A “Bare-Machine” Implementation of Ada Multi-tasking Beneath the Linux Kernel

Hongfeng Shen\textsuperscript{1}, Arnaud Charlet\textsuperscript{2}, and T.P. Baker\textsuperscript{1}

\textsuperscript{1} Department of Computer Science, Florida State University
Tallahassee FL, USA 32306-4019
baker@cs.fsu.edu

\textsuperscript{2} Ada Core Technologies, Inc.

Abstract. An Ada tasking kernel is implemented as a layer beneath the Linux operating system. This provides low overhead and precise control of execution timing for real-time Ada tasks, which execute within the Linux kernel address space, while allowing the Linux operating system to execute non-real-time tasks in the background. The Ada tasking kernel is derived from Yodaiken’s Real-Time Linux kernel, with new scheduling and synchronization primitives specifically to support the GNAT Ada runtime system. Implementing the Ada tasking primitives directly on the hardware does not just lower execution overhead and improve control over execution timing; it also opens the door for a simple restricted-tasking runtime system that could be certified for safety-critical applications.

1 Introduction

This paper describes how the GNAT multi-tasking runtime system (GNARL) has been ported to execute directly over the hardware of a generic Intel PC-compatible architecture, as a layer underneath the Linux operating system.

The GNAT Ada compilation system\cite{Gonzalez} has been very successful, and has been ported to many processor architectures and operating systems, but until recently its multi-tasking runtime system (GNARL) has not been implemented on a bare machine. The tasking model of the Ada 95 programming language was intended to permit a light-weight bare-machine implementation, and so was the GNARL. The components of GNARL that are dependent on a particular machine and operating system are isolated by a low-level tasking interface, called GNULLI. Previously, GNULLI has always been implemented as “glue code” to the services of an existing thread library, which in turn is layered over an underlying operating system. The performance of Ada tasking has been limited by the threads library and operating system, which in no case were designed to support Ada, and in most cases were not intended for real-time applications. Thus, it has remained an open question how well the tasking implementation would work if it were supported by a small, simple, and highly predictable implementation of the GNULLI, executing directly on the hardware.
The rest of this paper tells more about this project. Section 2 explains the motivation and background, including RT-Linux. Section 3 explains the design of the implementation. Section 4 reports on the outcomes of functional and performance tests.

2 Background

There are two independent motivations for doing a bare-machine implementation of the GNULLI. The first motivation is to achieve the kind of efficiency and predictable execution timing needed for hard real-time applications. In versions of the GNULLI where Ada tasks are implemented using the concurrent programming primitives of a commercial operating system, the activities of the operating system can compete with the tasks in the application, and cause unpredictable variations in their execution timing. The second motivation is to lay the groundwork for the kind of small, certifiably-safe implementation of a restricted subset of Ada tasking that was proposed at the 1997 International Real-Time Ada Workshop at Ravenscar [3]. A simple bare-machine implementation is needed for this because the added complexity of a full off-the-shelf operating system would be an obstacle to certification.

The target hardware architecture for our bare-machine implementation of GNULLI is a generic PC-compatible machine with an Intel 486/586 processor. This was chosen for its wide availability and low cost.

A tasking kernel by itself is not very useful. Interesting applications do input and output, and that requires device drivers. The complexity of a single device driver can exceed that of a tasking kernel. Moreover, hardware device interfaces are often poorly documented or even secret, and subject to frequent changes. Therefore, development and maintenance of an adequate collection of hardware device drivers is very daunting. One way to avoid putting a lot of new work into device drivers is to reuse the device drivers of an existing operating system, such as DOS or Linux. A problem is that these operating systems were not designed for real-time use, and their device drivers are sources of timing unpredictability. Reuse of non-realtime device drivers is possible, but only if they are run in the background, at low enough priority that they cannot impede the progress of hard-real-time tasks.

Support for background-foreground separation is generally a useful thing, since all but the simplest real-time systems include a combination of hard and soft real-time or non-real-time processing. For example, a system might have hard-real-time periodic tasks with tight jitter constraints to serve sensors and actuators, and soft-real-time or non-real-time tasks to perform actions that have inherently variable time delays, such as communicating over a network and logging data to a disk. In such a situation, the I/O drivers for the sensors and actuators would need to meet hard-real-time constraints, but the I/O drivers for the network and disk I/O might be reused from a non-real-time operating system.