an indication of the urgency of the task; it represents a point on an ideal physical time line. Unless otherwise specified, whenever tasks compete for processors or

other implementation-defined resources, the resources are allocated to the task with the earliest deadline.

This clause defines a package for representing a task's deadline and a dispatching policy that defines Earliest Deadline First (EDF) dispatching. A pragma is defined to assign an initial deadline to a task.

@i<@s8<Syntax>>

The form of a pragma Relative_Deadline is as follows:

@xindent<@b<pragma> Relative_Deadline (@i<relative_deadline_>@fa<expression>);>

@i<@s8<Name Resolution Rules>>

The expected type for @i<relative_deadline_>@fa<expression> is Real_Time.Time_Span.

@i<@s8<Legality Rules>>

A Relative_Deadline pragma is allowed only immediately within a @fa<task_definition> or the @fa<declarative_part> of a @fa<subprogram_body>. At most one such pragma shall appear within a given construct.

@i<@s8<Static Semantics>>

The @i<policy_>@fa<identifier> EDF_Across_Priorities is a task dispatching policy.

The following language-defined library package exists:

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

@i<@s8<Post-Compilation Rules>>

If the EDF_Across_Priorities policy is specified for a partition, then the Ceiling_Locking policy (see D.3) shall also be specified for the partition.

If the EDF_Across_Priorities policy appears in a Priority_Specific_Dispatching pragma (see D.2.2) in a partition, then the Ceiling_Locking policy (see D.3) shall also be specified for the partition.

@i<@s8<Dynamic Semantics>>

A Relative_Deadline pragma has no effect if it occurs in the declarative_part of the subprogram_body of a subprogram other than the main subprogram.

The initial absolute deadline of a task containing pragma Relative_Deadline is the value of Real_Time.Clock + @i<relative_deadline_>@fa<expression>, where the call of Real_Time.Clock is made between task creation and the start of its activation. If there is no Relative_Deadline pragma then the initial absolute deadline of a task is the value of Default_Deadline.
The environment task is also given an initial deadline by this rule.

The procedure Set_Deadline changes the absolute deadline of the task to D.
The function Get_Deadline returns the absolute deadline of the task.

The procedure Delay_Until_And_Set_Deadline delays the calling task until time Delay_Until_Time. When the task becomes runnable again it will have deadline Delay_Until_Time + Deadline_Offset.

On a system with a single processor, the setting of a task's deadline to the new value occurs immediately at the first point that is outside the execution of an abort-deferred operation. If the task is currently on a ready queue it is removed and re-entered on to the ready queue determined by the rules defined below.

When EDF_Across_Priorities is specified for priority range @i<Low>..@i<High> all ready queues in this range are ordered by deadline.
The task at the head of a queue is the one with the earliest deadline.

A task dispatching point occurs for the currently running task @i<T> to which policy EDF_Across_Priorities applies whenever:

@xbullet<a change to the deadline of @i<T> occurs; or>

@xbullet<a decrease to the deadline of any task on a ready queue for that processor occurs and the new deadline is earlier than that of the running task; or>

@xbullet<there is a non-empty ready queue for that processor with a higher priority than the priority of the running task.>

In these cases, the currently running task is said to be preempted and is returned to the ready queue for its active priority.

Whenever a task @i<T> to which policy EDF_Across_Priorities applies is added to a ready queue, other than when it is preempted, it is placed on the ready queue with the highest priority @i<P>, if one exists, such that:
@xbullet<a task is executing within a protected object with ceiling priority @i<P>; and>

@xbullet<task @i<T> has an earlier deadline than any task executing within a protected object with ceiling priority @i<P>; and>
@xbullet<the base priority of task @i<T> is greater than @i<P>.>
If no such ready queue exists the task is added to the ready queue for the lowest priority in the range specified as EDF_Across_Priorities.

When the setting of the base priority of a task takes effect and the new priority is in the range specified as EDF_Across_Priorities, the task is added to the ready queue.

When a task is chosen for execution it runs with the active priority of the ready queue from which the task was taken. If it inherits a higher active priority it will return to its original active priority when it no longer inherits the higher level.

For all the operations defined in this package, Tasking_Error is raised if the task identified by T has terminated. Program_Error is raised if the value of T is Null_Task_ID.

@i<@s8<Bounded (Run-Time) Errors>>

If EDF_Across_Priorities is specified for priority range @i<Low>..@i<High>, it is a bounded error to declare a protected object with ceiling priority
@i<Low> or to assign the value @i<Low> to attribute 'Priority. In either case either Program_Error is raised or the ceiling of the protected object is assigned the value @i<Low>+1.

@i<@s8<Erroneous Execution>>

If a value of Task_ID is passed as a parameter to any of the subprograms of this package and the corresponding task object no longer exists, the execution of the program is erroneous.

@i<@s8<Documentation Requirements>>

On a multiprocessor, the implementation shall document any conditions that cause the completion of the setting of a task's deadline to be delayed later than what is specified for a single processor.

@xindent<@s9<NOTES@hr
16 If two adjacent priority ranges, @i<A>..@i<B> and @i<B>+1..@i<C> are specified to have policy EDF_Across_Priorities then this is not equivalent to this policy being
specified for the single range, @i<A>..@i<C>.>>

@xindent<@s9<17 The above rules implement the preemption-level protocol (also called Stack Resource Policy protocol) for resource sharing under EDF dispatching. The preemption-level for a task is denoted by its base priority. The definition of a ceiling preemption-level for a protected object follows the existing rules for ceiling locking.>>

!ACATS test
Tests should be created to check on the implementation of this feature.

!appendix

Reference: [1] Baker, T.P., Stack-Based Scheduling of Real-Time Processes, Journal of Real-Time, Vol 3, No 1, pp67-99, March 1991.