Session: New Scheduling / Dispatching Policies

Rapporteur: Lars Asplund
Uppsala University
Department of Information Technology
P.O. Box 311, S-751 05 Uppsala, Sweden
Lars.Asplund@it.uu.se

The session, named *New Scheduling and Dispatching Policies*, was chaired by Juan de la Puente and covered a couple of new scheduling policies. A Non-Preemptive dispatching policy was proposed by Alan Burns. A suggestion to have per-task dispatching policies was presented by Michael González Harbour, where the per-task policies could be one of FIFO_Within_Priorities, Round_Robin_Within_Priorities, or Sporadic_Server. Michael also proposed adding CPU-time budgeting capabilities to the language.

1. Non-Preemptive Dispatching

One motivation for a Non-Preemptive dispatching is that, in the migration path from cyclic executives, non-preemptive is a step on the way to full preemptive scheduling.

An additional execution resource (not only the CPU) is created: the execution token. A ready task must acquire the execution token before becoming the running task. The running task releases the execution token whenever it becomes blocked or executes a delay statement.

The proposed syntax is:

```
pragma Task_Dispatching_Policy
   (Non_Preemptive_Fifo_Within_Priorities)
```

and

```
pragma Locking_Policy
   (Non_Preemptive_Locking)
```

The dynamic semantics of Non_Preemptive is similar to Fifo_Within_Priorities, both for handling the ready queues and the rule for selecting a new running task.

No locking or priority changes are required for protected objects (on a single CPU) unless a protected object is an interrupt handler (by using at least one of Interrupt_Priority, Interrupt_Handler, or Attach_Handler pragma); in that case Ceiling_Locking applies.

The running task may release the execution token by executing the delay 0.0 but may be reassigned it immediately if it is at the front of the highest priority ready queue (this is an easy way of inserting a preemption point).

Implementation Permission 9.5.3 (22) still applies (*An implementation may perform the sequence of steps of a protected action using any thread of control; it need not be that of the task that started the protected action. If an entry_body completes without requeuing, then the corresponding calling task may be made ready without waiting for the entire protected action to complete*).

It remains a bounded error to call a potentially blocking operation from within a protected object. This scheduling policy fits very well with the Ravenscar profile.

1.1. Discussions and comments

One comment was that the question of multi-CPUs must also be addressed, because ceiling locking is not sufficient for a multi-processor. A call to a blocked protected object in a multi-processor system is not a scheduling point. There was a discussion on whether the ARM did define processors and it does (actually in 18 different sections). It was agreed that the proposal should be fully developed for multi-processor systems.

The workshop found the motivation for migration from cyclic execution to be a valid positive one.

There was a discussion between the differences of the abort-completion points (ARM9.8) compared to the scheduling points defined for this scheduling policy. Abortion and ATC are asynchronous. Are scheduling points also abort-completion points? Further consideration of this point is needed.

The ARM says D.6 (2) *On a system with a single processor, an aborted construct is completed immediately at the first point that is outside the execution of an abort-deferred operation. Again multi-processor systems present*

particular issues that must be addressed. But note the Documentation Requirement:

D.6 (3) *On a multiprocessor, the implementation shall document any conditions that cause the completion of an aborted construct to be delayed later than what is specified for a single processor.*

It was agreed that the proposed scheduling policy can be useful for Ravenscar, but it has to also work well with a full Ada implementation. Ravenscar was argued to be more or less restricted to single processor, but there is nothing in the definition that does restrict Ravenscar to a single CPU. However all current implementations are on a single CPU.

1.2. Conclusion

Alan will continue to complete the definition and submit a paper to Ada Europe.

Adding a new policy does not imply that all implementations of Annex D have to implement it.

Conclusion: The proposal voted on was "*Is the proposed non-preemptive policy good and should it be defined in the full language*." The result was 23 in favour, 3 against and 2 abstentions.

A second vote that never came to a formal vote was "*Should it be an option in Ravenscar?*" The general view was that it is obvious.

2. Round Robin Scheduling

Michael González Harbour entered the stage and presented his proposals. The first one was round robin scheduling. The motivation is that if a system contains both real-time tasks and non-real-time tasks, the non-real-time tasks are better scheduled with a round-robin scheme. The use of `delay 0.0` could fix this, but is impractical to use. Another motivation is that this scheduling policy is supported by POSIX, thus for portability reasons it would be good to have it in Ada as well.

The interface to the round robin scheduling should use a Task_Dispatching_Policy named Fixed_Priorities, and a new pragma, where each task could be assigned an individual dispatching policy. The two choices for tasks are FIFO_Within_Priorities and Round_Robin_Within_-Priorities. The associated locking policy would be Ceiling_Locking. There should be one global time quantum to be used by the round robin scheduling, and it should be defined in package System.

2.1. Discussion

There were some discussions about the coupling between Ada and POSIX. Not all systems are built on top of POSIX.

Tasks scheduled according to round robin can call protected objects and in that case the implementation would be allowed to turn round robin scheduling off during the protected operation, to allow a priority-only based implementation of ceiling locking. Interrupts can eat round-robin time.

Round_Robin_Quantum is a constant in a given system, and cannot be changed (according to POSIX). For an application that supports the change of the quantum, a package could be supplied with routines for that purpose.

It was suggested that using a CPU-time clock one might be able to implement round robin scheduling by using a high priority task resetting the priority, and thus putting the task at the end of the queue. But concerns were raised that this was probably not possible.

It was discussed that the scheduling policies could be tied to different priority levels. If we go for on a per-task scheduling it would be possible to have some tasks be non-preemptive. What happens in this case if a task changes its priority?

2.2. Conclusion

When voting for Who are in favour for an optional round robin scheduling? the result was 20 for, one against and 3 abstentions. The reason for having it in the ARM is that we would like to have these policies to be well defined, and not to be different for various implementations.

The round robin quantum could be an attribute.

When voting for "*Should we have a mixed scheduling (RR and FIFO per task and mixed)*" the result was 15 for, 0 against, 8 abstain.

3. Sporadic Server Scheduling

Next topic for discussion presented by Michael was the issue of Sporadic Server Scheduling.

The motivation for this scheduling policy is that there is a need for aperiodic event handling. For timeliness, the handling of events requires a certain level of priority, but if this level of priority is maintained throughout the whole execution, excessive preemption may be caused to lower priority tasks with hard real-time requirements.

Another motivation is the same as for the round robin scheduling, i.e., it is found in POSIX.

3.1. Discussion

The argument for having POSIX support is for portability of applications from POSIX-based OS to other non-POSIX systems. The Ada POSIX binding is a well defined ISO-standard. Maybe this binding should be used instead. Vendors today supply POSIX bindings (the parts that the users need and want). There is a danger to implement everything that is defined in POSIX, since not all POSIX implementations implement everything.

Michael is pushing for an idea in the POSIX world for user-defined scheduling policies, which will require some underlying primitives, Andy Wellings thinks this is what we should be interested in from the Ada community.

3.2. Conclusions

The workshop may want different things out of the POSIX. We (the Ada people) will be more Ada-centric in what we bring into Ada from POSIX.

There is also the question if the Ada-POSIX binding is enough, do we need to put more into Ada?

There was a vote on whether "*Things that are in POSIX, should remain there and not be reimplemented in the Ada context*" the result was 7 for, 5 against, and the rest abstained. Consequently, there is not enough consensus to push standardization of all real-time POSIX services in Ada, globally.

Sporadic server was not pushed any further.

4. CPU-budgeting

Michael presented an idea for CPU-budgeting. There was a proposal within the Ada9X to handle individual per-task clocks. It did not enter the language because, even though it was fully thought out, it was not deemed to be important enough at the time.

4.1. Discussion and Conclusions

Michael will look at the original proposal, which people in the 9X-community will help to dig out.

It would be interesting to have elements of this feature in Ravenscar.

Although in Michael's proposal execution-time timers are protected objects with a protected entry on which a select-then-abort statement could be used to detect an execution time overrun, it was suggested that it would be better to have delay statements that could be invoked using a CPU-time type. One problem with this approach is that the relative delay only works for the type Duration. Another problem is that a delay on the task's own CPU-time clock would be a deadlock. These aspects should be further explored.

The vote "*Is cpu-budgeting essential for 200Y?*" resulted in 18 for, none against and 3 abstentions.

There were no further formal votes, but the workshop regarded the proposal as quite sensible.

5. Questions for Further Workshops

Are there any thoughts about EDF? (in the Java community there are). There is no notion of deadline in Ada. There are also several versions of EDF.