

A Framework for Real-Time Embedded PC Programming

Ian Braithwaite, Nils A. Andersen and Ole Ravn

Technical University of Denmark, Department of Automation,
2800 Kongens Lyngby, Denmark.
idb, naa, or@iau.dtu.dk

Abstract

A framework for developing real-time programs on embedded PC hardware is presented. Programs consist of a number of real-time tasks and a single non real-time task. Program development and testing is assisted by the automatic collection of task execution times and frequencies during system operation.

Tasks have access to device drivers for input/output and can communicate with each other. Additionally, the non real-time task has access to local disk storage and remote hosts via a network.

Flexible non real-time facilities, including an embedded scripting language, provide access to the timing data as they are generated. The same facilities are also used to implement both local and remote user interfaces to the system.

Application areas include control systems for autonomous robots and vision based controllers. Although still under active development, the framework has been used in a number of projects, proving its potential.

1 Introduction

This paper describes an ongoing project to provide a framework for producing real-time programs running on embedded PC hardware. By embedded PC, we mean a standard PC running without a separate operating system, booting directly from a floppy disk, hard disk or over a network. Standard PC hardware provides a wide range of readily available high performance processors, memory modules and add-on interface cards.

Our application area is the implementation of control systems in general, and autonomous, robotic systems in particular. Such systems are designed assuming that software runs with hard real-time deadlines. At the same time, however, the successful development of these systems requires a flexible environment that encourages experimentation.

A drawback of choosing the PC architecture for real-time programming is the complexity of its memory architecture. Layers of cache memory make calculation of worst case execution times, particularly in the presence of interrupts, difficult. In this work we have chosen instead to estimate execution times by measurement. By integrating the measurement process into the system, and making its use automatic, it is hoped that system testing can be used to provide good estimates for use in a schedulability analysis.

The framework is based on the combination of three components which have been brought together in this work. Firstly, a real-time executive provides for fixed priority scheduling of preemptable tasks. Secondly, a collection of library components based on the OSKit [5, 6] provides support for stand-alone operation. Thirdly the TCL [7] language interpreter provides a command line interface and an embedded web server.

By combining these components it is hoped to create an environment that facilitates incremental development and testing of real-time programs. Parts of the system requiring hard real-time guarantees are scheduled by the real-time executive. Simultaneously, real-time performance measures such as task execution times and frequencies are recorded and can be examined on-line using the command line or network facilities provided by the system.

The rest of the paper is organized as follows. Section 2 describes the real-time components of the system, while Section 3 describes the non real-time components. In Section 4 we describe how we plan to combine these parts, and what we hope to achieve. Finally, Section 5 contains some concluding remarks.

2 The real-time executive

This section describes the real-time components of the system. These consist of a priority based task dispatcher and a collection of device drivers providing input and output to real-time tasks.

```

setHandler(int:i, int:p, proc:f)
    Associate task f with interrupt i at priority p.
setPriority(int:p)
    Raise or lower the current task's priority.
softEvent(int:p, proc:f)
    Dispatch a task f with priority p.
exit()
    Exit the current task.

```

Figure 1. The real-time executive API

2.1 Task models

Theoretical analyses of real-time task execution typically use a task model where a set of tasks are described by parameters giving worst-case execution times, blocking times, inter-arrival times and deadlines [3]. Tasks are assigned priorities according to the type of scheduling being used, and a schedulability test can be evaluated to decide whether the task set is feasible.

In contrast, practical systems often provide a different, longer living concept of a task. Instead of running to completion, tasks use synchronization primitives to regulate their execution. Judicious use of such mechanisms allow these systems to be mapped over to the theoretical model and thus analyzed.

2.2 API primitives

A simple real-time Application Program Interface facilitates the automatic collection of real-time performance data. Such data can be used either on-line or off-line to evaluate scheduling feasibility under worst-case conditions.

To allow for direct measurement of task parameters, the theoretical model is used almost directly as a programming model. Tasks can not synchronize with other parts of the system, once started they run to completion. Tasks may only be delayed by preemption by other tasks when prescribed by the respective task priorities.

The API is summarized in Figure 1. All tasks execute in a single memory address space, and with the processor running with full privileges. So, tasks have direct access to hardware and can share memory directly with each other.

In order to protect shared variables, and to avoid extended priority inversion, tasks access shared memory using the immediate priority ceiling protocol. Thus, a task's priority can be temporarily raised while it runs.

Tasks can trigger other, lower priority tasks during their execution. Such task activations are queued by the executive until they become the highest priority active task. A clock task could, for example, schedule a number of other periodic tasks with various frequencies.

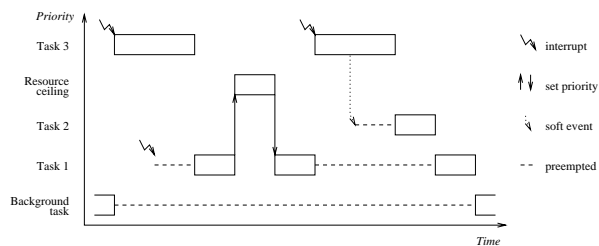


Figure 2. A task execution trace

Real-time activity is triggered by interrupts generated outside the processor. Such interrupts can be generated by periodic timers, external inputs or by devices such as serial communication controllers. Figure 2 shows a possible execution trace for a system. Tasks 1 and 3 are associated with external interrupts. Task 2 is triggered by task3, and task 1 accesses shared memory.

As well as real-time tasks, the executive allows for a single background task to run when no other tasks are active. It has effectively the lowest priority in the system.

2.3 Device drivers

In order to be useful, tasks must be able to interact with the environment. Since our application area is automatic control, we have implemented device drivers for a series of analog and digital I/O cards, as well as a video grabber card.

Analog input signals are typically connected to position, speed and other sensors, while analog outputs are used to control motors and other actuators. The video grabber allows for real-time, full frame rate image analysis.

This selection is particularly suited to vision-based control systems.

3 The embedded PC

The non real-time components provide the rest of the functionality of the system. This includes system startup and user interaction.

3.1 The OSKit

Using a PC as an embedded target requires a large amount of support code to be able to exploit the facilities available. This could include:

- startup and initialization,
- standard library facilities for e.g. string handling and math,
- memory management,

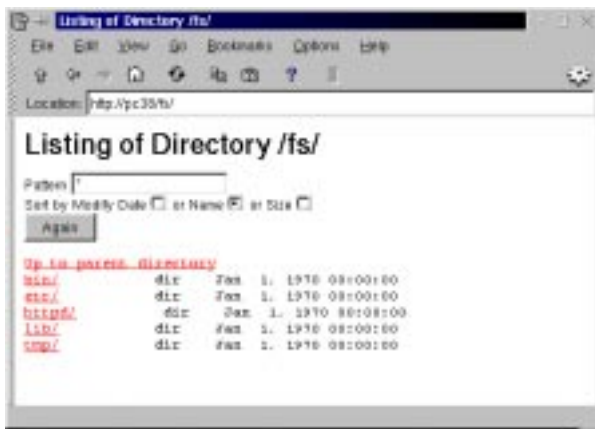


Figure 3. A web page from the embedded web server

- console input/output,
- disk input/output and file system support,
- network interface and protocol stack.

The OSKit project [5, 6] from The Flux Research Group at the Department of Computer Science, University of Utah, provides all of these components. Provided as a collection of libraries, applications are free to pick and choose the functionality they require.

The various components are taken from operating systems such as Linux, FreeBSD and NetBSD. The OSKit team have encapsulated these into an integrated framework allowing, for example, the FreeBSD TCP/IP stack to operate with Linux network drivers.

Since this code is not arbitrarily preemptable, it can not be called from real-time tasks. Thus, this part of the system is restricted to non real-time operation.

3.2 TCL interpreter

John Ousterhout's interpreted scripting language, TCL [7], is designed for embedding in application programs. It simplifies such tasks as providing command line and network interfaces for an application. In addition, the TCL event mechanism provides a simple form of multitasking within a single task.

Utilities written in TCL can be quickly integrated into the embedded PC code. Figure 3 shows an embedded TCL web server page displaying a list of files on the embedded PC. The web server can also generate dynamic web page content. That is, embedded TCL commands in web pages can be used to provide an interface to code on the embedded PC. Thus, dynamic data on the PC can be inspected via

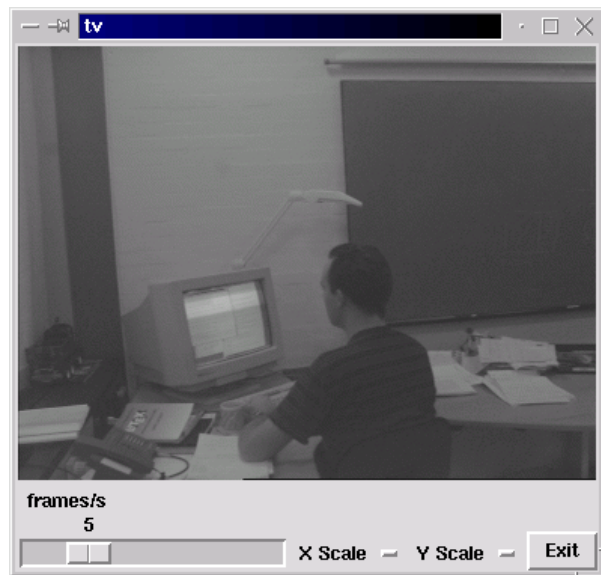


Figure 4. Video captured from the embedded image server

the network, and an interactive web interface can be implemented.

Figure 4 shows a live video feed provided by an embedded TCL server. Creating network servers in TCL requires surprisingly little code. Figure 5 shows the few lines of TCL required to provide the video feed. TCL's event mechanism allows several servers to operate cooperatively with each other.

4 The real-time PC development platform

The combination of components described in the previous two sections is hoped will provide a flexible real-time platform. The rich facilities for data storage via local disks

```

proc connect {chan addr port} {
    fconfigure $chan \
        -blocking off -translation binary
    # Fetch a pixmap from video driver
    video even get data h w
    puts $chan "P5 $w $h 255"
    puts -nonewline $chan $data
    close $chan
}

socket -server connect 3001

```

Figure 5. A network server written in TCL

or the network, together with interactive access allows for monitoring and testing systems as they are developed. The flexible booting arrangements, either locally from a disk or over the network, allow rapid prototyping.

Our user base consists mostly of Masters students, so its educational value is also relevant to us. Often students wish either to gain experience with building embedded systems or to test control engineering theory out in practice.

Seen as an educational tool, the system also allows real-time scheduling theory to be tried out in practice. Most of our students are at least familiar with scheduling concepts, so the interactive facilities provided should help with visualizing and understanding how such ideas can be applied and used.

4.1 Applications

Several projects are already using parts of the system. "GuideBot" [4], an autonomous robot has been operational 8 hours per day at the Experimentarium, a science and technology museum in Copenhagen, since April. An industrial PC is used for control, using data from a line sensor and a laser range scanner to control its two drive motors. "AcroBot" [2] is a highly non-linear double inverted pendulum, which uses several interface card drivers and whose control software has been developed with the embedded PC approach. A project using vision in control is also underway, where a video camera is being used to steer a ball through a maze by tilting the maze in two directions [1].

4.2 Platform development

The real-time executive chosen not only provides for real-time scheduling of a set of tasks, but also for collection of execution statistics.

We are in the process of building a user interface to these data. It will then be possible, for example, to display actual execution traces of the kind shown earlier in Figure 2. This can be used to track down problems or to confirm the expected operation of the system under test.

The data also allow for estimates to be obtained for all the parameters required for investigating worst-case schedulability. The nature of the API allows execution times, blocking times and task frequencies to be measured. Using these, estimates for worst-case completion times can be found, and compared with desired deadlines.

The real-time API is perhaps rather unusual. Certainly it is non-standard, bearing little resemblance to POSIX, for example. Our experience to date has been entirely positive, for our applications at least, it seems to be well suited.

Of course, the interactive possibilities are not restricted to scheduling data. They are also well suited to all sorts of other user interaction that a system might require.

5 Conclusion

A framework for building real-time systems with embedded PC hardware has been presented. Parts of the system have been used in various projects, but not all of the components have yet been used together.

Designing and implementing embedded systems following general software engineering principles often leads to a successful outcome, even where hard real-time performance is required but not specifically checked for with a formal analysis. A priority based scheduler with sensible choices for task priorities is often enough.

The challenge for us is to make schedulability analyses and checks no extra trouble to the developer. Verifying that timing constraints are being met and that all tasks are completing satisfactorily should be so simple, that it becomes a straightforward part of incremental system testing.

While the proposed framework can help with debugging timing related problems in real-time programs, the question of more ordinary debugging remains open for us. The OSKit libraries have support for a remote symbolic debugger which could be used in debugging the non real-time parts of a system. Whether and how such support could be extended to real-time tasks remains unresolved.

Integrating these components can certainly be achieved. The success of the project, however, really depends on whether the proposed model of program development turns out to have advantages. Forthcoming applications, particularly vision-based controllers and mobile robotics will provide test beds for the approach.

References

- [1] N. A. Andersen, O. Ravn, and A. T. Sørensen. Real-time vision based control of servomechanical systems. In *Second International Symposium On Experimental Robotics*, June 1991.
- [2] N. A. Andersen, L. Skovgaard, and O. Ravn. Control of an under actuated unstable nonlinear object. In *Seventh International Symposium On Experimental Robotics*, Dec. 2000.
- [3] A. Burns and A. Wellings. Advanced fixed priority scheduling. In M. Joseph, editor, *Real-time Systems*, chapter 3. Prentice Hall, 1996.
- [4] Experimentarium. GuideBot homepage. http://www.experimentarium.dk/uk/udstillinger/future_body/guidebot.html.
- [5] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The Flux OSKit: A substrate for kernel and language research. In *16th Symposium on Operating Systems Principles*, Saint-Malo, France, Oct. 1997. ACM.
- [6] U. of Utah Flux Research Group. The OSKit version 0.97. <http://www.cs.utah.edu/projects/flux/oskit/>, Jan. 1999.
- [7] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.