VITRAL: A Text Mode Windows Manager for RTEMS

Manuel Coutinho
Instituto Superior Técnico - Universidade Técnica de Lisboa,
DEEC, Av. Rovisco Pais, 1049-001 Lisboa, Portugal,
e-mail: mabeco@comp.ist.utl.pt

José Rufino
Faculdade de Ciências da Universidade de Lisboa,
Campo Grande - Bloco C8, 1749-016 Lisboa, Portugal.
Tel: +351-21-7500254 - Fax: +351-21-7500084
e-mail: ruf@di.fc.ul.pt

Carlos Almeida
Instituto Superior Técnico - Universidade Técnica de Lisboa,
DEEC, AC-Computadores, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
Telefone: +351-21-8418397 FAX: +351-21-8417499
e-mail: cra@comp.ist.utl.pt

Abstract

Most embedded control applications consist of several different tasks that need to be executed in a concurrent fashion and usually have real-time requirements. In some cases, these applications need to interact with the real-world, performing input/output operations through a set of devices such as sensors and actuators, but they may also need to interact with human users. This interaction, if not bounded, may jeopardize the system timeliness. Due to the basic requirements of these applications (concurrent tasks, real-time, input/output event handling), multitasking real-time kernels are a fundamental component to support their development. One example of such a real-time kernel is RTEMS, the Real-Time Executive for Multiprocessor Systems, which is a well-known, real-time multitasking kernel, with a modular architecture, offering interesting characteristics to support the development of real-time embedded control applications. The standard RTEMS visual interface lacks clarity and quality of data presentation to the user.

The VITRAL (Portuguese word for Stained Glass Window) driver is a simple yet reliable multiple text windows manager. It is compatible with standard input/output calls (stdio library) where each window can read from the keyboard and write to the output. To prevent event overload, VITRAL provides a protective mechanism to limit the rate of processed keyboard interruptions,

Keywords

Text mode windows manager, Real-time kernels, RTEMS, Timeliness properties

1. Introduction

Most embedded control applications consist of several different tasks that need to be executed in a concurrent fashion and usually have real-time requirements. In some cases these applications need to interact with the real-world, performing input/output operations through a set of devices such as sensors and actuators, but they may also need to interact with human users.

Due to the basic requirements of these applications (concurrent tasks, real-time, input/output event handling), multitasking real-time kernels are a fundamental component to support their development. One example of such a real-time kernel is RTEMS, the Real-Time Executive for Multiprocessor Systems [1], which is a well-known,

This work was partially supported by FCT through Project POSC/EIA/56041/2004 (DARIO).

real-time multitasking kernel, with a modular architecture, offering interesting characteristics to support the development of real-time embedded control applications. Applications are composed by a set of tasks, whose relative urgency is specified by a priority parameter. Tasks are scheduled preemptively, according to their priority. RTEMS is an open source operating system, that is currently maintained by OAR (On-Line Applications Research Corporation – USA). It is available for several different platforms/architectures, including the Intel IA-32 PC386, which is the one we use in this work.

Standard distributions of most real-time multitasking kernels, RTEMS included, only have a generic console for user interaction. In that console, there is only one window to which all the tasks must perform their output. In situations where there are several different tasks producing output to be read by a human user, a more user friendly interface is desired. With this objective, we developed VITRAL – a simple text mode windows manager for RTEMS.

The VITRAL (Portuguese word for Stained Glass Window) driver is a simple yet reliable multiple text windows manager. It is compatible with standard input/output functions (stdio library) where each window can read from the keyboard and write to the output (Figure 1).

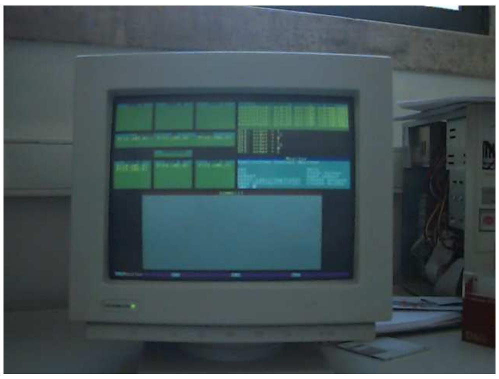

Figure 1. Aspect of VITRAL – a simple windows manager for RTEMS

On the other hand, embedded control systems and applications, needing to interact with the real-world to perform input/output operations (e.g. through sensors/actuators), exhibit in general a set of stringent real-time and dependability requirements. This means the system must be able to preserve timeliness even in the presence of disturbing factors, such as the occurrence of faults or transient overload.

Temporal protection of input/output operations for generic event-based systems has been addressed in [2]. The application and engineering of those mechanisms in RTEMS settings is further described in [3].

The console driver is a standard RTEMS component which calls for the use of input/output temporal protection mechanisms, at least whenever dependability and timeliness are a must. Furthermore, console input/output operations must not jeopardize overall timeliness constraints, thus requiring time bounded functions.

In the console driver, the input is the keyboard and the output the monitor. The keyboard is typically an asynchronous device that produces events when a user presses a key. If the number of events increases dramatically, due to a failure, accident or intentional action provoked by a malicious user, the constant processing of these events may jeopardize application timeliness. In order to prevent this potentially fatal overload, temporal protection mechanisms must be added to the system.

The VITRAL window manager addresses all these issues while minimizing the impact on existing applications. It replaces the original RTEMS console driver, and incorporates a temporal protection mechanism associated with console input. With this mechanism we are able to tolerate overload situations such as those induced by a *stuck key*, without compromising the timeliness of system and application tasks.

2. Related Work

There are many commercial or open-source window managers designed for embedded systems. We highlight the top four most popular technologies: NanoX (formerly MicroWindows) [4, 5], openGUI [6], Qt/Embedded [7] and MiniGUI [8]. All of these systems provide an elaborate graphical environment surpassing VITRAL (with text environment only) in quality of presentation. In embedded control systems, two key characteristics concern the timeliness properties and memory/resource consumption. Of the above managers, only MiniGUI targets real time systems. It has, however, undesirable characteristics during output operations: all of the output messages arrive at a central coordinator task which treats them in a FIFO manner, destroying their relative urgency. These systems take a large portion of its resources. While NanoX, openGUI and MiniGUI take typically 300 KiB, the minimum memory requirements for Qt/Embedded are about 1.5 MiB. VITRAL, being a text mode windows manager, is expected to lower this requirement (70 KiB). One other important aspect in real time systems regards event overload as a potential cause for performance degradation. None of the presented technologies provides protection against interrupt overload from the keyboard/mouse.

3. VITRAL

The VITRAL driver is a simple yet reliable multiple text windows manager. It is completely compatible with standard I/O calls (stdio library).

The real time operating system used, RTEMS, offers a high degree of functionality and extensive reports which aid to its comprehension. In particular, its architecture incorporates the use of drivers to allow a modular and uniform approach to deal with input/output operations, fundamental for reducing the cost of production and maintenance of the developed software. Furthermore, it provides a set of real-time components which allow the use of, for example, dynamic memory allocation and priority inheritance algorithms.

VITRAL is integrated with the RTEMS core in such a way that minimizes the dependency on a specific RTEMS distribution. This is possible because the deployment of VITRAL within RTEMS is performed in a static manner, during a configuration phase. Figure 2 represents the architecture of that integration, presenting VITRAL, the window manager console driver, as an extension to the RTEMS core, to be used by the application.

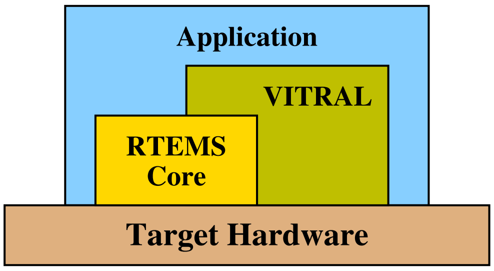

Figure 2. VITRAL architecture within the RTEMS scope

One key point, concerning the integration of VITRAL components, is related to a potential interference of the VITRAL management task with other tasks in the system. This may induce unbounded time delays, due to priority inversion problems, for a given VITRAL management task priority. On the other hand, the priority of this task should be associated to the urgency/importance given in the system to input operations. An elegant solution to this problem may use the dynamic assignment of VITRAL task management priority, e.g. through priority inheritance algorithms [9], available in RTEMS.

4. Engineering of VITRAL on RTEMS

The internal VITRAL architecture contains one task that manages the processing of special keys (*hot-keys*) and the creation of windows. *Hot-keys* are used to select the desired input window and for other functions such as hide/show a window. This is done through a special-purpose VITRAL component, installed in the keyboard interrupt service routine, which performs a test to verify if a *hot-key* was pressed. If so, a message is sent to the VITRAL management task, for processing. Otherwise, the normal key handling is performed.

Each application task, when created, is associated to a default window (*full-screen*), that represents its output window. This allows to an existent application which performs normal input/output operations to be unchanged from the user point of view. After starting the task, it can, at any time, associate itself to another window, performing its output through normal standard input/output library functions, such as printf, which are directed to the correct window. The output is therefore performed in the context of the calling task, unlike MiniGUI, thus allowing higher priority messages to preempt lower priority outputs. The RTEMS standard console performs a busy wait for the keyboard input. In real-time systems this is not desirable. In VITRAL, each window provides a buffer queue to store pressed keys – in order to minimize space, it is possible to define whether each window allows (or not) keyboard reading and the buffer dimension –, eliminating the unnecessary processing and allowing multiplexing of the keys to the correct window.

Components in RTEMS, such as the TCP/IP stack, have the internal tasks priority defined by the application to allow a greater level of configurability. Likewise, it is the application responsibility to establish the VITRAL management task priority. However, since the application can define a low priority (so as not to interfere with higher priority tasks), a priority inversion scenario may occur. For example, a high priority task sends a window creation message to the VITRAL management task and waits for the confirmation. If a middle priority task preempts the VITRAL management task during the window creation, the confirmation will be delayed and thus the high priority task will wait an additional completion time of the medium priority task. Upgrading the VITRAL management task priority to a pre-defined ceiling provides an elegant solution, by taking advantage of the RTEMS functionality, as described in Figure 3.

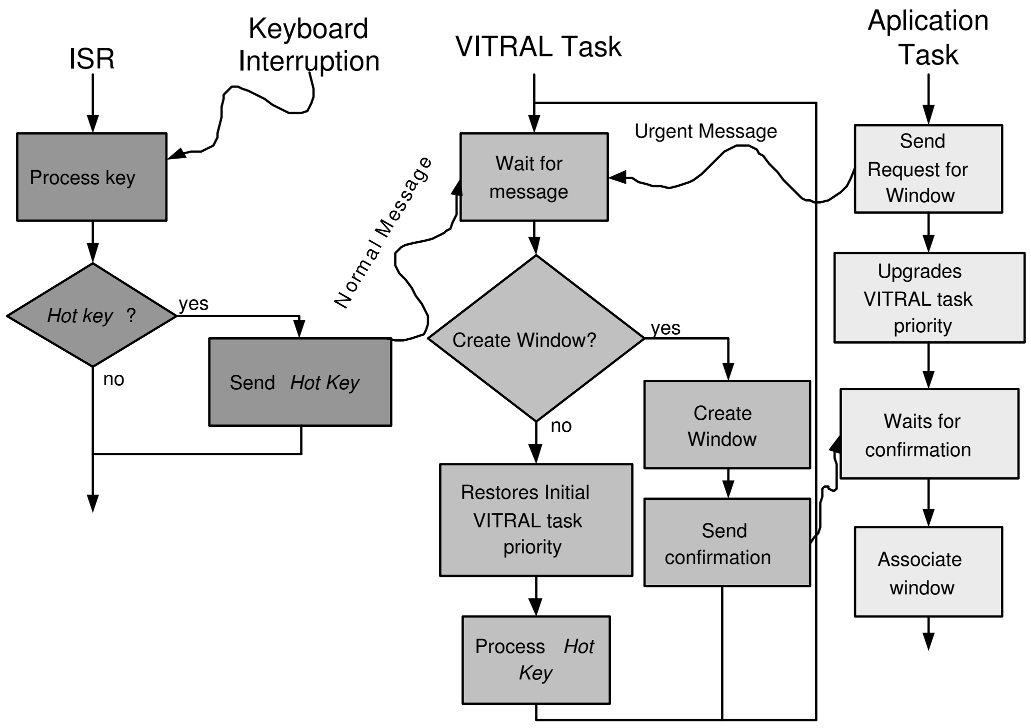

Figure 3. VITRAL functional description – processing of *hot keys* and creation of windows

As denoted on Figure 3, it is assumed that the creation of windows is more urgent than the processing of *hot keys* so as not to make the tasks wait until all the *hot keys* have been processed. Due to the fact that the priority ceiling algorithm is used, there is no higher priority task available to send a message since the VITRAL task has the highest priority. Hence, only two levels of priority are needed to associate to messages (just as many as RTEMS provides). This implementation scheme allows the creation of multiple windows in a concurrent fashion while minimizing the tasks waiting time for their creation.

Case the application needs to treat *hot keys* with greater priority, the VITRAL task should be created with a higher priority and the messages (both window creation and *hot keys*) should also be associated to a priority. The message priority would only order between *hot keys* and window creation messages. Because RTEMS does not possess this level of functionality and due to the low impact that this has on most systems (window creation is typically done during the system initialization only), this added treatment is not contemplated by the current VITRAL version.

Other fundamental issue concerns the integration of temporal protection mechanisms regarding keyboard operation. In order to detect an interrupt overload such as a *stuck key* and preventing it from interfering with the timeliness of system and application tasks, the keyboard interrupt service routine is extended with a component that controls the interrupt rate before calling normal key processing functions. If an overload is detected, keyboard interrupts are disabled and the system switches to a polling mode. A polling task reads periodically the value from the keyboard, determining if a failure is still present. If not, the system returns to the normal state, by enabling again keyboard interrupts.

When choosing the appropriate method to determine a failure, the simplest solution would be to define the minimum interval of time allowed between two consecutive interrupts. Any detection of a smaller value would mean a overload. However, that does not always mean the system is unable to handle overload in a timely manner. This is specially true of very small bursts, which may occur when handling the keyboard and may also apply to a few other devices.

The evaluation, within a given (finite) time interval of relevance, of the occurrence rate of a set of short-duration events can be obtained assuming event occurrence can be represented by a sequence of impulses [10].

This approach allows the use of discrete signal processing methods [10], which we apply to the case under analysis, i.e. the keyboard interrupts. In particular, we use a simple IIR (Infinite Impulse Response) filter.

To demonstrate the ability of this method to cope with both small bursts and overloads, a simple test involving a ”rapid” keyboard user and a deliberate *stuck key* is presented (Figure 4).

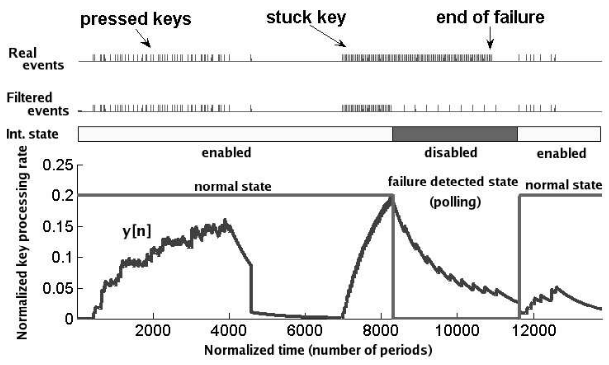

Figure 4. Using a temporal protection mechanism to avoid overload

Let us assume n is the (discrete) progression of real time. In the real system this is implemented through a local clock timer, supporting a time/clock service. Within the first 4000 discrete time periods, a user is pressing keys normally and the interrupt rate ($y[n]$) tends to stabilize in a value below a given threshold. In between $n = 7000$ and $n = 11000$ a *stuck key* is experimentally simulated, resulting in the rise of $y[n]$. One can see that the user is not fast enough to trigger an overload incident, whereas the rhythm of the *stuck key* is more than enough. When the interrupt rate ($y[n]$) goes above the pre-defined threshold, the system detects the overload incident, disabling keyboard interrupts and activating a special-purpose polling system task. When the overload situation ends, the interrupt rate drops below (other) pre-defined threshold, the system enables the interrupts again and returns to normal mode.

This way, the rate of event occurrence in RTEMS (and therefore in the VITRAL window manager) is controlled in order to preserve overall system and application timeliness.

The integration of the VITRAL component took into consideration RTEMS time characteristics. An extensive report on the worst time case scenarios under RTEMS can be found in [11]. Analysing VITRAL, it was found that one of the most important factors, the maximum interrupt disable time, is not exceeded by the VITRAL presence (on the target platform, a PC Intel 386 at 16 MHz, RTEMS establishes 13 μs as the maximum interrupt disable time whereas VITRAL maximum is 3 μs). Due to the automatic association of a task to a default window during its creation, the task creation routine has an overhead of also 3 μs.

With regards to the memory consumption, VITRAL has a maximum of 67 KiB needed for data storage and approximately 5 KiB for source code. As to be expected, these requirements are far less than the graphical environments.

5. Conclusions

In summary, the functionality provided by VITRAL enhances RTEMS in the support for the development of real-time embedded applications. Making sure that RTEMS intrinsic real-time characteristics are not compromised, and are even extended through the use of specific input/output event handling timeliness protection mechanisms, is extremely important for dependable real-time applications.

Although, in this work, the timeliness protection mechanisms are applied to the "console" device, where the interaction is with a human user, which is a "slow" interface, the same type of mechanisms can be used in generic input/output operations, e.g. related to device control or network interfacing, which have a smaller time granularity.

The timeliness protection mechanisms may be applied to any interrupt driven system, being implemented as a separate module, without the need to modify existing drivers. The timeliness protection mechanisms should be integrated in the interrupt processing flow, evaluating timeliness-related parameters (e.g. interrupt rate), before calling the original interrupt service routine.

The ability to deal with such problems and adapt in a dynamic and flexible way is of utmost importance when supporting dependable real-time applications.

The VITRAL window manager provides a RTEMS driver supporting simple multiple text mode windows, compatible with the standard input/output library functions, where each window can read from the keyboard and write to the monitor.

VITRAL includes temporal protection mechanisms bounding the interference in the time domain of window management and console handling, in system and application tasks. This is a significant enhancement of RTEMS functionality concerning a system user friendly interface that nevertheless preserves timeliness guarantees.

Acknowledgments

The authors wish to thank to Pedro Romano for his contribution to the engineering of an earlier version of VITRAL.

References

[1] On-Line Applications Research Corporation (OAR). *RTEMS C User's Guide*, edition 4.6.2, for rtems 4.6.2 edition, August 2003. (The RTEMS Project is hosted at http://www.rtems.com.).

[2] C. Almeida, M. Coutinho, and J. Rufino. Timeliness of input/output event handling in real-time kernels. Technical report, Instituto Superior Tecnico, 2005.

[3] M. Coutinho, J. Rufino, and C. Almeida. Control of event handling timeliness in RTEMS. In *Proceedings of the 17th IASTED International Conference on Paralel and Distributed Computing Systems - PDCS 2005*, Phoenix, Arizona, USA, nov 2005. IASTED.

[4] Gregory Haerr. The microwindows project: Enabling graphical applications on embedded linux systems. http://www.microwindows.org/MicrowindowsPaper.html.

[5] Nano-x programming tutorial. http://www.microwindows.org/Nano-XTutorial.html, 2000.

[6] Opengui home page. http://www.tutok.sk/fastgl/.

[7] trolltech. Qt 3.3 whitepaper. http://www.trolltech.com.

[8] Minigui white paper. http://www.minigui.org, 2004.

[9] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[10] S. Smith. *The Scientist and Engineers Guide to Digital Signal Processing*. California Technical Publishing, San Diego, California, 2nd edition, 1999. Analog Devices Technical Library.

[11] On-Line Applications Research Corporation (OAR). *RTEMS Intel i386 Applications Supplement*, edition 4.6.2, for rtems 4.6.2 edition, August 2003.