EFDA–JET–CP(08)04/09

A.Neto, F. Sartori, F. Piccolo, A. Barbalace, R. Vitelli, H. Fernandes
and JET EFDA contributors

Linux Real Time Framework for Fusion Devices

Linux Real Time Framework for Fusion Devices

A.Neto[1], F. Sartori[2], F. Piccolo[2], A. Barbalace[3], R. Vitelli[4], H. Fernandes[1]
and JET EFDA contributors*

JET-EFDA, Culham Science Centre, OX14 3DB, Abingdon, UK

[1]Associação Euratom-IST, Instituto de Plasmas e Fus„o Nuclear, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
[2]Euratom-UKAEA, Culham Science Centre, Abingdon, Oxon OX14 3DB, United Kingdom
[3]Euratom-ENEA Association, Consorzio RFX, 35127 Padova, Italy
[4]Dipartimento di Informatica, Sistemi e Produzione, Universit‡ di Roma, Tor Vergata,
Via del Politecnico 1-00133, Roma, Italy
** See annex of M.L. Watkins et al, "Overview of JET Results ",*
(Proc. 21 st IAEA Fusion Energy Conference, Chengdu, China (2006)).

Preprint of Paper to be submitted for publication in Proceedings of the
25th Symposium on Fusion Technology, Rostock, Germany
(15th September 2008 - 19th September 2008)

ABSTRACT

A new framework for the development and execution of real time codes is currently being developed and commissioned at JET. The foundations of the system are Linux, the Real Time Application Interface (RTAI) and a wise exploitation of the new i386 multi-core processors technology.

The driving motivation was the need to find a real time operating system for the i386 platform able to satisfy JET Vertical Stabilisation Enhancement project requirements: 50μs cycle time. Even if the initial choice was the VxWorks operating system, it was decided to explore an open source alternative, mostly because of the costs involved in the commercial product.

The work started with the definition of a precise set of requirements and milestones to achieve: Linux distribution and kernel versions to be used for the real-time operating system; complete characterization of the Linux/RTAI real-time capabilities; exploitation of the multi-core technology; implementation of all the required and missing features; commissioning of the system.

Latency and jitter measurements were compared for Linux and RTAI in both user and kernel-space. The best results were attained using the RTAI kernel solution where the time to reschedule a real-time task after an external interrupt is of 2.35 ± 0.35μs. In order to run the real-time codes in the kernel-space, a solution to provide user-space functionalities to the kernel modules had to be designed. This novel work provided the most common functions from the standard C library and transparent interaction with files and sockets to the kernel real-time modules. Kernel C++ support was also tested, further developed and integrated in the framework.

The work has produced very convincing results so far: complete isolation of the processors assigned to real-time from the Linux non real-time activities, high level of stability over several days of benchmarking operations and values well below 3μs for task rescheduling after external interrupt. From being the alternative option, RTAI has been finally chosen as the platform for the project. A first stable version of the framework has been integrated on the JET system and is already being commissioned. It will be soon be used on the Vertical Stabilisation Enhancement for the Plasma Control Upgrade (PCU) project at JET.

1. INTRODUCTION

A Real-Time Operating System (RTOS) is expected to guarantee that a given task will be performed within a defined time interval, providing deterministic behaviour to the real-time applications. To do so a RTOS uses specialised scheduling algorithms and interrupt management schemes that try to minimise the latency and jitter. At the same time it must also provide a set of optimised services, like interprocess communication mechanisms and resource sharing, allowing to develop complex real-time codes.

The EFDA-JET tokamak real-time systems are based on the commercial operating system VxWorks [1, 2 and 3]. It is a Unix like very efficient operating system that provides all the required functionalities: multitasking, shared memory, resource sharing and synchronisation mechanisms. The downside is that, as a commercial project, it has a high cost and royalties associated and it is

closed source. This also implies that system upgrades, like the code compiler, are always dependent on the VxWorks release cycle.

An alternative solution is the open-source Real Time Application Interface (RTAI), an extension to the Linux kernel which adds real-time capabilities to this operating system. RTAI uses its own real-time scheduler and places itself between the hardware and the operating system. This way all the hardware interrupts are trapped and a direct route to the real-time drivers is created. The same services provided by VxWorks are also present in RTAI. There is however a major difference between the two systems, mainly due to the way Linux itself is designed. Linux uses the concept of a user-space level, where user code is supposed to run, as opposed to the kernel-space, expected to do the low level interface to the hardware, processor, file system and memory.

This has some severe implications on a real-time system design if one wants to code in hard real-time: i) no access to the file system or network; ii) only standard C can be used to code in kernel hard real-time. The most common solution to these problems is to use an RTAI extension, named RTAI-LXRT, which provides real-time to user-space. Unfortunately this introduces several micro-seconds delays in the communication with the hardware. Another problem arises from in RTAI-LXRT all the hard real-time sections of the code must be clearly identified between specific tags. This solution is not portable, for large projects is difficult to maintain and doesn't allow to develop generic real-time code or libraries.

2. SUPPORTING SYSTEM

Since RTAI runs over Linux, the first milestone was to find a distribution for the target system with the following requirements: highly customisable, allowing to turn off all the unnecessary services which may compromise real-time; mainstream, guaranteeing the long term support and upgrades; source based distribution highly optimised for the target processor architecture. Gentoo [4] is a source based distribution that fulfils all the requirements and was selected as the distribution for the project. It is supported by a very strong community where users are encouraged to compile everything in the operating system from the source code, an operation that can be easily performed using Gentoo's package manager in a fully automated way, providing a final system which is fined tuned for the target architecture.

The selected kernel version was the 2.6.20.11, compiled selecting only the required features, services and drivers, providing a final binary image of less than two megabytes. In this kind of systems it is vital to have an operating system that only provides the required functionality, minimizing unexpected behaviour and software failures when the hardware is under heavy activity.

The target processors required for the project are based on the generic multi-core x86 architecture, allowing to develop and execute real-time codes in main-stream processors like Intel™ and AMD™. This new family of processors are built with several cores–typically two or four–combined together, allowing the parallel execution of different tasks.

On a multi-core system it can be of great interest to be able to explicitly (overriding the symmetric

multi processor scheduler decisions) assign tasks to a particular CPU. This is a standard Linux feature and can be used to allocate a CPU to all the Linux tasks and to place one or more real-time tasks on the remaining CPUs, which are controlled by the real-time scheduler. The hardware interrupts may also be routed to specific processors.

3. LATENCY AND JITTER TESTS

RTAI is already being successfully used in some fusion laboratories [5, 6] but it was never tested on the multi-core x86 platform. As referred in the previous section a RTOS must guarantee low latency in response to hardware interrupts and a precise scheduling of the real-time tasks. To verify these response times a test was designed [7] in order to: compare the Intel™ and AMD™ architectures; test the response to interrupt latency and jitter; obtain the average rescheduling time; compare the execution of the tasks using the RTAI kernel real-time scheduler and RTAI-LXRT; test the isolation of the real-time processors.

The best results were obtained using the dual-core Intel™, where the typical response for an hardware interrupt was of $2.35 \pm 0.35\mu s$ and the rescheduling of a task after an interrupt of $2.75 \pm 0.35\mu s$. The kernel real-time scheduler clearly outperformed RTAI-LXRT.

The detachment and protection of the real-time processes in specific processors was successfully tested by placing tasks in the real-time processors consuming 100% of the processing power and monitoring the impact on the Linux processor. The same was repeated but this time stressing the Linux processor and following the real-time tasks activity. These tests showed that both spaces are independent with no noticeable impact on each other.

4. REAL-TIME SERVICES

The plasma position and current control codes running at JET[8], like the vertical stabilisation and shape controllers, share a C++ common interface framework library named BaseLib[1]. This framework uses the *code once, run everywhere*, philosophy and allows to reuse and execute the same application in different operating systems and different hardware vendors without rewriting new code. BaseLib was already successfully used in AMD™, Intel™ and PowerPC™. In what concerns the operating systems it was tested in Windows™ NT4, VxWorks, Linux (not real-time) and RTAI.

This is only possible due to BaseLib being divided in several consecutive and independent layers where the bottom layer (named Level0 in figure 1) is programmed specifically for each operating system, providing all the required operating system services (access to files, socket, semaphores, threading, ...) to the rest of the framework. All the other layers provide a large set of functionality, greatly simplifying the development of generic and reusable real-time codes[8].

Unfortunately in kernel-space, where the RTAI hard real-time tasks run, most of the common services that are provided in standard programs are not available, in particular there is no direct access to the file system nor sockets. This was solved by developing a generic communication

module, named fcomm, which makes an extensive usage of Inter Process Communication (IPC) methods, like shared memory and semaphores, delegating the real functionality to the user space, where a series of tasks work in parallel, each handling a different request.

The two main components of fcomm are a kernel-space module and a user-space thread pool waiting for requests (see figure 2). Developers are provided with an application programming interface(API) exactly equal to the one provided by GNU C Library. The fcomm API is used by BaseLib to provide the RTAI interface to the framework.

In the fcomm internals all the available function calls behave in a very similar way: first a copy of the function parameters, passed by the caller, is performed into a shared memory, which returns an unique identifier for each of the parameters; next it searches in small database for the requested function unique identifier and triggers a semaphore that is waiting in the user-space. This semaphore wakes one of the user-space workers to perform the real job. When it finishes, the return value, or an error, is passed through the shared memory mechanism and back again to the calling function. All of these operations are synchronised and protected by a set of auxiliary semaphores, guaranteeing the correct functionality and allowing to execute several calls in parallel from different tasks. All of this is completely transparent to the real-time task developer which only needs to call the function as it would do in user-space. It is obvious that no real-time is guaranteed to a block of code when it is interacting with fcomm, but this is the expected behaviour since no system may guarantee real-time, when hardware input/output operations (in particular the file system) are performed.

The last requirement to be implemented was the ability to execute C++ code in kernel space. This was developed in an independent module that provides most of the C++ features like: new and delete operators; pure virtual functions for inheritance; Run Time Type Identification (RTTI) for a plug-in based and modular system.

5. RTAI IMPROVEMENTS

During the development of the project some software problems related to the handling of PCI interrupts and the floating point unit in 64 bit processors, were found, solved and committed back to the RTAI project. A new type of semaphore was developed, allowing to synchronise a series of tasks in a common waiting condition.

6. FIRST APPLICATION

The real-time framework itself is also being upgraded into a new version named MARTe (Multithreaded Application RealTime executor) and is going to be used to run the vertical stabilisation [9, 10] code. MARTe is a complex system composed of several independent modules with specific functions (see figure 3). The key component is the real-time thread, responsible for guaranteeing the loop cycle time, that in the case of the vertical stabilisation has the target of 50μs. Most of the modules, like the communication with Control and Data Acquisition System (CODAS), the state machine and the communication with the hardware, have already been successfully tested in the

JET real environment. Soon the real magnetic signals will be acquired and commissioned so that the real loop cycle times can be calculated and it can started to be used in parallel with the old vertical stabilisation system. The hardware is based on the Advanced Telecommunications Computing Architecture (ATCA™)[11] with a controller based on the Intel™ Quad-Core Technology.

CONCLUSIONS

BaseLib and MARTe, JET's C++ real-time framework, were further improved and support for RTAI and Linux was added, allowing to reuse the applications that already run on other operating systems and hardware architectures, without changing the original source code. This was achieved by a developing and improving a series of services which take full advantage of the new multi-core technology. While executing the road map some limitations and software problems were found in the RTAI core. These were solved and committed back to the community. The first target application is to use MARTe for the vertical stabilisation, running in a 4 core hardware controller and to achieve a close loop cycle time of 50μs.

ACKNOWLEDGEMENTS

This work, supported by the European Communities under the contract of Association between EURATOM/IST, was carried out within the framework of the European Fusion Development Agreement. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

REFERENCES

[1]. G. De Tommasi, F. Piccolo, A. Pironti, and F. Sartori, A flexible software for real-time control in nuclear fusion experiments, Control Engineering Practice, vol. **14**, Issue 11, Nov. 2006, pp. 1387-1393

[2]. M. Riva, L. Zabeo, E. Joffrin, D. Mazon, D. Moreau, A. Murari, R. Felton, et. al, Real time safety factor profile determination in JET, Fusion Engineering and Design, vol **66-68**, Sep. 2003, pp. 779-784

[3]. R. Felton, E. Joffrin, A. Murari and JET EFDA Contributors, Real-time measurement and control at JET signal processing and physics analysis for diagnostics, Fusion Engineering and Design, vol. **74**, Issues 1-4, Nov. 2005, pp. 769-774

[4]. GK.Thiruvathukal, Gentoo linux: *The next generation of linux*, Computing in Science … Engineering, vol. **6**, Issue 5, Sep. 2004, pp. 66-74

[5]. A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, C. Taliercio, *Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application*, IEEE Transactions on Nuclear Science, vol. **55**, Issue 1, Feb. 2008, pp. 435-439

[6]. C. Centioli, R. Iannone, M. Ledauphin, M. Panella, L. Pangione, S. Podda, et. al., *Using real*

time workshop for rapid and reliable control implementation in the Frascati tokamak upgrade feedback control system running under RTAI-GNU/Linux, Fusion Engineering and Design, vol **74**, Nov. 2005, pp. 593-597

[7]. A. Barbalace, A. Neto, F. Sartori and F. Piccolo, *Realtime Application Interface Characterization on x86 Dual Core Systems Architectures,* submitted to IEEE Computer magazine

[8]. G. De Tommasi, F. Piccolo, A. Pironti, and F. Sartori, "A flexible software for real-time control in nuclear fusion experiments," Control Engineering Practice, vol. **14**, no. 11, Nov. 2006, pp. 1387-1393

[9]. M. Lennholm; D. Campbell; F. Milani; S. Puppin; F. Sartori; B. Tubbing, "*Plasma vertical stabilisation at JET using adaptive gain*",Fusion Engineering and Design, vol 1, Issue 6-10, Oct. 1997, pp. 539-542

[10]. F. Sartori, S. Dorling, A. Horton, P.J. Lomas, M.E.U. Smith and R.C. Stephen, *"n = 2 Compensation and variable gains for JET vertical stabilisation"*, Fusion Eng. Des. **66–68** (2003), pp. 727–734

[11]. A.J.N. Baptista, J. Sousa and C.A.F. Varandas, "*ATCA digital controller hardware for vertical stabilization of plasmas in tokamaks*", Rev. Sci. Instrum, vol. **77**, Issue 10, Oct. 2006.

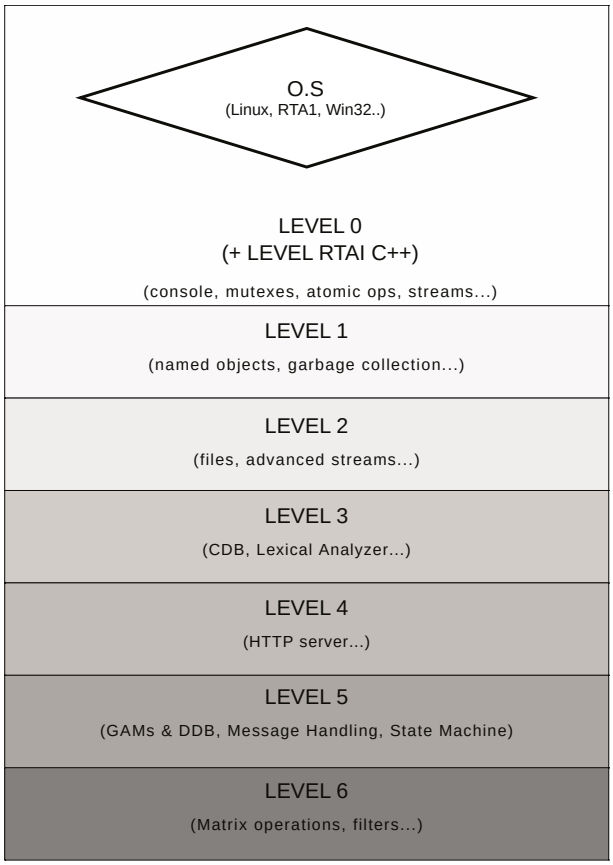

Figure 1: The real-time library is divided in several layers, where Level0, provides the interface to the target operating system. All of the others modules use the operating system functions (files, sockets, string manipulation, etc.) exported by this layer. This scheme allows to port the library into new operating systems without making major modifications in the library and to run applications without changing the source code.

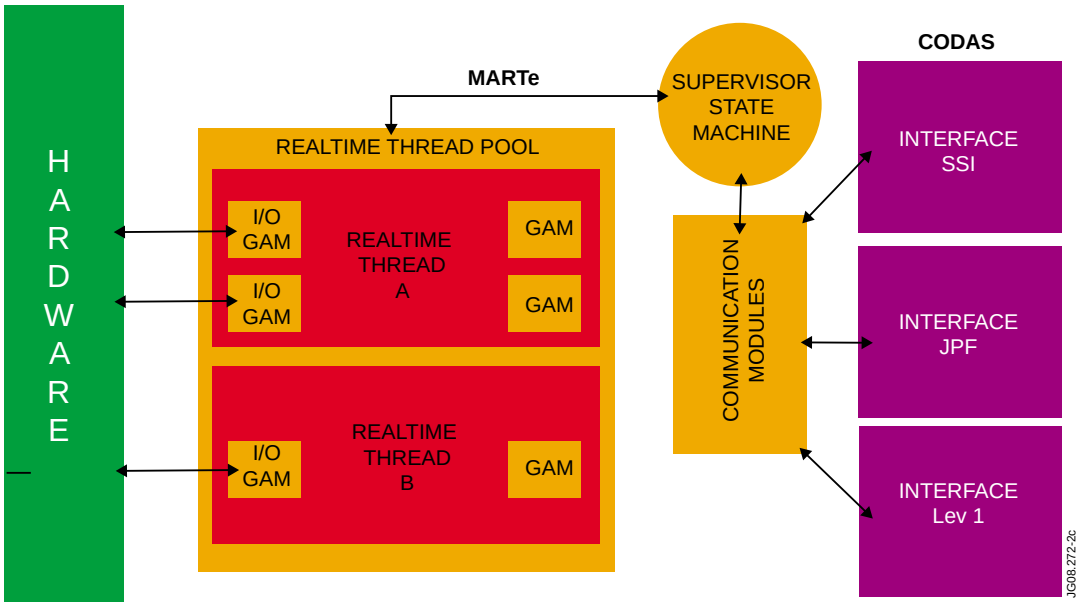

Figure 2: fcomm allows to access services that are normally disabled in the kernel space (file system and sockets in particular) , so that applications using hard real-time codes can still access this kind of services. fcomm is multitasking so that more than one task is able to perform operations without blocking the rest of the system.

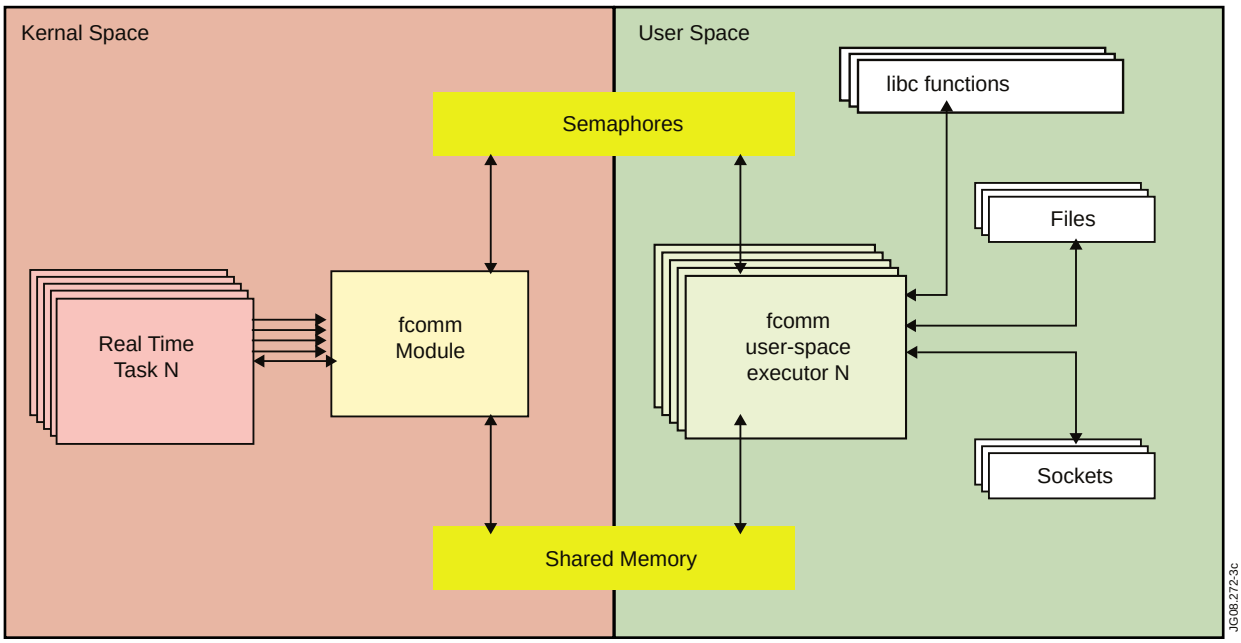

Figure 3: MARTe is the new JET real-time framework. It uses BaseLib as the real-time library and consists in a large set of independent modules, each with a very specific function in the system. The realtime threads are responsible for the scheduling and execution of the Generic Acquisition Modules (GAMs) and also provide the interface between the state machine and the real-time codes.