# Dynamic Ceiling Priorities in GNAT Implementation Report

Javier Miranda
Applied Microelectronics Research Institute (IUMA)
Universidad de Las Palmas de Gran Canaria, Spain
jmiranda@iuma.ulpgc.es

Edmond Schonberg
Courant Institute of Mathematical Sciences
New York University, USA
schonberg@cs.nyu.edu

Miguel Masmano, Jorge Real, Alfons Crespo
Universidad Politécnica de Valencia, Spain
{mmasmano,jorge,alfons@disca.upv.es}

## Abstract

*This document presents the required modifications to the GNAT compiler to support dynamic ceiling priorities in protected objects [1]. In a nutshell, dynamic ceilings for protected objects can be implemented in a simple and efficient manner in the GNAT compiler and run-time.*

## 1 Introduction

In Ada, the ceiling priority of a protected object (PO) is static and thus can only be assigned once, by means of pragma Priority, at PO creation time. By contrast, task priorities are dynamic. The ability to dynamically modify ceiling priorities is especially useful in multi-moded systems, systems scheduled with dynamic priorities, and dynamically adapting library units containing POs.

For multi-moded systems, a common workaround is to use the so called "ceiling of ceilings", i.e. the highest ceiling that PO will have across all operating modes. Unfortunately, this approach may have a considerable impact on blocking times: by using the ceiling of ceilings, a task running in a particular mode can be blocked by another task due to the ceiling priority of a PO being higher in another operating mode. It can be the case that the priority assignment for that particular mode would produce a lower ceiling for the PO, thus resulting in a null blocking time for some tasks.

Two new attributes are proposed in [1, 5] to assign or read the current PO's ceiling priority: 'Set_Ceiling and 'Get_Ceiling, respectively. These attributes can only be used from within the PO. More precisely:

- 'Set_Ceiling can only be called from within a protected procedure or entry of the affected PO.
- 'Get_Ceiling can be called from within any protected operation of the affected PO and it can also be used in barrier expressions at entries of the affected PO.

The prefix for both attributes can denote a protected type or a protected object. The thread currently executing the protected operation must be executing, at most, at the current ceiling priority of the PO. Otherwise, Program_Error is raised in the caller, as is the rule in Ada 95 for accessing a PO (ARM D.5-11). In other words, the ceiling change must follow the ceiling locking protocol. This approach eliminates the need for an additional lock on the PO.

The attribute 'Get_Ceiling can be used in a barrier expression of an entry of the affected PO. This feature can be very useful for the application to avoid violating the ceiling by tasks queued in protected entries while the ceiling change occurs. Nevertheless, 'Get_Ceiling is not a guarantee itself: an incorrect program may use 'Get_Ceiling in a barrier and still produce the error. For instance, if the task has a priority 10 and the barrier of a protected entry is "when PO'Get_Ceiling $< 5$", then the ceiling violation will eventually occur [1].

The change of the ceiling by means of 'Set_Ceiling does not take place until the end of the enclosing protected operation. This avoids the awkward situation that might occur when lowering the ceiling if the change was immediate. The implementation can easily conform to this semantics by using two variables for the ceiling: the one to become and the one still in effect. Set_Ceiling just assigns the first one. At the end of a protected procedure that modifies the ceiling, the new ceiling overwrites the old one. From that

moment on, tasks calling a protected operation on the PO will see the new ceiling. According to this semantics, a call to 'Get_Ceiling after a call to 'Set_Ceiling within the same protected operation will still return the old ceiling.

The rest of this paper is structured as follows: Section 2 presents the required modifications to the GNAT front-end; Section 3 presents the required modifications to the GNAT run-time. We close with some conclusions.

## 2 GNAT Front-end Modifications

In order to support dynamic ceiling priorities (described in [1]), the following modifications must be done to the GNAT-3.15p compiler:

1. Addition of a new compilation flag (-gnat+C). This flag enables the use of dynamic ceiling priorities for protected objects and, at the same time, permits us to check that the modifications done to the compiler are correct. For this purpose, after all the modifications described in this report were implemented, all the front-end sources were compiled with the modified GNAT front-end, and the resulting compiler was checked to be correct. The addition of the new flag follows the steps described in [3], Section 3.1 (cf. GNAT files switch-c.adb and opt.ads).

2. Addition of new identifiers, i.e. names known to the compiler: Get_Ceiling and Set_Ceiling. The addition of these identifiers follows the steps described in [4], Section 2 (cf. GNAT files s-names.ads, s-names-adb, and a-snames.h).

3. Semantic analysis of the new attributes (cf. GNAT file sem_attr.adb). The semantic analyzer must ensure that the new attributes are called from within a protected operation of the affected protected object. In addition, Set_Ceiling can not be called from within a protected function, and Get_Ceiling can be used in the barrier expressions of entries of the affected protected object.

   NOTE: Although [1] also proposes the addition of a new pragma to mark the protected objects with dynamic ceiling priorities, we have found that this is not really a requirement. In our implementation the semantic analyzer automatically marks the protected types that use the new attributes, and thus no additional cost is added to POs that do not use the new attributes.

4. Expansion of the new attributes (cf. GNAT file exp_attr.adb). Get_Ceiling is expanded into a call to a run-time function that returns the current ceiling of the protected object. Set_Ceiling is expanded into a call to a run-time procedure that sets the new ceiling of the protected object. The change of the ceiling does not take place until the end of the enclosing protected operation (as described later). In order to allow the front-end to call these new run-time subprograms, they are registered in the GNAT file rtsfind.ads, which provides name links between the compiler and run-time subprograms.

## 3 GNAT Run-Time Modifications

The GNAT implementation of protected objects considers two special cases: (1) PO without entries, and (2) PO with entries. Although this is important for the GNAT compiler (to give support to critical systems with special restrictions), it does not add special complexity to our implementation: the routines must just be added in two run-time packages (cf. GNAT files s-taprob.ad[sb] and s-tpoben.ad[sb]). In the following sections we present the basic functionality of the new run-time subprograms, and the required modifications for each case.

### 3.1 New Run-Time Data

Following the proposal discussed in [1], a new field was added to the GNAT data type associated to the protected object:

```
type Protection is record
   ...
   New_Ceiling : System.Any_Priority;
end record;
```

Obviously, the run-time routine responsible for the protected object initialization was also modified to set the initial value of New_Ceiling to the initial ceiling of the protected object.

### 3.2 New Run-Time Subprograms

The new run-time subprograms basically follow the implementation proposed in [1]. Their body is as follows:

```
function Get_Ceiling (Object : Protection)
                      return Any_Priority is
begin
   return Object.Ceiling;
end Get_Ceiling;

procedure Set_Ceiling (Object   : Protection_Access;
                       Priority : Any_Priority) is
begin
   Object.New_Ceiling := Priority;
end Set_Ceiling;

procedure Adjust_Ceiling (Object : Protection_Access) is
begin
   Set_Prio_Ceiling (Object.L, Object.New_Ceiling);
   Object.Ceiling := Object.New_Ceiling;
end Adjust_Ceiling;
```

The service Set_Prio_Ceiling is not available in the current versions of GNAT. Its implementation should call the

corresponding POSIX service to update the ceiling of the PO's lock. Note that a bare-machine implementation of the run-time system could avoid this call only by relying on priorities and the ceiling locking policy to achieve mutually exclusive access to the PO.

In the following sections we discuss how and where the front-end generates calls to Adjust_Ceiling, which is the critical point at which the new ceiling is set and the new value becomes visible to other tasks accessing the object.

Readers with a deep knowledge of the GNAT run-time system will detect that the code fragments in next subsections have minor differences with respect to the real code generated by the compiler. We have adapted the code expanded by GNAT to the Ada syntax to ease its reading without the need of special linguistic support outside of the scope of standard Ada.

### 3.2.1 PO without entries

For each protected operation Op, the GNAT compiler generates two subprograms OpP and OpN. OpP simply takes the object lock, and invokes OpN. OpN contains the user code. If a call is an internal call, i.e. a call from within an operation of the same object, the call invokes OpN directly. If the call is external, it is implemented as a call to OpP. In addition, one additional parameter is added by the compiler to the parameters profile of the protected subprograms: the object itself, designated _object in the sources. For example:

```
procedure procN (_object: in out poV; ... );
procedure procP (_object: in out poV; ... );
```

In order to see the point at which the call to Adjust_Ceiling must be done, let us see the code of the P subprogram in detail.

```
procedure procP (_object : in out poV; ... ) is

  procedure Clean is
  begin
     GNARL.Adjust_Ceiling (_object._object'access);
     GNARL.Unlock (_object._object'access);
     GNARL.Abort_Undefer;
  end Clean;

begin
   GNARL.Abort_Defer;
   GNARL.Lock_Write (_object._object'access);
   procN (_object; ... );
   Clean;
exception
   when others =>
      declare
         E : Exception_Occurrence;
      begin
         GNARL.Save_Occurrence
               (E, GNARL.Get_Current_Exception);
         Clean;
         GNARL.Reraise (E);
      end;
end procP;
```

As the reader can see, this case is straightforward; because the protected object has no entries, we just need to set the new ceiling of the protected object inside the Clean subprogram (which is called at the end of the protected subprogram execution, whether the operation completes successfully or abnormally). Therefore we modified the corresponding expander package (GNAT file exp_ch7.adb) to generate this call. Obviously, because protected functions cannot modify the ceiling of the protected object, the call to Adjust_Ceiling is not generated in the Clean subprogram associated with protected functions.

### 3.2.2 PO with entries

Two cases must be considered here: protected procedures and protected entries. Since GNAT follows the proxy model for the implementation of protected objects (cf. [2], Chapter 5), after the code of a protected entry or protected procedure is executed, the run-time subprogram Service_Entries is called to service calls queued on entries of the PO. The desired semantics defined by the IRTAW are to postpone the effective ceiling change until the end of the protected action, not just the protected operation. Therefore, the evaluation of guards and the execution of entry bodies with open barriers are performed at the old ceiling priority.

**1 - Protected Procedures**

```
 1: procedure procP (_object : in out poV; ... ) is
 2:
 3:   procedure Clean is
 4:   begin
 5:      GNARL.Service_Entries (_object._object'access);
 6:      GNARL.Adjust_Ceiling (_object._object'access);
 7:      GNARL.Unlock (_object._object'access);
 8:      GNARL.Abort_Undefer;
 9:   end Clean;
10:
11 begin
12:    GNARL.Abort_Defer;
13:    GNARL.Lock_Write (_object._object'access);
14:    procN (_object; ... );
15:    Clean;
16: exception
17:    when others =>
18:       declare
19:          E : Exception_Occurrence;
20:       begin
21:          GNARL.Save_Occurrence
22:               (E, GNARL.Get_Current_Exception);
23:          Clean;
24:          GNARL.Reraise (E);
25:       end;
26: end procP;
```

This case is similar to the case of a protected object without entries. The main difference can be found in line 5 (the call to the Service_Entries routine). In this case, the call to Adjust_Ceiling must be added "after" the call to Service_Entries because, according to [1], the pending calls and the barriers must be evaluated with the old active priority. The run-time subprogram Service_Entries needs not be modified since, according to [1], the ceiling change does not take effect until the end of the protected action, which in the proxy model includes servicing queued tasks whose

barriers have opened as the effect of executing the protected subprogram.

**2 - Protected Entries** The entry body is translated by the GNAT compiler into a procedure [2]:

```
procedure Entry_Name
   (Object      : Address;
    Parameters  : Address;
    Entry_Index : Protected_Entry_Index) is
  ...
begin
   <Statement_Sequence>
   GNARL.Complete_Entry_Body (_object._object);
exception
   when others =>
      GNARL.Exceptional_Complete_Entry_Body
         (_object._object, GNARL.Get_GNAT_Exception);
end Entry_Name;
```

In this case we add the call to the new Adjust_Ceiling routine inside the Exceptional_Complete_Entry_Body subprogram (Complete_Entry_Body calls Exceptional_Complete_Entry_Body with a null exception occurrence).

### 3.3 Exception Handling

If an exception occurs inside the protected operation that changes the ceiling, the ceiling remains unchanged.

Queued tasks whose barriers become open as the effect of executing the protected operation where 'Set_Priority is called, will not produce ceiling violations since the change of the ceiling does not take effect until the end of the protected action, therefore these tasks with open barriers will still execute at the old ceiling priority (this is immediate in the proxy model implementation of protected objects, but requires an additional ceiling check in the self-service model).

A queued task whose barrier remains closed after the protected action, could violate the ceiling when the barrier gets open. This situation is already considered by the language (ARM paragraph D.5(11) [6]) with respect to dynamic priorities for tasks. A ceiling violation may occur due to dynamic priorities for tasks or, with dynamic ceilings, due to the possibility of dynamically changing the ceiling. The language allows to temporarily run the task with a lowered priority, or raise Program_Error to the queued task, or both, or neither.

## 4 Acknowledgements

This work has been partly funded by the Spanish Government's Ministry of Science and Technology projects number DPI2002-04432-C03-01 and TIC2001–1586–C03–03. Thanks to Arnaud Charlet, Michael González and Mario Aldea for their comments.

## 5 Conclusions

The conclusion from this work is clear: dynamic ceilings can be efficiently implemented in the GNAT Ada compiler. Nevertheless, it is important to note that implementations of the run-time system on an underlying POSIX layer, need the pthread_mutex_setprioceiling service. Moreover, we have found that the POSIX standard does not specify whether a task already owning the lock of a mutex is allowed to call this service on that mutex. A clarification request has been issued to the POSIX committee about this service.

Initially, a pragma was proposed to make this feature optional for each protected type. The idea behind the proposal was that the overhead of dynamic ceilings could affect the performance of protected operations in general, even in the case of a PO with static ceiling. But, because the semantic analyzer can mark the protected types which use the new attributes, the additional pragma is no longer required: if a protected object does not use dynamic ceilings, then it imposes no additional run-time cost.

## References

[1] Ada Issue AI-00327. Dynamic Ceiling Priorities. http://www.ada-auth.org/ais.html, September 2003.

[2] J. Miranda. A Detailed Description of the GNU Ada Run-Time. http://gnat.webhop.info and http://www.iuma.ulpgc.es/users/jmiranda, June 2002.

[3] J. Miranda, F. Guerra, E. Martel, J. Martin, and A. Gonzalez. How to Use GNAT to Efficiently Preprocess New Ada Sentences. In *J.Blieberger, A. Strohmeier (Eds): Proceedings of 7th Reliable Software Technologies - Ada-Europe 2002, Lecture Notes in Computer Science*, volume 2361, pages 179–192. Springer Verlag, June 2003.

[4] J. Miranda, F. Guerra, J. Martin, and A. Gonzalez. How to Modify the GNAT Front-End to experiment with Ada Extensions. In *J. P. Rosen, A. Strohmeier (Eds): Proceedings of 8th Reliable Software Technologies - Ada-Europe 2003, Toulouse, France, June 16-20, 2003. Lecture Notes in Computer Science*, volume 1622, pages 226–237. Springer Verlag, June 1999.

[5] J. Real, A. Crespo, A. Burns, and A. Wellings. Protected Ceiling Changes. *Ada Letters*, XXII(4):66–71, 2002.

[6] T. Taft and R. Duff and R. Brukardt and E. Ploedereder (eds.). Consolidated Ada Reference Manual. *Springer, Lecture Notes on Computer Science*, 2219, 2000.