Study and Comparison of the RTHAL-based and ADEOS-based RTAI Real-time Solutions for Linux

Guoyin Zhang, Luyuan Chen, and Aihong Yao
School of Computer Science and Technology, Harbin Engineering University
Harbin, 150001, China
zhangguoyin@hrbeu.edu.cn; cly_here@126.com; yaoaihong@hrbeu.edu.cn

Abstract

This paper tries to explain the rationales of two different key infrastructures of Real Time Abstraction Interface (RTAI): Real Time Hardware Abstraction Layer (RTHAL) and Adaptive Domain Environment for Operating Systems (ADEOS), and also makes a comparison with measurement to help understanding the difference between them. Our study shows that ADEOS as a virtual resource layer can intercept hardware interrupts like RTHAL which is essentially just a data structure. And through measurement, we found that both the RTHAL-based RTAI and the ADEOS-based RTAI offer maximum scheduling latencies in microseconds, while latencies of the latter are all slightly greater than those of the former under different situations. However, these differences are so trivial that ADEOS can be competent to replace RTHAL for RTAI. Moreover, ADEOS can help RTAI solve the patent problem with RTLinux, so it has already replaced RTHAL since RTAI 3.0 actually.

1. Introduction

As a general-purpose operating system, Linux has become a powerful and mature operating system since its birth. However, to apply Linux in some special fields such as embedded application, some necessary modifications are needed. And because lots of embedded devices demand hard real-time performance for interaction with outside environments, Linux has to be adapted to become hard real-time. The open source essence of Linux and its related projects allows people to look into the kernel, and thus makes the proper modifications possible. Compared to other expensive commercial real-time operating systems such as VxWorks, pSoS and so on, the modified real-time Linux can not only remarkably reduce the cost of embedded development, but can also provide good compatibility and portability [1].

2. The real-time features of Linux and its limitations

For hard real-time applications, operating system must ensure the given task should complete in the appointed time, and the result output time can absolutely not exceed the required deadline. From version 1.3, Linux kernel begins to be compatible with the real-time specification of POSIX 3.1b, introduces the concept of real-time process, and provides other methods such as memory lock, POSIX signal and so on [2]. However, to support hard real-time application, Linux still have some limitations as follows:

First, Linux kernel is non-preemptible, which means a process with higher priority has to wait for the return of system call of another process with lower priority. It is obviously unacceptable for hard real-time requirement.

Second, Linux masks interrupts on operation on critical area, which impairs the system ability to respond timely to the outside environment.

Third, Linux lacks a timer with fine granularity [3]. Standard Linux timers are triggered by a periodic tick interrupt, which on x86 machines is generated by the programmable interval timer (PIT) and has a period of 10 ms (in kernel 2.6, it changes to be 1 ms). Hence, Linux is not suitable for those real-time applications that need time resolution in microseconds.

In addition, the usage of virtual memory in Linux can also incur some response latency.

3. The real-time solutions

Generally, there exist two different approaches to make Linux real-time at present [4]. One approach intrusively modifies the standard Linux kernel for preemptibility, latency, timing, and scheduling, such as Ingo Molnar's preemption patch [5]. Another approach uses a separate real-time kernel that runs Linux as the lowest priority task, such as RTLinux [6] and RTAI [7]. Each approach remarkably improves the real-time performance of Linux, but no single approach offers an all-inclusive set of real-time capabilities. And in this

Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)
0-7695-2581-4/06 $20.00 © 2006 IEEE

paper, we focus on discussion about the latter approach, or rather, the RTAI solution.

RTAI was initially developed by The Dipartimento di Ingeneria Aerospaziale Politecnico di Milano (DIAPM) as a variant of the New Mexico Institute of Technology's (NMT) RTLinux, at a time when neither floating point support nor periodic mode scheduling were provided by RTLinux. Since then, RTAI has added many new features without compromising performance, one of which is the famous RTHAL [8].

Unfortunately, a patent on the design concept of RTLinux brings the open source project RTAI some potential problems [9]. Hence, the RTAI project has been working to replace RTHAL with ADEOS since version 3.0 to be free of the RTLinux patent.

In this paper, we try to illustrate the rationales of the RTHAL-based and the ADEOS-based RTAI solutions, and then make a comparison with measurement between them.

4. The RTHAL-based RTAI solution

RTAI is basically an interrupt dispatcher [10]. As far as Intel architecture is concerned, the processor interrupts (from 0 to 31) are still managed by Linux. RTAI mainly intercepts the peripherals ISA interrupts, and re-routes them to Linux (e.g. disk interrupt) if necessary (see Figure 1).

RTHAL is a structure holding few important functions and data, whose container generally includes:

{

Pointer to IDT

Functions to enable/disable processor interrupts (e.g. cli, sti & flags)

Functions to mask/unmask interrupt controller (e.g. 8259)

Functions to manage SMP machine HW (APIC)

Data descriptor for interrupts (status, handler, nested level …)

…

}

At the beginning, Linux initializes the RTHAL with functions pointing to the native ones. When RTAI becomes active, it saves the function pointers and changes them to point to RTAI own ones.

As a result, with RTHAL between RTAI and Linux, RTAI is able to capture and redirect the interrupts. When an interrupt occurs, RTAI intercepts the interrupt and decides what to dispatch. If there is a real time handler for the interrupt, the appropriate handler is invoked. If there is no real time interrupt handler, or if the handler indicates that it wants to share the interrupt with Linux, then the interrupt is marked as pending. If Linux has requested that interrupts be enabled, any pending interrupts are enabled, and the appropriate Linux interrupt handler invoked, with hardware interrupts re-enabled. Regardless of the state of Linux (e.g. running in kernel mode; running a user process; disabling or enabling interrupts), RTAI is always able to respond to an interrupt. In this way, RTAI is able to preempt the Linux kernel and makes the whole system hard real-time.

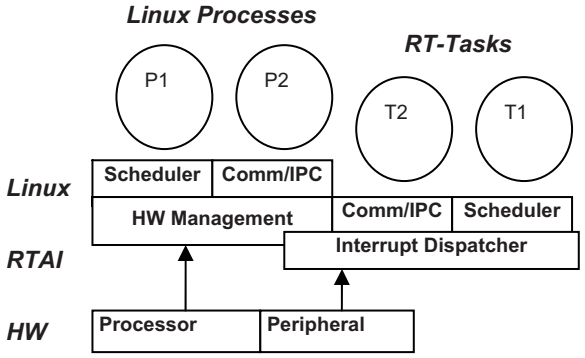

Figure 1. The RTHAL-based RTAI Architecture

From a real-time point of view, it is quite similar to RTLinux, whereas RTLinux is an intrusive modification of Linux kernel. RTAI uses the concept of RTHAL instead to get information from Linux and to trap some fundamental functions, yet with much fewer modifications to kernel. Below are some sample codes of modifications to kernel:

```
if (!(status & IRQ_DISABLED))
// original code
// enable_8259A_irq(irq);
// new code using the RTHAL function
rthal.unmask_8259A_irq(irq);
```

However, RTAI is more than an interrupt dispatcher, but rather a real-time micro-kernel. RTAI's full feature set can be broken down into a set of basic services such as the schedulers, FIFOs, and shared memory, and a set of advanced features such as POSIX and dynamic memory allocation.

Both basic and advanced services are provided via kernel modules, which can be loaded and unloaded using the standard Linux *insmod* and *rmmod* commands. Although the *rtai* module is required every time when any real-time service is needed, all other modules are necessary only when their associated real-time services are desired.

5. The ADEOS-based RTAI solution

ADEOS is a resource virtualization layer available as a Linux kernel patch, which general design has been proposed by Karim Yaghmour in a technical paper, back in 2001 [11].

The current incarnation of this proposal makes it a simple, yet efficient real-time system enabler, providing a mean to run a regular GNU/Linux environment and a RTOS, side by side on the same hardware.

To this end, ADEOS enables multiple entities called domains to exist simultaneously on the same machine. These domains do not necessarily see each other, but all of them see ADEOS. However, all domains are likely to compete for processing external events (e.g. interrupts) or internal ones (e.g. traps, exceptions), according to the system-wide priority they have been given [12]. As far as RTAI and Linux are concerned, they are client domains of ADEOS, Figure 2 shows the ADEOS-based RTAI architecture.

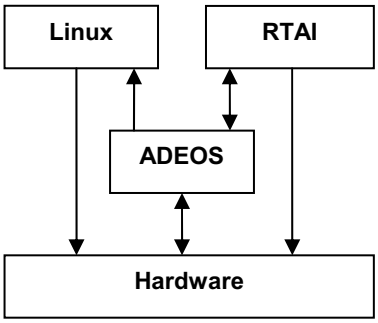

Figure 2. The ADEOS-based RTAI Architecture

5.1. The event pipeline

To share the hardware among different domains, ADEOS implements an event pipeline. Every domain has an entry in the pipeline. Each event that comes in the pipeline is passed on to every domain in an orderly manner according to their respective priority in the system. In fact, all active domains are queued according to their respective priority, forming the pipeline abstraction used by ADEOS to make the events flow, from the highest to the lowest priority domain. Incoming events are pushed to the head of the pipeline (i.e. to the highest priority domain) and progress down to its tail (i.e. to the lowest priority domain). Hence, it is possible to provide for timely and predictable delivery of such events.

As far as RTAI and Linux are concerned, RTAI is the domain with highest priority, so it's always the first to intercept interrupts and be able to preempt Linux. The pipeline abstraction can be illustrated with Figure 3. And as we can see in Figure 3, Linux kernel still has a special role since it stands for the root domain, and the RTAI domain needs Linux to install it by mean of loading the kernel modules which embody it.

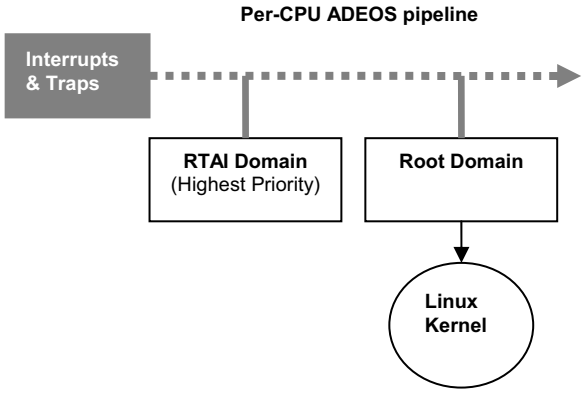

Figure 3. ADEOS Event Pipeline with RTAI and Linux

5.2. Optimistic interrupt protection

In order to dispatch interrupts in a prioritized manner, while still allowing domains to create interrupt free sections, ADEOS implements the so-called o*ptimistic interrupt protection* scheme.

The stage of the pipeline occupied by any given domain can be stalled, which means that the next incoming interrupt will not be delivered to the domain's handler, and will be prevented from flowing down to the lowest priority domain(s) in the same move. While a stage is stalled, pending interrupts accumulate in the domain's interrupt log, and eventually get played when the stage is unstalled. Domains use this feature to protect their own critical sections from unwanted preemption by their own interrupt handlers. However, thanks to the virtualization of the interrupt control ADEOS brings, a higher priority domain can still receive interrupts, and eventually preempt any lower priority domain.

Practically, this means that, although the Linux kernel regularly stalls its own stage to perform critical operations, RTAI running ahead of it in the pipeline would still be able to receive interrupts any time, with no incurred delay.

5.3. System event propagation

Interrupts are not the only kind of events which can flow through the pipeline abstraction, internal events triggered by the Linux kernel itself or system calls issued by Linux applications can also get notified to any interested domains through the pipeline.

5.4. Generic API

Additionally to its straightforward virtualization capabilities, another key advantage of ADEOS relies in its ability to export a generic API to client domains, which does not depend on the CPU architecture.

As far as Linux and RTAI are concerned, Linux calls *adeos_register_domain()* early during system startup. When RTAI is being mounted, it is added as a client domain to the event pipeline. At first, it registers to ADEOS by *adeos_register_domain()*, and then calls *adeos_virtualize_irq()* for all the IRQs that RTAI wishes to be notified about in the pipeline. At last, RTAI uses *adeos_renice_domain()* to get the highest priority in the pipeline.

6. Performance Measurement

To check and compare the real-time performance of the two different solutions, we built two Linux kernels which separately applied the ADEOS-based patch and the RTHAL-based patch, and use the test programs in rtai-3.0\rtai-testsuite\kern\latency, which can measure the scheduling latency. Below list the test details and results.

Hardware environments: an x86 PC with Pentium 930 MHz CPU and 256M SDRAM.

Software environments are listed in Table 1.

Table 1. Software Environments

Name	Description
Linux with ADEOS	Red Hat Linux 8.0 with a 2.4.20 kernel patched by hal10-2.4.20.patch
Linux with RTHAL	Red Hat Linux 8.0 with a 2.4.20 kernel patched by legacy-2.4.20.patch
RTAI	RTAI-3.0
Test Program	rtai-3.0\rtai-testsuite\kern\latency
Disk Load	while "true"; do ls -lR / >list; done
Network Load	ping -f localhost
Idle	no special loads for Linux

The results (all in nanoseconds) are listed in Table 2 where both the RTHAL-based RTAI and the ADEOS-based RTAI offer maximum scheduling latencies in microseconds under different situations, which means both solutions are able to make Linux hard real-time. However, we also can see that latencies of the latter are all slightly greater than the corresponding ones of the former. The cause inside is that ADEOS is a resource virtualization layer from which RTAI call services, while RTHAL is a structure exposed to RTAI and thus can be changed directly by RTAI.

Table 2. Test Results (all in nanoseconds)

	Idle	Disk Load	Network Load
RTHAL-RTAI	7 559	9 129	7 920
ADEOS-RTAI	8 257	11 311	8 709

7. Comparison of RTHAL and ADEOS

As the infrastructures of RTAI, both RTHAL and ADEOS are able to intercept the incoming interrupts before Linux kernel has opportunity to notice them, which means they all can help RTAI to be the only master of hardware.

RTHAL is simply a structure holding few important functions and data that can be saved and changed when RTAI is installed as a kernel module, and restored when RTAI gets uninstalled, while ADEOS is more complex as a resource virtualization layer, which provides services by exporting a generic API to client domains such as RTAI.

With this structural difference, the ADEOS-based RTAI has slightly greater maximum scheduling latencies than those of the RTHAL-based RTAI under different situations. However, the difference is so trivial that ADEOS can be competent to replace RTHAL functionally for RTAI.

And as mentioned early, RTHAL has some potential conflicts with RTLinux patent, while ADEOS is another active open source project which makes RTAI free of that problem, so ADEOS has already become the replacement of RTHAL since RTAI 3.0.

8. Conclusions

In this paper, we have discussed the real-time features of Linux and its limitations, and have also tried to explain the rationales of the RTHAL-based and the ADEOS-based RTAI solutions. At the end of this paper, a comparison with measurement has been made, which can help people to know more about the differences between the two solutions just mentioned.

As Linux 2.6 has improved its preemptibility, it's beneficial to integrate the existing RTAI solution with the improving kernel, which can enhance the real-time performance of processes in user space. Actually, a project named as Xenomai [13] is on the way, and this is what we will make some studies on in the future. And moreover, based on studies on these solutions, making proper migration of them to some embedded Linux such as uCLinux is our eventual target.

References

[1] J. S. Qi, D. W. Cui, and X.H. Hei, “Research and implementation of hard real-time performance of embedded Linux,” *Computer Applications*, Vol. 23, No. 6, June, 2003, pp. 34-35.

[2] X. Gao and Y. Lu, “Research of modifications and implementing of real-time support of Linux,” *Computer Engineering and Applications*, Vol. 41, No. 20, 2005, pp. 102-104.

[3] L. Abeniy et al., "A measurement-based analysis of the real-time performance of Linux,” *Real-Time and Embedded Technology and Applications Symposium*, Proceedings, Eighth IEEE, Sept, 2002.

[4] T. J. Zuo, Y. Y Zuo, and P. Chen, “The analysis of extensions for Linux to a real-time operating system,” *Computer Science*, Vol. 31, No. 5, 2004, pp. 110-111.

[5] Ingo Molnar's preemption patch home page, http://kerneltraffic.org/kernel-traffic/quotes/Ingo_Molnar.html

[6] RTLinux home page, http://www.fsmlabs.com/

[7] RTAI home page, http://www.rtai.org

[8] P. Mantegazza et al., “RTAI: real time application interface,” *Linux Journal*, 72, 2000.

[9] E. Moglen, “RTAI and the RT-Linux patent,” http://www.aero.polimi.it/~rtai/documentation/articles/moglen.html.

[10] P. Mourot, “RTAI Internals Presentation,” http://www.aero.polimi.it/~rtai/documentation/articles/patric_mourot-rtai_internal_presentation.html.

[11] K. Yaghmour, “Adaptive domain environment for operating systems,” ftp://ftp.opersys.com/pub/Adeos/adeos.ps.

[12] P. Gerum, “Life with Adeos,” http://download.gna.org/xenomai/documentation/tags/v2.0.1/pdf/Life-with-Adeos.pdf.

[13] Xenomai home page, http://www.nongnu.org/xenomai/.