AI-003035 New Pragma and additional Restriction Identifiers for Real-time Systems

!standard D.7 (10) 05-03-18 AI95-00305/09

!standard D.7 (15)
!standard D.7 (19)
!standard H.5 (01)
!class amendment 02-07-17
!status Amendment 200Y 02-10-23
!status WG9 Approved 02-12-13
!status ARG Approved 10-0-1 02-10-11
!status work item 02-07-17
!status received 02-07-17
!priority High
!difficulty Medium
!subject: New pragma and additional restriction identifiers for real-time systems

!summary

A new pragma and addition restriction identifiers are defined to enhance the ability to create highly efficient and predictable tasking runtime systems.

!problem

Experience constructing the "highly efficient tasking runtime systems" of D.7 has shown that the set of restrictions is insufficient. In particular, the Ravenscar Profile is commonly used in Safety-Critical and High-Integrity applications to provide a highly efficient tasking runtime. However, the profile requires restrictions beyond those defined by the Standard, requiring users to fall back on vendor-defined extensions.

!proposal

This amendment introduces several new restriction identifiers to define runtime behaviors that are to be restricted when using the Ravenscar profile. These identifiers may be used to specify runtime behavior which is independent of the Ravenscar definition.

A new pragma is also defined to force an implementation to detect blocking within a protected operation.

!wording

The following new static restriction_identifiers are defined and inserted after D.7 (10):

No_Calendar
There are no semantic dependencies on package Calendar.

No_Dynamic_Attachment
There is no call to any of the operations defined in package Interrupts (Is_Reserved, Is_Attached, Current_Handler, Attach_Handler, Exchange_Handler, Detach_Handler, and Reference).

No_Local_Protected_Objects
Protected objects shall be declared only at library level.

No_Protected_Type_Allocators
There are no allocators for protected types or types containing

protected type subcomponents.

No_Relative_Delay
There are no delay_relative_statements.

No_Requeue_Statements
There are no requeue_statements.

No_Select_Statements
There are no select_statements.

No_Task_Attributes_Package
There are no semantic dependencies on package Task_Attributes.

Simple_Barriers
The Boolean expression in an entry barrier shall be either a static Boolean expression or a Boolean component of the enclosing protected object.

The following new dynamic restriction_identifier is defined and replaces D.7 (15/1):

No_Task_Termination
All tasks are non-terminating. It is implementation-defined what happens if a task attempts to terminate.

The following new dynamic restriction_parameter_identifier is defined and inserted after D.7 (19/1):

Max_Entry_Queue_Length
Max_Entry_Queue_Length defines the maximum number of calls that are queued on an entry. Violation of this restriction results in the raising of Program_Error at the point of the call.

The following pragma is defined in a new subsection H.5:

H.5 Pragma Detect_Blocking

The following pragma forces an implementation to detect potentially blocking operations within a protected operation.

Syntax
The form of a pragma Detect_Blocking is as follows:
pragma Detect_Blocking;

Post-Compilation Rules
A pragma Detect_Blocking is a configuration pragma.

Dynamic Semantics
An implementation is required to detect a potentially blocking operation within a protected operation [, and to raise Program_Error (see 9.5.1)].

Implementation Permissions
An implementation is allowed to reject a compilation_unit if a potentially blocking operation is present directly within an entry_body or the body of a protected subprogram.

Notes
An operation that causes a task to be blocked within a foreign language domain is not defined to be potentially blocking, and need not be detected.

!discussion

The restriction No_Task_Hierarchy must impose the constraint that all tasks depend directly on the environment task as a result of all task objects being created by library level declarations. The restriction means that no support is needed for "masters" and "waiting for dependent tasks". This also matches the semantics for restriction No_Local_Protected_Objects.

No_Protected_Type_Allocators is similar to the existing restriction
No_Task_Allocators.

No_Delay in H.4 is too strong for the Ravenscar Profile since we want to allow delay_until Ada.Real_Time.Time, but not relative delay (non-deterministic) nor package Ada.Calendar (too coarse).

No_Select_Statements excludes selective_accept, timed and conditional (protected) entry calls, and asynchronous_select.

Some restriction identifiers concerning tasking are actually defined in H.4. For reasons of minimum change we did not move them.

When pragma Detect_Blocking is in force, we allow implementations to reject protected bodies that contain potentially blocking operations. Such a static check prevents problems from appearing in fielded systems from a potentially blocking operation which is rarely executed. We limit the check to protected bodies so that libraries which contain potentially blocking operations (such as a lock) which cannot be executed do not cause the program to be rejected.

!ACATS test

ACATS tests should be constructed for these features.

!corrigendum D.7(10)

@dinsa
@xhang<@xterm<No_Asynchronous_Control>
There are no semantic dependences on the package Asynchronous_Task_Control.>
@dinss
@xhang<@xterm<No_Calendar>
There are no semantic dependencies on package Calendar.>

@xhang<@xterm<No_Dynamic_Attachment>
There is no call to any of the operations defined in package
Interrupts (Is_Reserved, Is_Attached, Current_Handler,
Attach_Handler, Exchange_Handler, Detach_Handler, and Reference).>

@xhang<@xterm<No_Local_Protected_Objects>
Protected objects shall be declared only at library level.>

@xhang<@xterm<No_Protected_Type_Allocators>
There are no @fa<allocator>s for protected types or types containing
protected type subcomponents.>

@xhang<@xterm<No_Relative_Delay>
There are no @fa<delay_relative_statement>s.>

@xhang<@xterm<No_Requeue_Statements>
There are no @fa<requeue_statement>s.>

@xhang<@xterm<No_Select_Statements>
There are no @fa<select_statement>s.>

@xhang<@xterm<No_Task_Attributes_Package>
There are no semantic dependencies on package Task_Attributes.>

@xhang<@xterm<Simple_Barriers>
The Boolean expression in an entry barrier shall be either a static Boolean expression or a Boolean component of the enclosing protected object.>

!corrigendum D.7(15)

@drepl
@i<This paragraph was deleted>
@dby
The following @I<restriction_>@fa<identifier>s are language defined:

@xhang<@xterm<No_Task_Termination>
All tasks are non-terminating. It is implementation-defined what happens if a task attempts to terminate.>

!corrigendum D.7 (19)

@dinsa
@xhang<@xterm<Max_Tasks>
Specifies the maximum number of task creations that may be executed over the lifetime of a partition, not counting the creation of the environment task. A value of zero prevents any task creation and, if a program contains a task creation, it is illegal. If an implementation chooses to detect a violation of this restriction, Storage_Error should be raised; otherwise, the behavior is implementation defined.>
@dinst
@xhang<@xterm<Max_Entry_Queue_Length>
Max_Entry_Queue_Length defines the maximum number of calls that are queued on an entry. Violation of this restriction results in the raising of Program_Error at the point of the call.>

!corrigendum H.5 (1)

@dinsc

The following @fa<pragma> forces an implementation to detect potentially blocking operations within a protected operation.

@i<@s8<Syntax>>

The form of a @fa<pragma> Detect_Blocking is as follows:
@xindent<@b<pragma> Detect_Blocking;>

@i<@s8<Dynamic Semantics>>

An implementation is required to detect a potentially blocking operation within a protected operation, and to raise Program_Error (see 9.5.1).

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

A pragma Detect_Blocking is a configuration pragma.

@i<@s8<Implementation Permissions>>

An implementation is allowed to reject a @fa<compilation_unit> if a potentially blocking operation is present directly within an @fa<entry_body> or the body of a protected subprogram.

@xindent<@s9<NOTES@hr
10 An operation that causes a task to be blocked within a foreign language domain is not defined to be potentially blocking, and need not be detected.>>

!appendix

Editor's Note: This AI was split out of the Ravenscar AI, AI-249.

****

Editor's Note:

The rule that pragma Detect_Blocking is a configuration pragma was moved to Post-Compilation Rules and rewritten to be consistent with other configuration pragmas, including pragma Profile.