Real-Time Programming with GNAT: Specialised Kernels versus POSIX Threads[1]

Juan A. de la Puente[1], José F. Ruiz[1], and Jesús M. González-Barahona[2]

[1] *Departamento de Ingeniería de Sistemas Telemáticos,*
Universidad Politécnica de Madrid
ETSI Telecomunicación, E-28040 Madrid, Spain
[2] *Departamento de Informática, Universidad Carlos III de Madrid*
E-28911 Leganés, Madrid, Spain
E-mail: jpuente@dit.upm.es, jfruiz@dit.upm.es, jgb@computer.org

Abstract. *The fact that most of the GNAT ports are based on non real-time operating systems leads to a reduced usability for developing real-time systems. Otherwise, existing ports over real-time operating systems are excesively complex, since GNAT uses only a reduced set of their functionality, and with a very specific semantic. This paper describes the implementation of a low-level tasking support for the GNAT run-time. In order to achieve a predictable real-time behaviour we have developed a very simple library, built to fit only the GNAT tasking requirements. We have also designed a bare machine kernel which provides the minimum environment needed by the upper layers.*

1. Introduction

The development of GNAT was a decisive step towards the widespread availability of an efficient, high quality compiling environment to Ada programmers. The fact that GNAT is free software is of great interest for researchers, since it allows new developments from existing source code.

Although GNAT provides an effective, high quality compiling environment for Ada 95, its usability for real-time systems development is limited, as most of the GNAT ports are based on non real-time operating systems. Although all GNAT ports implement most of the Annex C and D functionality, many important features, such as true pre-emptive priority scheduling, monotonic time, ceiling locking, and kernel metrics, are not provided as specified in the LRM. As a result, most GNAT implementations cannot be used to program real-time systems with a predictable behaviour.

Looking at GNAT ports over real-time operating systems, we can cite RTEMS[8], a free real-time executive with a POSIX interface and support for multiprocessor systems. But its has been designed for a generic use, and there is a big overhead when using it as low-level support for the GNAT tasking system.

The most common way of implementing GNARL[2] is on top of native threads (usually POSIX threads) for the given architecture. But GNAT tasking implementation is very complete and specific, and when implementing GNARL on top of Pthreads there is a high

1. This work has been partially supported by CICYT under project TIC96-0614.
2. GNu Ada Runtime Library.

overhead motivated by the similar level of abstractions of Ada tasks and Pthreads[2]. Aside from the loss of performance, it increases the complexity, leading to a difficult measuring and bounding of the kernel metrics. In the case of many embedded systems a full-blown implementation of Pthreads is usually considered to be too expensive, and then the existence of a reduced and simple thread support could be of great help.

Therefore, our purpose is to develop a very simple and efficient real-time support for the GNAT tasking system, adapted to its requirements. By not requiring support for the more complex thread features, this approach permits an implementation with very tight efficiency and timing predictability requirements. The library that implements the low level tasking (we call our library JTK from Jose's Tasking Kernel) provides GNARL semantics and is written in Ada[1]. The kernel that interacts with the underlying hardware is written in C, with a small amount of assembly code.

Our intention is to provide a freely available test-bed for experimentation in language, compiler, and run-time support for developers of real-time embedded systems. This is a first implementation to explore the validity of this approach.

The remainder of this paper is organized as follows. Section 2 is a brief overview of the JTK architecture and its integration inside GNARL. Section 3 explains the main implementation details of JTK and how they are implemented. Section 4 shows some performance results to endorse the feasibility of our approach. Section 5 concludes with a summary of assessments, comments and possible future work of this implementation.

2. The architecture of JTK

One of the goals of the GNARL development was to provide an easily portable implementation, by means of a layered design. Each layer provides all of the tasking-related services required by the next higher layer through a procedural interface. The architecture independent components are clearly separated from the machine dependent parts by means of a well defined interface, called GNULLI[2][1].

Therefore, the replacement of the lower-level tasking implementation can be carried out in a very straightforward manner. There are two ways of implementing a new run-time system using this interface:

- A GNULLI interface can be built for an already existing thread library. This is the most common approach to building new non real-time GNAT ports. This is also the way that the RTEMS and VxWorks ports have been built.

  This approach has clear advantages: most real-time operating systems have a POSIX.1c compliant interface, and it is currently well known how to build a GNULLI on top of Pthreads. However, it is not very efficient for two main reasons: First, the Pthread interface is often built as a library which masks the native thread interface for the operating system. And second, the GNAT run-time library already provides a lot of support for tasks, and implementing the high level part of GNARL on top of Pthreads induces abstraction inversion which causes inefficiency in the implementation.
- The other way to build a real-time GNARL is to implement a minimum kernel for task support with a GNULL interface, which does not make use of any thread

1. About 2500 lines of code, not including test programs.
2. GNU Lower-Level Interface.

mechanism from the underlying operating system. We believe that this approach, although harder to implement, leads to more efficient and predictable run-time support. Since GNARL provides most of the functionality needed for tasking, only the missing, lower level functions, have to be implemented, which results in a comparatively small executive.

At the lowest level, we have developed a very simple kernel that offers the functionality that a minimal real-time operating system should provide. It has also been designed to fit Ada tasks semantics as tightly as possible. In order to provide an easily predictable environment, this kernel has been designed with this objective in mind, from hardware up. Therefore it operates in single virtual address space (no paging), it does not have a file system, and the only devices supported are the timer and the serial port. These simplifications eliminate unpredictable time delays due to page faults, waiting for completion of I/O, and I/O completion interrupt processing[3].

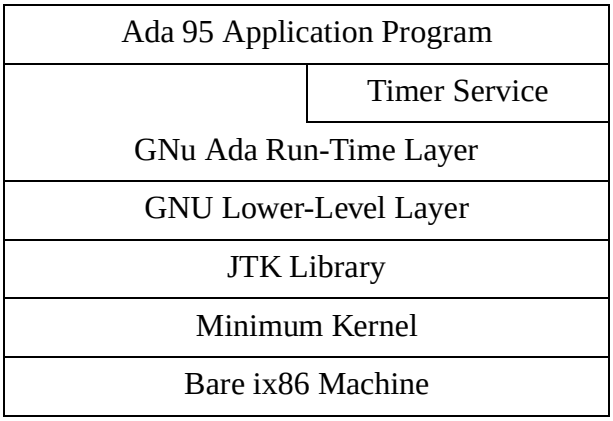

Figure 1: Structure of the developed GNAT tasking system.

JTK offers a priority driven scheduler with pre-emption. It has a FIFO queue for each priority to permit a good scalability. Tasks are run until they are blocked or until another task with a higher priority becomes ready.

It also provides two basic synchronization methods: mutual exclusion and conditional synchronization. For the mutual exclusion two mutex objects are provided: one operating as a simple binary semaphore and other using the Immediate Priority Ceiling Protocol, which offers the possibility for read and write locking. For the conditional synchronization, JTK implements condition variables, used by the timer and protected objects.

The tasks and schedulers have been defined as tagged objects, so there is an easy and clean way of extending the implementation with a new scheduling policy. There are some scheduling schemes in the literature, and none of them can fit all the different kinds of applications. Therefore we have tried to establish a flexible framework to allow the user to implement the scheduling policy that really needs.

3. Implementation

When thinking about developing a real-time application, one of the most difficult aspects is the bounding of the worst-case execution time (WCET). To obtain analytically a tight measurement of this value, two issues are very important, to have access to the source code (application, run-time libraries and operating system) and to work with an imple-

mentation as simple as possible. Since we are working with GNAT the source code is accessible (our libraries and kernel are also free software), and simplicity is the most important issue that we have had in mind in our design.

Signal management is one important issue for improving Pthreads behaviour. GNARL has a very specific way of managing signals[5], similar in many senses to the way used by Pthreads. What happens is that GNARL does almost all the job on its own, and uses the low level support in a minimal mode. Since Pthread does not offer the possibility of a minimalistic use there is a lot of redundant and complex processing that obscure the measurement of the kernel metrics. Our approach is to notify the GNARL layer of the occurrence of the signal and let this layer process the signal.

As for the timer support, we have modified the GNARL layer to issue one timer request at a time (using a timer server task), and therefore the underlying library does not have to deal with more than one simultaneous request. This simplification leads to reducing the overhead involved in managing different queues, and to an easier way of understanding what these layers do. This also makes it easier to bound execution times.

4. Performance

To evaluate the performance of this implementation we have tested two kinds of programs as a first attempt:

1. Two tasks which only perform context switches. The times reported are averages taken over 100 000 iterations of each task.
2. Two tasks executing 100 loops inside which they perform a select statement with a delay alternative. The measurements indicate the amount of processor time that a third task could use normalized to the maximum value obtained. In this way, changing the value of the delay alternative produces results which are of the same amount.

The tests have been executed on the same machine (Pentium II at 233 MHz). The measurements have been taken over three different GNAT implementations:

1. GNAT 3.10p over JTK.
2. GNAT 3.10p over Linux, using the FSU implementation of Pthreads.
3. GNAT 3.10p over DOS, using the FSU implementation of Pthreads.

Context switch time is a very important feature in real-time systems, with a high influence on timing behaviour. The notion of cheap concurrency has been around in Pthreads and related works, which try to offer lightweight processes. Figure 2 shows the results of the first test, from which we can see that there is an increase of 100% of efficiency in context switch time with respect to the Linux implementation, and of 200% compared to the DOS implementation.

One aspect where we have tried to enhance is the time management. The second test is focused on showing a measurement of this improvement. This test has not been applied to the GNAT port that uses Pthreads over DOS because it does not run well with priorities. In fact, DOS implementation has not a true pre-emptive priority scheduling. Figure 3 displays the results of this test, which show that there is an improvement of about 20% in the amount of CPU time which is needed for the execution of delay statements.

Efficiency is certainly an issue, but in real-time applications predictability is the main problem. Figure 3 has not enough resolution to show this aspect, but from the experiments

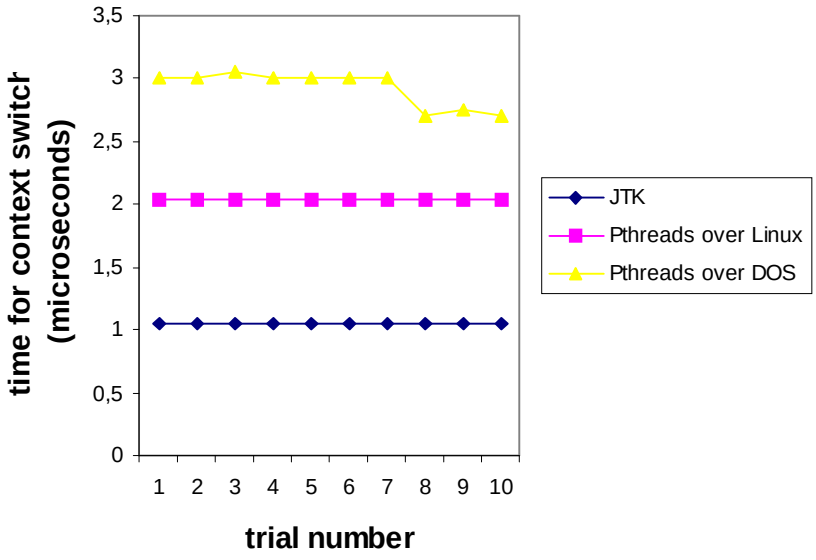

Figure 2: Time taken by a task context switch

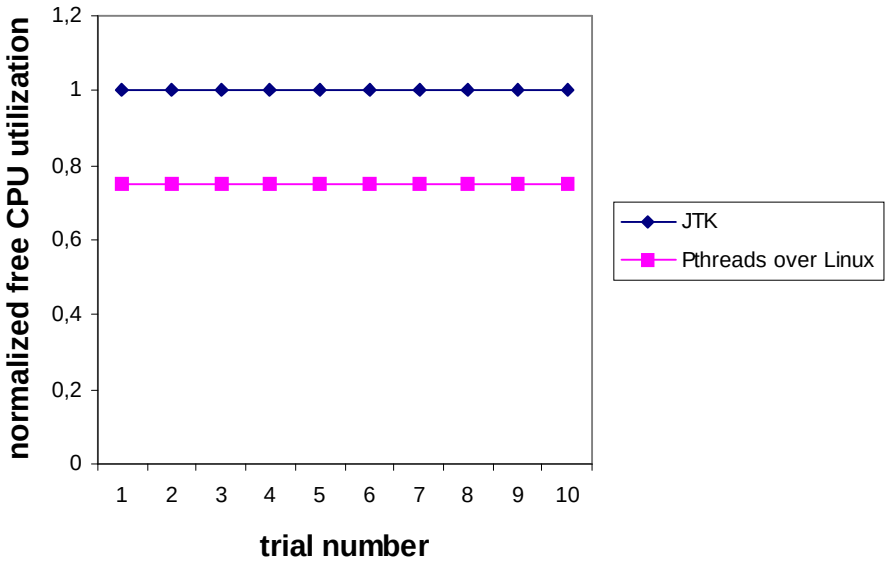

Figure 3: Free CPU when using select statements with a delay alternative

that we have carried out we have measured a difference between the best and the worst trial which is about 350 times bigger on the Linux implementation than on JTK.

5. Conclusion

We have shown that it is possible to significantly improve the predictability and efficiency of tasking in GNAT by designing a simple multitasking kernel which is directly tailored to support the Ada 95 functionality. By eliminating dependence on Pthreads, we have achieved a bare machine implementation with a predictable execution timing. We also have eliminated the very unpredictable delays due to operating system processes. In summary, we believe that true real-time programming in Ada 95 is a real possibility with our approach.

Of course, since the POSIX interface and semantics are not maintained, there is a loss in portability and generality. We believe that is the price that has to be paid for improved determinism and efficiency, which is the primary goal of real-time systems.

The JTK is available as free software at ftp://ftp.dit.upm.es/~str/software/jtk/.

References

1. T.P. Baker and E.W. Giering, GNU Low-Level Interface Definition, Technical Report, Florida State University, June 1993. Available by anonymous ftp from ftp.cs.fsu.edu.
2. E.W. Giering and T.P. Baker, POSIX/Ada Realtime Bindings: Description of Work in Progress, In Proceedings of the Ninth Annual Washington Ada Symposium, ACM, July 1992.
3. T.P. Baker, Frank Mueller and Viresh Rustagi, Experience with a Prototype of the POSIX Minimal Realtime System Profile, Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software, May 1994.
4. Dong-Ik Oh and T.P. Baker, Gnu Ada'95 Tasking Implementation: Real-Time Features and Optimization, In Proceedings of the 1997 Workshop of Languages, Compilers and Tools for Real-Time Systems (LCT-RTS), Las Vegas, Nevada, 1997.
5. Dong-Ik Oh, T.P. Baker and Seung-Jin Moon, The GNARL Implementation of POSIX/Ada Signal Services, In Ada-Europe,96 Proceedings, pages 276-286.
6. T.P. Baker, Dong-Ik Oh and Seung-Jin Moon, Low-Level Ada Tasking Support for GNAT Performance and Portability Improvements, Wadas'96 Proceedings, 1996.
7. Dong-Ik Oh and T.P. Baker, Optimization of Ada'95 Tasking Constructs, Triada'97, St. Louis, Missouri, 1997.
8. On-Line Applications Research Corporation, RTEMS Applications C User's Guide, Sep 1997, http://www.gnatrtems.com.