ORK: An Open Source Real-Time Kernel for On-Board Software Systems*

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

Departamento de Ingeniería de Sistemas Telemáticos
ETSIT, Universidad Politécnica de Madrid
Ciudad Universitaria, E-28040 Madrid, Spain
Phone +34 9 13 36 73 42; Fax +34 9 13 36 73 33

Abstract

Ada tasking is a powerful abstraction mechanism for developing concurrent systems. However, many implementations of concurrent tasking have been seen as potentially unsafe for critical systems because of their high degree of indeterminism. The Ravenscar profile is a subset of Ada 95 tasking with purpose of providing a basis for the implementation of certifiable critical systems. ORK is an open-source real-time kernel which provides full conformance with the Ravenscar profile on ERC32 computers. The kernel has a reduced size and complexity, and has been carefully designed to allow the building of reliable software for on-board space applications. This kernel is integrated in a cross-compilation system based on GNAT 3.13, including a tasking-aware version of GDB.

1 Introduction

Mission-critical on-board software has usually been developed on top of a cyclic executive that invokes the execution of application tasks according to a predefined static schedule. There are thus no concurrent threads of execution, and the application code is made of a set of purely sequential procedures.

This approach leads to simple, robust implementations, and provides a deterministic time behaviour which has often been considered a requirement for critical real-time systems. However, as the functionality and complexity of on-board software increases, and there is more and more pressure for shortening development times and reducing costs, while keeping up with critical reliability requirements, its low-level nature and lack of flexibility make it less appropriate. As a consequence, more attention is being devoted to higher level, abstract development methods that include concurrency as a means of decoupling application tasks and making software easier to design and test [20].

Indeed, many implementations of concurrent tasking have been seen as potentially unsafe for critical systems because of their high degree of indeterminism, which may make programs difficult to validate. This has led to either completely banning tasking for critical software applications, which is the traditional approach, or to the more flexible approach of building specialised kernels with reduced functionality. By limiting the way tasks are executed and synchronized, it can be expected that concurrent systems can be analysed and tested, so that safe concurrent systems can be built in a way that has significant advantages over cyclic executives from the point of view of flexibility and structuring [13].

Ada [1] is the language of choice for many critical systems due to its careful design and the existence of clear guidelines for building safe systems [12]. While the first approaches to developing safe Ada software did not make use of Ada tasking [10, 4], recent advances in real-time systems timing analysis methods [2] have paved the way to safe tasking in Ada. The Ravenscar profile [3, 6] is a subset of Ada 95 tasking that was defined at the 8th International Real-Time Ada Workshop (IRTAWS) with purpose of providing a basis for the implementation of certifiable critical systems. The first implementation of the profile, Aonix’ Raven, has indeed shown the feasibility of the approach and the possibility of building certifiable applications based on it [8].

Based on this early experience, the European Space Research and Technology Centre (ESTEC) launched the Open Ravenscar Real-time kernel (ORK) project in September 1999. The aim of the project is to develop an open-source kernel, compliant with the Ravenscar profile, for its current standard on-board computer, ERC-32, which is a radiation-hardened

*The work described in this paper is being carried out under ESA/ESTEC contract no. No.13863/99/NL/MV.
implementation of the SPARC v7 architecture. The kernel is to be integrated with the GNAT/GCC compilation system [17], as well as GDB and other open-source tools, and can be used to develop both Ada 95 and C programs in such a way that certification may be sought by users in the future.

2 The Ravenscar profile

The Ravenscar profile defines a subset of the tasking features of Ada 95 for high-integrity applications. The profile is based on a computation model with the following features:

- A single processor.
- A fixed number of tasks.
- A single invocation event for each task. The invocation event may be generated by the passing of time (for time-triggered tasks) or by a signal from either another task or the environment (for sporadic tasks).
- Task interaction only by means of shared data (protected objects) with mutually exclusive access.

The profile forbids many of the most complex tasking features, including dynamic and nested tasks and protected objects, requeue, asynchronous transfer of control, task termination, task abortion, task entries and rendez-vous, dynamic priorities, relative delays, select statements, and multiple protected entries.

The tasking model defined by the profile includes tasks and protected types and objects at the library level, a maximum of one protected entry with a simple boolean barrier for synchronization, a real-time clock, absolute delays, preemptive priority scheduling with ceiling locking access to protected objects, and protected procedure interrupt handlers, as well as some other features, which allow the development of on-board embedded real-time systems. For a full description, see [5].

The Ravenscar profile is well adapted to space applications. It defines a computation model similar to the one proposed by Vardanega [19], which is based on the HRT-HOOD method [7]. The profile allows implementing space on-board systems with the tasking facilities provided by Ada, restricted so as to ensure that the system can be analysed for accurate timing and safety requirements. Preliminary experience confirms the value of this approach for on-board software development [18].

3 The Open Ravenscar Real-Time Kernel

3.1 General description

The Open Ravenscar Real-Time Kernel (ORK) provides full conformance with the Ravenscar profile on a ERC32 computer. It is integrated in a cross-compilation system based on GNAT 3.13. In this way, the restrictions of the "standard" (i.e. IRTAW defined) Ravenscar profile can be enforced on the application programs at compilation time by means of appropriate GNAT restriction configuration file, as shown in figure 1.

```plaintext
-- gnat.adc file for Ravenscar programs
pragma Ravenscar;
pragma Restrictions (...); -- additional restrictions, if any
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
```

Figure 1: Sample configuration file for GNAT

The kernel can be used as a supporting execution environment for on-board real-time software running on a bare ERC32 machine, with no other underlying operating system.

The development system for programs using the kernel is based on a PC-compatible workstation with the Linux operating system (porting to Solaris is envisaged in the near future). The current GNAT software tools can be used for system developments. These software tools include the GNAT Emacs Ada mode and GLIDE (an integrated development environment for GNAT).
Although the kernel is mainly intended at Ada 95 applications, a C interface is also available that supports partial or total development in C using GCC for the same target.

3.2 Integration with the GNAT compilation system

The ORK software includes, in addition to the kernel itself, a specific cross-compilation system based on GNAT 3.13 and targeted to ERC32 computers. An enhanced version of GDB (the GNU debugger) for the ORK kernel is also included.

The standard GDB has been improved to deal with additional ORK tasking awareness. DDD, the graphical front-end for GDB, has also been modified in order to include inside its GUI these new debugging capabilities.

Ada tasking is implemented in GNAT by means of the run-time library, called GNARL [9]. The latest versions of GNARL (since GNAT 3.12) include a restricted run-time intended to provide just the functionality required by the Ravenscar profile. This reduced run-time library is used if pragma Ravenscar is included in the GNAT configuration file (see figure 1). This restricted run-time is used to provide tasking functionality on top of ORK.

The parts of GNARL which are dependent on a particular machine and operating system are known as GNULL, and its interface to the platform-independent part of the GNARL is called GNULLI. Most implementations of GNULL are built as a layer of glue code added on top of an existing set of POSIX thread functions [11], which in turn may be implemented on top of an operating system.

The ORK version of GNAT uses this approach, but instead of providing a full POSIX conformant kernel interface, which would impose too much overhead on the system, as GNARL already provides most of the functionality which is needed for tasking, takes advantage of a simpler interface at the kernel level, which provides an almost direct implementation of GNULLI, with the GNULL packages acting as a thin glue layer for GNAT (figure 2).

<table>
<thead>
<tr>
<th>Hardware (ERC32)</th>
<th>ORK (Open Ravenscar Real-Time Kernel)</th>
<th>GNULL (GNU Low-level Library)</th>
<th>Restricted GNARL (GNU Ada Runtime Library)</th>
<th>Ada 95 Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GNARLI (GNARL Interface)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GNULLLI (GNULL Interface)</td>
</tr>
</tbody>
</table>

Figure 2: Architecture of the GNAT run-time system based on ORK

3.3 Building applications with ORK

Ada applications are developed in the usual way, by writing a set of Ada packages that are compiled and linked using the GNAT/ORK compilation system (figure 3). Of course, only Ada 95 programs complying with the Ravenscar profile restrictions can be compiled and linked with the ORK version of GNAT. This can be checked at compilation time by using appropriate restriction pragmas. Therefore, a configuration file as shown in figure 1 is mandatory. In this way, programs are forced to use only the subset of Ada tasking defined by the profile. Additional restrictions on the sequential part of the language can also be enforced by including the corresponding restriction pragmas in the GNAT configuration file.

C applications are developed in much the same way, except that it is not possible to enforce the profile at compilation time. Tasking is implemented by means of procedure calls to kernel operations, using the ORK API for C.

4 ORK functionality

The functionality provided by ORK can be divided into the following sets of services:
Task (thread) management. Ada tasks are implemented at the lower level by kernel threads. Threads are created at program startup, so that there is no need to dynamically allocate or deallocate resources such as TCBs or stack space.

Real-time scheduling. Threads are scheduled according to the FIFO within priorities policy (ALRM [1], D2). There is a ready queue which is ordered by priority and arrival order. The synchronization primitives directly insert or remove threads from the ready queue.

Synchronization. Two kinds of synchronization elements are provided by the kernel: mutexes and condition variables. These elements are used by GNARL to implement protected objects and their implementation has been simplified taking advantage of the Ravenscar profile restrictions.

Mutexes implement the Immediate Priority Ceiling Protocol (IPCP), which is the same as the Ceiling Locking policy defined by the ALRM [1] (D3).

Storage allocation. Only a limited form of memory management is provided, in order to allocate TCB and stack space for new threads, because it seems reasonable to expect Ravenscar compliant programs not to use dynamic storage pools.

Time-keeping and delays. According to the Ravenscar profile specification, only an absolute delay (delay until) operation is provided. Delayed threads are kept in a delay queue which is ordered by wake up time.

Ada interrupt handlers. Ravenscar profile restrictions avoid a protected procedure handler to be blocked waiting for a protected object to become free. Therefore, ORK kernel uses protected procedures (together with some kernel preamble and epilogue) as low level interrupt handlers.

Stack checking. When a task tries to move its stack pointer outside the bounds of its stack Storage_Error is raised. This mechanism is implemented using the memory access protection functionality provided by the MEC (Memory Controller) of ERC32.

5 ORK architecture

In order to implement the ORK functionality, the kernel is divided into a set of Ada packages, all of them children of an empty root package called Kernel (figure 4). The kernel is coded using Ada 95, except for some low-level routines (coded in assembly). A brief description of these packages is:

Kernel.Threads. This package provides all the support required for thread creation, scheduling, and synchronization.
Kernel.Time. This package provides support for the real-time clock and absolute delays.

Kernel.Interrupts. This package provides support for interrupt identification, and for attaching interrupt handlers to interrupts.

Kernel.Memory. This package provides limited dynamic storage management.

Kernel.Parameters. This package contains types and constants which can be modified by the user to tailor the kernel to a concrete board or application.

Kernel.CPU_Primitives. This package isolates the target processor dependent primitives.

Kernel.Peripherals. This package provides the interface to the peripherals available in the target board.

The porting of GNAT to ORK has included the development of an adapted version of GNU LL, the GNU Lower Level packages that are part of GNARL, the GNU Ada Runtime Library for GNAT. Also some of the higher level GNARL packages are also adapted in order to provide Ravenscar profile conformance.

In this way, the package System.Interrupts has been reimplemented to provide Ada interrupt handling with the restricted GNARL. The standard implementation of System.Interrupts use interrupt service tasks that are activated when an interrupt occurs, and then call the associated interrupt handler procedure. These service tasks have entries, which are forbidden in the Ravenscar profile.

The kernel interface is a purely procedural one (i.e. ORK is a non-threaded kernel), as there is no need for separate user and supervisor execution modes. All the program runs in supervisor mode, as it is common in embedded systems. Mutual exclusion in the kernel is achieved by means of a monolithic monitor [14] protected by disabling interrupts, so that interrupt delivery is postponed when a kernel function is executed [16, 15].

The kernel supports 15 different priorities in the range of Interrupt_Priority. These priorities correspond with the SPARC architecture processor interrupt levels. Ada tasks can execute at these interrupt priority levels (masking the lower level interrupts). Also, in order to shorten the interrupt latency, the nesting of Ada interrupt handlers (protected procedures) is supported by the kernel.
6 Conclusions and future work

The current version of the kernel supports the whole Ravenscar profile, and is available at the ORK project server www.openravenscar.org. The distribution package includes binaries, the adapted sources of the tools needed for building the ORK cross-compilation system (binutils, gcc, newlib, GNAT, GDB), GDB scripts for the improved ORK tasking debugging capabilities, and the ORK kernel itself. This current version is targeted to the SPARC Instruction Simulator (SIS) (ftp://ftp.estec.esa.nl/pub/wsd/erc32/tools/sis/).

ORK has been implemented to provide an open source kernel compliant with the Ravenscar profile. Although the ORK project is primarily targeted to the on-board space software domain, we expect that it will be useful to the hard real-time systems community at large.

ORK has been carefully designed to isolate hardware dependencies, allowing easy retargeting. We do not envisage major problems in porting ORK to other hardware monoprocessor platforms, such as PC compatible or PowerPC architectures.

We plan to integrate the ORK kernel with other ERC32 open tools, such as Remote Debugger Monitor (rdbmon). This will enable it to be used with real ERC32 targets.

Acknowledgments

The Open Ravenscar Real-Time Kernel project is being carried out by a consortium including the Technical University of Madrid (UPM), CASA, and the University of York. We especially acknowledge the fruitful discussions we have had with Ángel Álvarez, Alejandro Alonso, Jesús González-Barahona, Vicente Matellán, Alan Burns, and Andy Wellings, as well as with Jorge Amador and Tullio Vardanega. Mike Kamrad and Joel Sherrill have significantly contributed to our insight in the kernel from their experience with the Top Layer and RTEMS kernel respectively.
References


