Exception Support for the Ravenscar Profile*

José F. Ruiz Juan A. de la Puente Juan Zamorano Ramón Fernández-Marina

Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid, Spain
E-mail: {jfruiz,jpuente,rfm}@dit.upm.es jzamora@fi.upm.es

Abstract

The GNAT/ORK cross-compilation system was designed to support the Ravenscar profile semantics in a simple and efficient way. The implementation of the underlying kernel support for the exception handling was not difficult, but it raised two main issues. First, the handling of certain exceptions (namely task termination and elaboration) is undefined in the Ravenscar profile. Second, the kernel mechanism required to support Ada exceptions (using the GNAT runtime) is very simple if we do not adhere to the complex POSIX signal approach.

1 Introduction

This paper describes some issues raised while implementing the Ada exception support in the Open Ravenscar Real-time Kernel (ORK) [6], an open-source real-time kernel of reduced size and complexity, compliant with the Ravenscar profile, and integrated into the GNAT cross-compilation system.

First, it must be noted that the Ravenscar profile does not place any limit on the features of sequential Ada [4]. Hence, it does not restrict the use of exceptions (either user defined or predefined exceptions), so that ORK was forced to implement the required support for the exception handling. Likewise, no a priori limit was imposed (by the Ravenscar profile) to the runtime and the kernel on the use of internal exceptions, although the kernel does not use exceptions (see section 4).

Handling Ada exceptions require some kernel support, but our experience with ORK showed that the mechanisms that must be implemented are indeed very simple.

The difficulties came up from the collateral effects that the Ravenscar tasking model have on exception occurrences. It was not anticipated that the behavior in case of task termination (forbidden by Ravenscar) and errors in task elaboration was left completely undefined in the Ravenscar profile. A clear description of what happens in these cases is mandatory for having a good and complete definition of the profile.

Sections 2 and 3 of this paper explain how exceptions are handled in GNAT and ORK respectively. The runtime support needed to handle these exceptions is described in section 4. Section 5 illustrates the problems that came up from the under-specification of the handling of some exceptions within the Ravenscar profile, while section 6 shows some alternatives that could fill this gap. Finally, section 7 outlines some conclusions that came up from the implementation of the exception support for the Ravenscar profile.

2 Exception Model in GNAT

Current implementation of exception handling in GNAT uses setjmp and longjmp functions to save and restore exception scopes. This is not a very efficient manner of handling nested exception scopes, but it is quite simple and easy to implement.

There is a more efficient exception handling procedure, called the "zero-cost exception model" [1]. This scheme was designed to manage the cascading in exception parts more efficiently, based on storing tracebacks in exception occurrences when the target supports it. However, this option is currently only supported on *Solaris* and *Linux* where you explicitly need to use the gnatgcc flag -funwind-tables when compiling every file in your application.

The way how synchronous hardware traps (division by zero, stack overflow, etc.) are handled as Ada exceptions in GNAT is by means of POSIX signal mechanisms. Hardware traps are translated by the kernel into some reserved POSIX signals (SIGFPE, SIGSEGV, ...), and these signals are mapped by the GNAT runtime into standard Ada exceptions [8] (Storage_Error and Constraint_Error). This mechanism is not very efficient due to the burden and overhead of POSIX signals, but it is highly portable. The following section describes how efficiency can be improved.

*This work has been funded by ESA/ESTEC contract no. No.13863/99/NL/MV.

3 Exception Handling in GNAT/ORK

GNAT/ORK uses the mechanisms provided by GNAT runtime to handle exceptions. The runtime relies on the underlying ORK kernel layer only for the translation from hardware exception occurrences into the required Ada exception. ORK is only in charge of redirecting trap occurrence to the appropriate exception handler. It is achieved by filling the trap table of the processor with the pointer to a generic wrapper function which in turn calls the appropriate Ada exception handler. This way hardware exceptions are easily translated into Ada exceptions in ORK.

This approach is very simple and efficient, even more if compared to the default one in GNAT, which relies on the complex POSIX signal mechanism. Of course the ORK implementation has a penalty, as the portability of the POSIX signals is higher.

4 Exception Occurrence Inside the Runtime

The ORK kernel was carefully designed following the recommendations made by the *Ada High Integrity Systems Standard* [7]. A safe subset of Ada was defined for the implementation of the kernel [10]. No formal method (such as, for example, the one supported by SPARK [3]) was used to avoid exception occurrences. However, assuming that applications do not make direct calls to the kernel, but to GNAT runtime, and that GNAT runtime has been carefully designed and tested, we have a high degree of confidence about the absence of unexpected error situations. Moreover, the static nature of a Ravenscar compliant program reduces notably the potential casuistry. Hence, the kernel itself has neither explicit exception handler nor explicit exception raising statements.

Nevertheless, if an exception is raised while executing a kernel operation (for example due to a stack overflow) this exception is propagated to the task that invoked the kernel function, and hence it is the task responsibility to handle this exception adequately.

5 The Ravenscar Exception Model

The Ravenscar profile places some restrictions that make exception occurrence and handling different from full Ada. Exceptions that can appear in a Ravenscar compliant program can be categorized as follows:

- Synchronous traps. These hardware traps are redirected to Ada exceptions. For example, division by zero must raise Constraint_Error, accesses to non aligned data must raise Storage_Error, etc. The ORK initialisation fills the required places of the trap table with a pointer to a wrapper procedure which calls the appropriate exception handler.
- Software exceptions. They are raised by the user software by means of "raise" statements. They are simply translated by the runtime into the required procedure call, and hence it does not need support from the underlying kernel.
- Exceptions raised by the kernel. The ORK kernel does not use any internally defined exception. All the required exceptions are raised either from the upper GNAT runtime layers or from synchronous traps (which are redirected to Ada exceptions).
- Exceptions in interrupt handlers. Any exception raised within an interrupt handler causes further execution of the handler to be abandoned, and it can only be handled within the scope of that interrupt handler. Any exception propagated from that interrupt handler has no effect. These are the required semantics specified by ALRM C.3 [2], and hence the way how ORK handle them.
- Blocking statements in protected procedures and functions. According to the ALRM 9.5.1 a bounded error occurs when calling a blocking or potentially blocking procedure or statement in a protected procedure. ORK always detects this bounded error and raises Program_Error.
- Ceiling violations. Ceiling Locking policy is mandatory for Ravenscar compliant programs. Hence, when a task calls a protected operation, a check is made that its active priority is not higher than the ceiling of the corresponding protected object. Program_Error is raised to the calling task if this check fails.
- Entry calls made on an entry that already has a queued call. In this case the queue length would become higher than one, and as it is forbidden by the Ravenscar profile a Program_Error is raised to the task that made the call.
- Exceptions in task elaboration. Any exception during task elaboration should make the exception Tasking_Error to be propagated in the parent unit. However, the Ravenscar profile does not allow the handling of this exception because of the non-hierarchical definition of tasks in the Ravenscar profile (the Ravenscar profile only allows library level tasks). The library package where the task has been declared is the parent unit of the task. Therefore, any application with a task raising an exception during its elaboration will silently terminate the faulty task at elaboration time, while the application will try to continue its 'normal' execution.

- Task termination. This case is a bounded error under the Ravenscar profile. The ALRM 1.5 requires the effect of a bounded error to be bounded and documented. Raising Program_Error is explicitly allowed as a possible outcome of such an error, but the occurrence of this exception cannot be handled, as the parent unit for any task in a Ravenscar compliant program is a library unit. The task will then terminate silently unless any other action could be taken. ORK allows the user to change the default behavior in case of task termination to give a chance to the application to take any corrective action.

The last four cases of exception occurrences cited previously are specific for the Ravenscar profile. The exception raised when there is a ceiling violation or when a task makes an entry call on an entry that already has a queued call has no ambiguity. However, the behavior in the case of exceptions in task elaboration and task termination was left undefined by the Ravenscar profile. It is our believe that these aspects are very important and there should be a clear definition of the expected behavior within the profile definition. It is unsafe to allow these two events to be hidden to the programmer. Our proposed solution to these situations is described in the following section.

6 Extensions to the Ravenscar Exception Model

This section tries to define some issues about how the Ada exception model could be modified to determine a completely characterized behavior for Ravenscar compliant programs.

In the case of task termination, the ORK software implements an internal procedure that is executed when any task terminates [9]. This procedure, by default, suspends the tasks forever (silent death). The ORK programming model allows the user to modify this default behavior by replacing this procedure by an user defined one (using one internal procedure called Set_Exit_Task_Procedure).

This way, application programmers can define their own recovery policy. However, this solution is obviously non portable to other systems. It is desirable a solution integrated into the language, such as defining a new standard exception valid under the Ravenscar profile restrictions (that could be called Task_Termination). This way, each task can associate its own exception handler to that event.

The case of an exception during task elaboration is more complex. The silent death of a task that could not elaborate is not desirable at all. The solution to this problem possibly requires the definition of new elaboration rules. These rules should allow users to specify the desired behavior when tasks cannot finish its elaboration.

The current definition of the Ravenscar profile allows a task to have several invocation points, such as a delay until, an entry call to a protected object, and a wait call on a suspension object. The Ravenscar programming model was originally defined to support the single invocation restriction [5] for tasks, under which tasks can be either time triggered or event triggered. This model is currently supported, but it is not compulsory. If it were, the analysis of the system would be easier. A new exception (such as Single_Invocation_Violation) could be raised at runtime when violating this model.

Another potential simplification for the exception model in Ada 95 (and by extension to the Ravenscar profile) is the limitation of the places where exception can be handled. For example, it is quite usual to have exception handlers defined only at the outer and the inner levels of nesting, and not in the levels between. If this scheme is detected (or defined by the user), the exception handling protocol can be highly lighten because the context of the intermediate levels would not need to be stored.

7 Conclusions

Error detection and recovery in Ada is based on exceptions (ALRM 11). Ada exceptions provide a high-level, structured mechanism for building reliable applications on top of it, leaving the application programmer a high degree of flexibility. Therefore, it is essential to support exceptions within the Ravenscar profile, but a detailed definition of the desired behavior in all cases would be very valuable. Some special situations appear when using the Ravenscar profile (such as task termination and exceptions in task elaboration), and therefore it is not enough to rely on the semantics specified by the *Ada 95 Reference Manual*. The Ravenscar profile must address the definition of the desired behavior in these situations in order to be valuable and usable.

By other side, the implementation of ORK showed that the kernel mechanisms required to support Ada exceptions are very simple. No specific kernel functionality is necessary, but the means of redirecting hardware traps to software exception handlers. This functionality is usually achieved using POSIX signals, but ORK provides a simpler mechanism that achieves the work more efficiently, although it is less portable.

References

[1] Ada Core Technologies. *GNAT Reference Manual. Version 3.14a*, January 2001.

[2] *Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-*

8652:1995, 1995. Available from Springer-Verlag, L-NCS no. 1246.

[3] John Barnes. *High Integrity Ada. The SPARK Approach*. Addison Wesley, 1997.

[4] Alan Burns. The Ravenscar profile. Technical report, University of York, 2000. Available at `http://www.cs.york.ac.uk/~burns/ravenscar.ps`.

[5] Alan Burns, Brian Dobbing, and George Romanski. The Ravenscar profile for high integrity real-time programs. In Lars Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, number 1411 in L-NCS. Springer-Verlag, 1998.

[6] Juan A. de la Puente, José F. Ruiz, and Juan Zamorano. An open Ravenscar real-time kernel for GNAT. In Hubert B. Keller and Erhard Ploedereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[7] ISO/IEC/JTC1/SC22/WG9. *Guide for the use of the Ada Programming Language in High Integrity Systems*, 2000. ISO/IEC TR 15942:2000.

[8] Dong-Ik Oh, T.P. Baker, and Seung-Jin Moon. The GNARL implementation of POSIX/Ada signal services. In *Proceedings of the Ada-Europe'96*, 1996.

[9] UPM. *Open Ravenscar Kernel — Operation Manual*, March 2001. Revision 2.2.

[10] UPM. *Open Ravenscar Kernel — Software Design Document*, February 2001. Revision 1.8.