Ada Issue 10188 Setting a Task Base Priority is Immediate

!standard D.5 (10) 04-12-02 AI95-100188-02/02
!standard D.5 (12)
!standard D.4 (15)
!class binding interpretation 04-11-08
!status Amendment 200Y 04-12-02
!status ARG Approved 8-0-2 04-11-20
!status work item 04-11-08
!status received 04-11-08
!priority Medium
!difficulty Medium
!subject Setting a task base priority is immediate

!summary

Setting a task's base priority becomes immediate, but the reordering of entry queues is allowed to be deferred.

!question

Must the setting of a task base priority happen "immediately" on a mono-processor, in analogy with the requirements for preemptive abort? (Yes.)

!recommendation

Rather than allow an implementation to change the priority of a task "as soon as practical", the change is required to be immediate, but the reordering of entry queues is allowed to be deferred. This deals with the problem of entry queues and layered run-time kernels, but requires an immediate change if the task is on a ready queue or the delay queue.

!wording

Replace D.5(10):

> On a system with a single processor, the setting of a task's base priority to the new value occurs immediately at the first point that is outside the execution of an abort-deferred operation.

Add after D.5(12):

Documentation Requirements

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

Add after D.4(15):

Implementations are allowed to defer the reordering of entry queues following a change of base priority of a task blocked on the entry call if it is not practical to reorder the queue immediately.

AARM Note:

The reordering should occur as soon as the blocked task can itself perform the reinsertion into the entry queue.

!discussion

The questioner was expecting a response along the lines of:

There seems no reason that the timing requirements for changing a task's priority and aborting a task on a single processor should differ significantly. Hence, the term "immediately" should be used in the definition of when a task's base priority is changed on a single processor, once the task is outside a protected action.

In particular, there seems no justification to allow real-time implementations to defer the change of base priority until the task wakes up on its own at its old priority, because the purpose of raising its priority may very well be to have the target task "immediately" preempt some intermediate priority task.

However, there is some justification for the difference. The difference is that "completing" a construct is not the same as "exiting" the construct, and priority change is closer to exiting, since it has a big nasty side-effect -- namely, removal and re-insertion of queued entry calls -- that may require waiting for the caller-task to execute. This makes if difficult to implement on a multi-threaded (multi-lock) run-time kernel, or on a layered (non-Ada) kernel.

From an implementation point of view, we can have some other task mark a given task as having "completed" a construct, but to actually execute the finalization code and leave the construct we need to wait for the affected task to run.

By analogy, here we can change the priority of a task, but if we want the task's entry calls to be repositioned, we have to wait until the task can be scheduled.

The abort statement is defined in the core language and the IRTAW was concerned that a 'valid' implementation could postpone the effect of abort indefinitely - hence the use of "immediate" in the Real-Time Annex. By contrast, the dynamic priorities package is a Annex D facility and hence it was be assumed that the implementation does the right thing, and that "as soon as practical" is strong enough. However there is evidence that implementations not only postpone the change of priority for a task on an entry queue but also for those on ready queues and the delay queue.
This does undermine the fundamental purpose of changing another task's priority; and hence some tightening of the language definition is justified.

!corrigendum D.4(15)

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

@dinst
Implementations are allowed to defer the reordering of entry queues following a change of base priority of a task blocked on the entry call if it is not practical to reorder the queue immediately.

!corrigendum D.5(10)

@drepl
Setting the task's base priority to the new value takes place as soon as is practical but not while the task is performing a protected action. This setting occurs no later then the next abort completion point of the task T (see 9.8).
@dby
On a system with a single processor, the setting of a task's base priority to the new value occurs immediately at the first point that is outside the execution of an abort-deferred operation.

!corrigendum D.5(12)

@dinsa
If any subprogram in this package is called with a parameter T that specifies a task object that no longer exists, the execution of the program is erroneous.
@dinst
@i<@s8<Documentation Requirements>>

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

!ACATS Test

We need tests like the ones in CXD6001, CXD6002, and CXD6003, which purport to test immediate abort.

!appendix