A means of specifying priority specific dispatching is provided so that FIFO is not the only 'within_priorities' scheme supported. A Round_Robin_Within_Priorities dispatching policy is defined.

Although Ada defines a number of mechanisms for specifying task dispatching policies, only one, FIFO_Within_Priorities is guaranteed to be supported by all implementations of the Real-Time Systems Annex. Many applications have a mixture of real-time and non-real-time activities. The natural way of scheduling non-real-time activities is by time sharing the processor using round robin scheduling. Currently, the only way of achieving this is by incorporating yield (e.g. delay 0.0) operations in the code. This is ad hoc and intrusive.

This AI proposes a new scheduling policy which allows one or more priority levels to be identified as round robin priorities. A task whose base priority is set to one of these levels is scheduled in a round robin manner with a user-definable quantum.

The method proposed is a general one and will allow any priority level/band to have a specific scheme defined (e.g. FIFO, Round_Robin, EDF, etc.). This not only extends the facilities of Ada but also provides a well defined means of combining different dispatching schemes. This is a need that is increasingly identified in OS provisions and application surveys.
Section D.2 (Priority Scheduling) currently (as modified by AI-321) has 4 subsections:

D.2.1 The Dispatching Model
D.2.2 Pragma Task_Dispatching_Policy
D.2.3 Preemptive Dispatching
D.2.4 Non-Preemptive Dispatching

Two new specific policies are proposed (Round Robin in this AI and EDF in AI-357). In addition a means of specifying mixed scheduling is proposed in this AI.

This AI deals with modifying D.2.2 and D.2.3, and adding a new D.2.5 (it assumes EDF is also to be supported, in D.2.6).

In addition a new package is added to D.2.1 for parameters of dispatching policies.

Add before Dynamic Semantics in D.2.1 (as updated by AI-321).

Static Semantics

The following language-defined library package exists:

```ada
package Ada.Dispatching is
   pragma Pure(Dispatching);
   Dispatching_Policy_Error : exception;
end Ada.Dispatching;
```

Dispatching serves as the parent of other language-defined library units concerned with dispatching.

Modify D.2.2 to the following:

D.2.2 Task Dispatching Pragmas

This clause allows a single task dispatching policy to be defined for all priorities, or the range of priorities to be split into sub ranges that are assigned individual dispatching policies.

Syntax

The form of a pragma Task_Dispatching_Policy is as follows:

```ada
pragma Task_Dispatching_Policy(policy_identifier);
```

The form of a pragma Priority_Specific_Dispatching is as follows:

```ada
pragma Priority_Specific_Dispatching(policy_identifier, first_priority_expression, last_priority_expression);
```
Name Resolution Rules

The expected type for first_priority_expression and last_priority_expression is Integer.

Legality Rules

The policy_identifier used in a pragma Task_Dispatching_Policy shall be the name of a task
dispatching policy.

The policy_identifier used in a pragma Priority_Specific_Dispatching shall be the name of a task
dispatching policy.

Both first_priority_expression and last_priority_expression shall be static expressions in the
range of System.Any_Priority; the value of last_priority_expression shall be greater than or
equal to that of first_priority_expression.

Static Semantics

Pragma Task_Dispatching_Policy specifies the task dispatching policy.

Pragma Priority_Specific_Dispatching specifies the task dispatching policy for the specified
range of priorities. Tasks within the range of priorities specified in a
Priority_Specific_Dispatching pragma are dispatched according to the specified dispatching
policy.

If a partition contains one or more Priority_Specific_Dispatching pragmas the dispatching policy
for priorities not covered by any Priority_Specific_Dispatching pragmas is
FIFO_Within_Priorities.

Post-Compilation Rules

A Task_Dispatching_Policy pragma is a configuration pragma.
A Priority_Specific_Dispatching pragma is a configuration pragma.

The priority ranges specified in more than one Priority_Specific_Dispatching pragma within the
same partition shall not be overlapping.

If a partition contains one or more Priority_Specific_Dispatching pragmas it shall not contain a
Task_Dispatching_Policy pragma.

If a partition contains one or more Priority_Specific_Dispatching pragmas then the
Ceiling_Locking policy (see D.3) shall also be specified for that partition.

Dynamic Semantics

A task dispatching policy specifies the details of task dispatching that are not covered by the
basic task dispatching model. These rules govern when tasks are inserted into and deleted from
the ready queues. A single task dispatching policy is specified by a Task_Dispatching_Policy
pragma. Pragma Priority_Specific_Dispatching assigns distinct dispatching policies to ranges of
System.Any_Priority. If neither pragma applies to any of the program units comprising a
partition, the task dispatching policy for that partition is unspecified.
If a partition contains one or more Priority_Specific_Dispatching pragmas a task dispatching point occurs for the currently running task of a processor whenever there is a non-empty ready queue for that processor with a higher priority than the priority of the running task.

A task that has its base priority changed may move from one dispatching policy to another. It is immediately dispatched according to the new policy.

Implementation Permissions

Implementations are allowed to define other task dispatching policies, but need not support more than one task dispatching policy per partition.

An implementation need not support pragma Priority_Specific_Dispatching if it is infeasible to support it in the target environment.

Add to D.2.3 The Standard Task Dispatching Policy

Static Semantics

The policy_identifier FIFO_Within_Priorities is a task dispatching policy.

Add to D.2.4 Non-Preemptive Dispatching - Legality Rules (see AI-298):

Non_Preemptive_FIFO_Within_Priorities shall not be specified as the policy_identifier of pragma Priority_Specific_Dispatching (see D.2.2).

Add a new section:

D.2.5 Round Robin Dispatching

This clause defines the task dispatching policy Round_Robin_Within_Priorities and the package Round_Robin.

Static Semantics

The policy_identifier Round_Robin_Within_Priorities is a task dispatching policy.

The following language-defined library package exists:

```ada
with System;
with Ada.Real_Time;
package Ada.Dispatching.Round_Robin is
  Default_Quantum : constant Ada.Real_Time.Time_Span := <implementation-defined>;
  procedure Set_Quantum(Pri : in System.Priority;
                         Quantum : in Ada.Real_Time.Time_Span);
  procedure Set_Quantum(Low,High : in System.Priority;
                         Quantum : in Ada.Real_Time.Time_Span);
end Ada.Dispatching.Round_Robin;
```

function Actual_Quantum
    (Pri : System.Priority) return Ada.Real_Time.Time_Span;
function Is_Round_Robin (Pri : System.Priority) return Boolean;
end Ada.Dispatching.Round_Robin;

When task dispatching policy Round_Robin_Within_Priorities is the single policy in effect for a
partition, each task with priority in the range of System.Interrupt_Priority is dispatched according
to policy FIFO_Within_Priorities.

Dynamic Semantics

The procedures Set_Quantum set the required Quantum value for a single level Pri or a range
of levels Low .. High. If no quantum is set for a Round Robin priority level, Default_Quantum is
used.

The function Actual_Quantum returns the actual quantum used by the implementation for the
priority level Pri.

The function Is_Round_Robin returns True if priority Pri is covered by task dispatching policy
Round_Robin_Within_Priorities; otherwise it returns False.

A call of Actual_Quantum or Set_Quantum raises exception
Ada.Dispatching.Dispatching_Policy_Error if a predefined policy other than
Round_Robin_Within_Priorities applies to the specified priority.

For Round_Robin_Within_Priorities, the dispatching rules for FIFO_Within_Priorities apply with
the following additional rules:

o When a task is added or moved to the tail of the ready queue for its base priority, it has an
execution time budget equal to the quantum for that priority level. This will also occur when a
blocked task becomes executable again.

o When a task is preempted (by a higher priority task) and is added to the head of the ready
queue for its priority level, it retains its remaining budget.

o While a task is executing, its budget is decreased by the amount of execution time it uses.
The accuracy of this accounting is the same as that for execution time clocks (see D.14).

o A task that has its base priority set to a Round Robin priority is moved to the tail of the ready
queue for its new priority level.

AARM Note: It will be given a budget as described in the first bullet.

o When a task has exhausted its budget and is without an inherited priority (and is not executing
within a protected operation), it is moved to the tail of the ready queue for its priority level. This
is a task dispatching point.

AARM Note: It will be given a budget as described in the first bullet.

Documentation Requirements
An implementation shall document the quantum values supported.

An implementation shall document the accuracy with which it detects the exhaustion of the budget of a task.

Notes

Due to implementation constraints, the quantum value returned by Actual_Quantum might not be identical to the value set by Set_Quantum. However, if no value is set by Set_Quantum, then the value returned by Actual_Quantum will be identical to that of Default_Quantum.

A task that executes continuously with an inherited priority will not be subject to round robin dispatching.

In D.4(7), change "appears in" to "applies to".

!example

To specify round robin dispatching for the lowest priority in a 32-priority system:

`pragma Priority_Specific_Dispatching (FIFO_Within_Priorities, 2, 32);`
`pragma Priority_Specific_Dispatching (Round_Robin_Within_Priorities, 1, 1);`

!discussion

For the Round Robin proposal: the rule concerning budget exhaustion gives the important details of the proposal. First it is the base priority of a task that is significant. If a task's base priority is at a round robin level then it will consume its budget whenever it is executing even when it has inherited a higher priority (i.e. its active priority is greater than its base priority). The final rule also deals with the key question of what happens if the budget becomes exhausted while executing in a protected object. To ensure mutual exclusion, without requiring a further lock, it is necessary to allow the task to keep executing within the PO. It will consume more than its quantum but the expected behavior of the system is maintained. The usual programming discipline of keeping the code within protected objects as short as possible will ensure that quantum overrun is minimized. Further support for these semantics comes from observing that execution within a PO is abort-deferred. Quantum exhaustion is a less severe state than being aborted; deferred behavior thus seems appropriate.


The proposal is easily implemented on top of the POSIX provision for Round Robin scheduling. Indeed it has been implemented in this way by Michael Gonzalez Harbour. He reports no difficulty with the implementation.
This wording also fixes a wording glitch; pragmas Task_Dispatching_Policy and Queuing_Policy are configuration pragmas, and thus never appear inside of units; but the last sentence of D.2.2(6) and and D.4(7) implies that they do. The Corrigendum fixed this for D.3(6), but not the other two cases.

!corrigendum D.2.1(01)

@dinsa
The task dispatching model specifies preemptive scheduling, based on conceptual priority-ordered ready queues.
@dinss
@i<@s8<Static Semantics>>

The following language-defined library package exists:

@xcode<@b<package> Ada.Dispatching @b<is>
    @b<pragma> Pure(Dispatching);
    Dispatching_Policy_Error : @b<exception>;
@b@end> Ada.Dispatching;
>

Dispatching serves as the parent of other language-defined library units concerned with dispatching.

!corrigendum D.2.2(00)

@drepl
The Standard Task Dispatching Policy
@dby
Task Dispatching Pragmas

!corrigendum D.2.2(01)

@dinsb
@i<@s8<Syntax>>

The form of a @fa<pragma> Task_Dispatching_Policy is as follows:
@dinst
This clause allows a single task dispatching policy to be defined for all priorities, or the range of priorities to be split into sub ranges that are assigned individual dispatching policies.

!corrigendum D.2.2(02)

@dinsa
@fa<@b<pragma> Task_Dispatching_Policy (@i<policy_>identifier);>
@dinss
The form of a @fa<pragma> Priority_Specific_Dispatching is as follows:

@fa<@b<pragma> Priority_Specific_Dispatching (@i<policy_>identifier, @i<first_priority_>expression, @i<last_priority_>expression);>
The expected type for \texttt{first\_priority\_} and \texttt{last\_priority\_} is Integer.

\textbf{corrigendum D.2.2(03)}

\texttt{policy\_identifier} shall either be FIFO\_Within\_Priorities or an implementation-defined \texttt{identifier}.

\texttt{policy\_identifier} used in a \texttt{pragma} Task\_Dispatching\_Policy shall be the name of a task dispatching policy.

The \texttt{policy\_identifier} policy\_identifier used in a \texttt{pragma} Priority\_Specific\_Dispatching shall be the name of a task dispatching policy.

Both \texttt{first\_priority\_} and \texttt{last\_priority\_} shall be static expressions in the range of System.Any\_Priority; the value of \texttt{last\_priority\_} shall be greater than or equal to \texttt{first\_priority\_}.

\textbf{corrigendum D.2.2(04)}

A \texttt{Task\_Dispatching\_Policy} pragma is a configuration pragma.

The priority ranges specified in more than one Priority\_Specific\_Dispatching pragma within the same partition shall not be overlapping.

If a partition contains one or more Priority\_Specific\_Dispatching pragmas then the Ceiling\_Locking policy (see D.3) shall also be specified for that partition.
A task dispatching policy specifies the details of task dispatching that are not covered by the basic task dispatching model. These rules govern when tasks are inserted into and deleted from the ready queues, and whether a task is inserted at the head or the tail of the queue for its active priority. The task dispatching policy is specified by a Task_Dispatching_Policy configuration pragma. If no such pragma appears in any of the program units comprising a partition, the task dispatching policy for that partition is unspecified.

If neither pragma applies to any of the program units comprising a partition, the task dispatching policy for that partition is unspecified.

If a partition contains one or more Priority_Specific_Dispatching pragmas a task dispatching point occurs for the currently running task of a processor whenever there is a non-empty ready queue for that processor with a higher priority than the priority of the running task.

A task that has its base priority changed may move from one dispatching policy to another. It is immediately dispatched according to the new policy.

Implementations are allowed to define other task dispatching policies, but need not support more than one such policy per partition.

For optimization purposes, an implementation may alter the points at which task dispatching occurs, in an implementation defined manner. However, a delay_statement always corresponds to at least one task dispatching point.
The \texttt{FIFO\_Within\_Priorities} is a task dispatching policy.

\textit{corrigendum D.2.4(01)}

This is a fake to trigger conflict processing. The real change is
\textit{corrigendum D.2.5(1)}

Non\_Preemptive\_FIFO\_Within\_Priorities shall not be specified as the \texttt{policy\_} of \texttt{pragma Priority\_Specific\_Dispatching} (see D.2.2).

The \texttt{Round\_Robin\_Within\_Priorities} is a task dispatching policy.

The following language-defined library package exists:

\begin{verbatim}
with System; 
with Ada.Real_Time;
package Ada.Dispatching.Round_Robin is
  Default_Quantum : constant Ada.Real_Time.Time_Span := 
    @ft<i>implementation-defined>@ft;i;
  procedure Set_Quantum (Pri : in System.Priority; 
                        Quantum : in Ada.Real_Time.Time_Span);
  procedure Set_Quantum (Low, High : in System.Priority; 
                        Quantum   : in Ada.Real_Time.Time_Span);
  function Actual_Quantum (Pri : System.Priority) return Ada.Real_Time.Time_Span;
  function Is_Round_Robin (Pri : System.Priority) return Boolean;
end Ada.Dispatching.Round_Robin;
\end{verbatim}

When task dispatching policy \texttt{Round\_Robin\_Within\_Priorities} is the single policy in effect for a partition, each task with priority in the range of \texttt{System.Interrupt\_Priority} is dispatched according to policy \texttt{FIFO\_Within\_Priorities}.

The procedures Set\_Quantum set the required Quantum value for a single level.
Pri or a range of levels Low .. High. If no quantum is set for a Round Robin priority level, Default_Quantum is used.

The function Actual_Quantum returns the actual quantum used by the implementation for the priority level Pri.

The function Is_Round_Robin returns True if priority Pri is covered by task dispatching policy Round_Robin_Within_Priorities; otherwise it returns False.

A call of Actual_Quantum or Set_Quantum raises exception Ada.Dispatching.Dispatching_Policy_Error if a predefined policy other than Round_Robin_Within_Priorities applies to the specified priority.

For Round_Robin_Within_Priorities, the dispatching rules for FIFO_Within_Priorities apply with the following additional rules:

- When a task is added or moved to the tail of the ready queue for its base priority, it has an execution time budget equal to the quantum for that priority level. This will also occur when a blocked task becomes executable again.

- When a task is preempted (by a higher priority task) and is added to the head of the ready queue for its priority level, it retains its remaining budget.

- While a task is executing, its budget is decreased by the amount of execution time it uses. The accuracy of this accounting is the same as that for execution time clocks (see D.14).

- A task that has its base priority set to a Round Robin priority is moved to the tail of the ready queue for its new priority level.

- When a task has exhausted its budget and is without an inherited priority (and is not executing within a protected operation), it is moved to the tail of the ready queue for its priority level. This is a task dispatching point.

Documentation Requirements

An implementation shall document the quantum values supported.

An implementation shall document the accuracy with which it detects the exhaustion of the budget of a task.

Due to implementation constraints, the quantum value returned by Actual_Quantum might not be identical to that set with Set_Quantum.

A task that executes continuously with an inherited priority will not be subject to round robin dispatching.
Two queuing policies, FIFO_Queuing and Priority_Queuing, are language defined. If no Queuing_Policy pragma appears in any of the program units comprising the partition, the queuing policy for that partition is FIFO_Queuing. The rules for this policy are specified in 9.5.3 and 9.7.1.

@dby
Two queuing policies, FIFO_Queuing and Priority_Queuing, are language defined. If no Queuing_Policy pragma applies to any of the program units comprising the partition, the queuing policy for that partition is FIFO_Queuing. The rules for this policy are specified in 9.5.3 and 9.7.1.

!ACATS test

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