Comparative Analysis of RTLinux and RTAI

Gahan Saraiya
Institute of Technology
Nirma University
Ahmedabad, India
18mcec10@nirmauni.ac.in

Rushi Trivedi
Institute of Technology
Nirma University
Ahmedabad, India
18mcec08@nirmauni.ac.in

***Abstract*—This paper portrays depth analysis of RTLinux and Real Time Application Interface in context to performance and application analysis. Various performance parameters are proposed and compared in this paper.**

***Index Terms*—RTLinux, Real Time Application Interface, Real Time Operating System**

I. Introduction

Real Time Application Interfaceis a real-time extension for the Linux kernel, which lets users write applications with strict timing constraints for Linux. RTAI consists mainly of two parts:

> An Adeos-based patch to the Linux kernel which introduces a hardware abstraction layer
> A broad variety of services which make lives of real-time programmers easier.

RTAI versions over 3.0 use an Adeos kernel patch, slightly modified in the x86 architecture case, providing additional abstraction and much lessened dependencies on the ”patched” operating system. Adeos is a kernel patch comprising an Interrupt Pipeline where different operating system domains register interrupt handlers. This way, RTAI can transparently take over interrupts while leaving the processing of all others to Linux. Use of Adeos also frees RTAI from patent restrictions caused by RTLinux project.

RTLinux is a hard realtime real-time operating system (RTOS) microkernel that runs the entire Linux operating system as a fully preemptive process. The hard real-time property makes it possible to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines from RTLinuxapplications.

The general idea of Real-time Linux (RTLinux) is that a small real-time kernel runs beneath Linux, meaning that the real-time kernel has a higher priority than the Linux kernel. Real-time tasks are executed by the real-time kernel, and normal Linux programs are allowed to run when no realtime tasks have to be executed. Linux can be considered as the idle task of the real-time scheduler. When this idle task runs, it executes its own scheduler and schedules the normal Linux processes. Since the real-time kernel has a higher priority, a normal Linux process is preempted when a real-time task becomes ready to run and the real-time task is executed immediately.

II. RTAI - Real Time Application Interface

The development of Real Time Application Interface(RTAI) was started just about the year 2000 by Professor Mantegazza at Dipartimento di Ingegneria Aerospaziale in Mailand. Also note that it is a fork from RTLinux. The following architectures are supported by RTAI 5.1 are listed below [1]:

- x86 (with and without FPU and TSC)
- x86_64
- PowerPC
- ARM (StrongARM; ARM7: clps711x-family, Cirrus Logic EP7xxx, CS89712, PXA25x)
- m68k (supporting both MMU and NOMMU cpus)

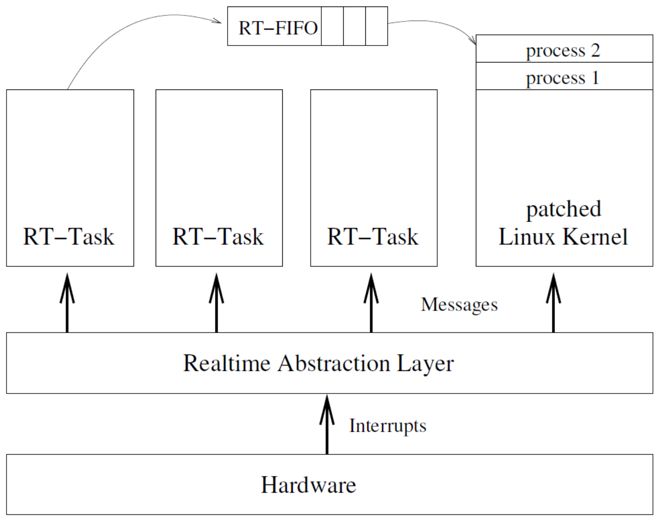

Fig. 1. RTAI Architecture

Strictly speaking, it RTAI not a real time operating system, such as VXworks or QNX. It is based on the Linux kernel, providing the ability to make it fully pre-emptable.

Linux faces lack of real time support. To obtain timing correctness behavior, it is necessary to make some changes in the kernel sources, i.e. in the interrupt handling and scheduling policies.

RTAI provides the same services as of the linux kernel core listed below:

- hardware management layer dealing with event polling or processor/peripheral interrupts

- scheduler classes dealing with process activation, priorities, time slice
- communications means among applications

The kernel components of the RTAI project are under the GNU General Public License (GPL). The User-Space libraries are under the terms of the GNU Lesser General Public License (LGPL). Thus, it is possible to distribute proprietary hard realtime User-Space applications based on RTAI/LXRT and/or RTDM.

RTAI mainly traps the peripherals interrupts and if necessary re-routes them to Linux. It is not an intrusive modification of the kernel; it uses the concept of HAL (hardware abstraction layer) to get information from Linux and to trap some fundamental functions. This HAL provides few dependencies to Linux Kernel. This leads to a simple adaptation in the Linux kernel, an easy RTAI port from version to version of Linux and an easier use of other operating systems instead of RTAI. RTAI considers Linux as a background task running when no real time activity occurs.

RTAI offers a testsuite called "Showroom", which covers almost every feature available in RTAI. It is very useful for testing the proper functionality of RTAI and can be also used for basic measurements, e.g. scheduling latencies, under different circumstances. In order to dispatch external events in

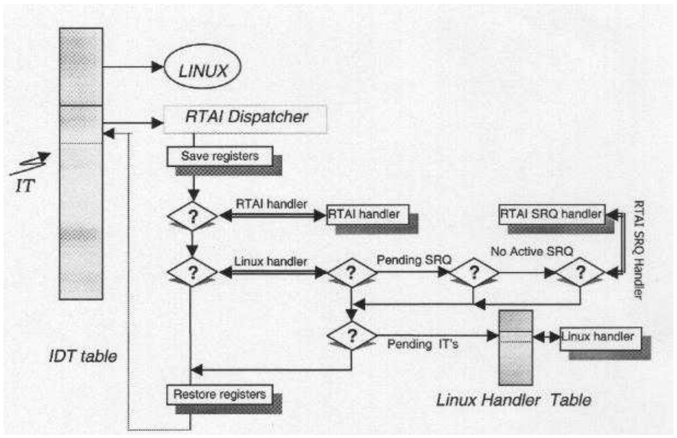

Fig. 2. RTAI Interrupt Dispatcher [2]

a prioritized manner, Adeos offers the possibility for stalling events. That means, during interrupt handling in stage "n", domain "n" can block interrupt propagation for all stages "$k, k > n$", providing domain "0" is the one owning the highest priority. Interrupts are simply stored in a per-CPU log file and are dispatched after calling a special Adeos unstall routine. As a consequence of this, a standard Linux kernel blocking interrupts for performing critical sections, does not influence realtime tasks embodied in a domain lying ahead of it in the interrupt pipeline.

A. Scheduling

The scheduler is the heart of RTAI, it provides trough a series of mechanisms the real-time capabilities which are peculiar to this project. By using the RTAI scheduler the process can meet hard real time constraints and being able to run deterministically, that means that the process can be executed precisely as designed and not limited by the general purpose GNU/Linux scheduler. A real-time process which links to RTAI can therefore used to control complex applications, such as numerical control, industrial process and any complex task which requires "correct" scheduling.

RTAI provides symmetric hard real time services inter/intra user/kernel space. Such a support comes through two schedulers, that at the moment are called $rtai\_lxrt$ and $rtai\_sched$. They can operate in both user and kernel mode and they differ only in relation to the objects they can schedule. That means that $rtai_lxrt$is simply a GNU/Linux co-scheduler, this means that it supports hard real time for all Linux schedulable objects like processes/threads/kthreads. The $rtai\_sched$ instead supports not only hard real time for all Linux schedulable objects, like processes/threads/kthreads, but also for RTAI own kernel tasks, which are very light kernel space only schedulable objects proper to RTAI.

Note that the big advantage of RTAI's light kernel tasks is their **fast switching time** with respect to Linux kernel threads, but on the other side it's important to know that they operate outside any Linux environment. This behavior requires some special care if one needs to inter-operate with Linux.

III. Performance Analysis

In order to draw conclusions about the performance, parameters have to be defined which are important in a realtime environment. Basically, this paper deals with external interrupt latencies and scheduling latencies. Other parameters which can give information about the quality of real time capabilities are for example context switch times, process dispatch latencies or the time that is needed to allocate a certain amount of memory.

A. Parameters for performance measurement

Before discussing the trade-off of both Real Time Operating System first of all it is necessary to describe the parameters and their affect on measurement of Real Time Operating System.

1) Interrupt latency
   In general interrupt latency is also known as interrupt response time, is the *length of time that it takes for a computer interrupt to be acted on after it has been generated*. In most computers, a trade-off exists among interrupt latency, throughput, and processor utilization. The measurement of the response time to external interrupts gives a good idea about realtime capabilities.
2) Scheduling latency
   In general, Scheduling latency refers to that *time from when a task is ready to run to when it is actually gets CPU time*.

B. Interrupt latency

The dominant single-kernel solution is RTLinux, based on $PREEMPT.RT$. $PREEMPT.RT$ reworks the kernel's "spinlock" locking primitives to maximize the preemptible sections inside the Linux kernel. ($PREEMPT.RT$ was originally known as the Sleeping Spinlocks Patch).

Instead of running interrupt handlers in hard interrupt context, $PREEMPT.RT$ runs them in kernel threads. When interrupt arrives, instead of running interrupt handler, just wake up the corresponding kernel thread which runs the handler. Advantage of this method is – kernel thread becomes interruptible and shows up in process list with PID (process id).

IV. Conclusion

The aim of the reported work has been the performance evaluation of Real Time Linux and RTAI(Real Time Application Interface), for this evaluation we have started with basics of both RTLinux and RTAI, in this basic discussion the working procedure of both the real time systems are included. Section III includes the performance analysis focusing on mainly two performance parameters – interrupt latency and scheduling latency.

It is concluded that with dual kernel approach in RTAI compared to RTLinux offer lower latency however the differences are not as great as claimed in real world scenarios and RTLinux is easier to develop for and maintain.

References

[1] Rtai - real time application interface official website. [Online]. Available: https://www.rtai.org/?Homepage

[2] P. Mourot. (2006, Jan) Rtai internals presentation, alcatel france. [Online]. Available: https://www.rtai.org/userfiles/documentation/documents/history.pdf