2008 14th IEEE International Conference on Parallel and Distributed Systems

Real-time Enhancements for Embedded Linux

N Vun, H F Hor, J W Chao
School of Computer Engineering
Nanyang Technology University
Singapore

Abstract

With the popularity of using Linux for embedded systems, its real-time performance is increasingly becoming an importance issue for applications that require short latency and task execution predictability as commonly encountered in many embedded systems. This paper presents a survey of the approaches used by commercial vendors and open source community to enhance the real-time performance of the Linux Kernel. It outlines some of the main factors that affect the real-time performance of the Linux kernel and describes the two general approaches that are used to improve the real-time performance of the kernel, follows by a comparative study of the various real-time enhanced versions of the Linux that are created for embedded systems.

I. INTRODUCTION

Based on the embedded market studies conducted by the Embedded Systems Design magazine for the three years period from 2006 to 2008, the most important factor that is considered when choosing an embedded operating system has consistently been found to be the real-time performance[1]. With the Linux kernel now available for many types of embedded processor family like ARM, PowerPC, SPARC, MIPS, Motorola 68K and Hitachi SuperH, Linux is increasingly being used in the embedded computing domain. The mainline Linux kernel is well-known for its stability and resilient against fault-prone user applications, and hence provide a very good uptime in many real-world deployments. However, for the kernel version 2.4, its lack of real-time features and unpredictable latency limits its usability for embedded system where real-time requirements is important. Some of the features that are of important for real-time operating system but are not found with the 2.4 kernel includes the followings.

a) Determinism: A truly deterministic OS will have the same average and worst-case timing. However, for 2.4 kernel, the difference between its average and worst-case timing is nowhere near that of a practical real-time OS. This is not surprising since the original Linux was designed to be used as a general purpose operating system where throughput is the more important factor than the deterministic behavior that is considered in a real-time system.

b) Interrupt Latency: While the interrupt latency of a computing system depends very much on its hardware platform (e.g. processor speed), with all things being equal, typical real-time operating system (RTOS) will outperform Linux in the area of interrupt latency. Interrupt handler can be delayed by the preemption of a new interrupt arise; interrupt disabled by the kernel code; or preempted by other interrupt handling mode, e.g. fast interrupt mode in Linux. This has resulted in the inability of a Linux system to respond quickly (in microseconds) to an incoming interrupt. The longer the interrupts are disabled in the kernel, the longer will be the delay in servicing them.

c) Preemptive Kernel: A preemptive Kernel allows the scheduler to switch a higher priority task to take over from a lower priority task, which usually also leads to better deterministic response. While the 2.4 Linux kernel has introduced certain level of preemptive to provide a somewhat soft real-time response, it is still not sufficient to reduce the latency that are expected of a typical RTOS used for embedded system. On SMP systems, the Linux kernel also uses locks and semaphores that further causes delay in switching to a high priority task that is ready to run.

With Linux increasingly being adopted by the embedded industries, more stringent real-time specifications are now required of the kernel by standard bodies like CGL (Carrier Grade Linux[2]) and CELF (Consumer Electronics Linux Forum[3]). As such the latest 2.6 version kernel has included new features to address the latency issue to improve its real-time performance. The three main new features included in the 2.6 kernel for this purpose are as follows [4] [5].

(i) Preemptive Option (CONFIG_PREEMPT) - make kernel v2.6 code became preemptible, except the spinlock.
(ii) O(1) Scheduler – make the time on searching for new running task became constant to help reducing the scheduler duration.
(iii) Optional Virtual Memory – allow user to not use the virtual memory to reduce the context switching time.

With features like the above included in the 2.6 kernel, the latest mainline Linux are now becoming more suitable for applications that require certain amount of real-time performance, and indeed is the main reason why Linux have increasingly been used in commercial consumer electronic

1521-9097/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPADS.2008.108

products that normally only require soft real-time performance. However, the mainline 2.6 kernel is still not considered to be a hard real-time operating system due to some unresolved issues like the unpredictable latency that can be caused by the kernel spinlock. Furthermore, its unpredictable latency to response and handle a new interrupt violates the deterministic characteristic required of a RTOS.

In view of the real-time limitations in the 2.4 kernel and the 2.6 kernel, quite a number of real time extensions have been proposed and developed over the years to make Linux OS become a hard RTOS to fulfill the market needs. This paper hence presents a summary on the various versions of real-time enhanced Linux that are available in the market and compare their techniques to bring real time features into the standard Linux. Of particular interest for education purposes are the open source community effort versions like RTAI [6] and Xenomai [7], where the free availability of their program codes provide a good source for students to investigate, explore, experiment and understand the various features that are of important for real-time embedded systems.

II. REAL-TIME ENHANCEMENT FOR LINUX

There are two general approaches employed to improve the real time performance of the Linux operating system. The first approach involves patching the kernel to make it more preemptible, provide better interrupt latency, better context switching time, better scheduling latency and having a finer timer granularity. The second approach involves using a small real-time kernel (i.e. a sub-kernel in a dual kernel approach) to run the Linux kernel as a low priority task, while handle interrupts from the underlying hardware, and only forward the relevant ones to the Linux kernel.

a) The followings lists the more common techniques that are involves in patching the kernel in order to improve its real-time performance, particularly applicable for the 2.4 Kernel.

i) Addition of Preemption Points

Preemption points are calls to the scheduler to check if a higher priority task is ready to run. These calls provide opportunities for the kernel to preempt itself in the event that there is a higher priority task waiting to be executed. As Linux Kernel contains some long execution paths that normally cannot be preempted and cause delays in executing a higher priority task, inserting preemption points along these paths allows the system to be more responsive to higher priority tasks and external events like interrupts.

However, there are sections of code in the kernel that cannot be safely preempted. These critical sections cannot be executed by more than one task at the same time and are protected by spin locks. In creating a preemptible kernel, these spin locks are modified to prevent preemption at these places. though will still be permitted in other sections of the code.

ii) Rescheduling at Lock Breaking

Calling the scheduler at the time of releasing a lock is another attempt at preemption other than adding the preemption points. This is a relatively simple effort. When the kernel acquires a lock and then subsequently releases it, preemption will be checked for at the point of release.

iii) Utilizing SMP Macros

The Linux kernel supports Symmetric Multi-Processing (SMP) systems. These are systems with multiple identical processors that are connected to a single shared memory bus. There are SMP macros in the kernel that can be invoked to provide preemption. This technique involves tricking a SMP kernel running on a uniprocessor system to think that it is running on a SMP platform.

iv) Miscellaneous real-time patches

There are also real-time patches that provide for microsecond timers, higher number of task priorities as well as improved thread synchronization methods. Microsecond timers are important for applications that require a finer-grained timing mechanism. To these applications, the standard 10ms “heartbeat” in the Linux kernel is insufficient. Increasing the number of priority levels for a task is important for systems with many tasks running. Each task can be assigned its own priority without running out of priority levels. As for improved thread synchronization methods, these methods offer more options and hence allows greater configurability.

b) A sub-kernel is a very small and minimal operating system that provides a very good real-time performance in terms of determinism, response time, preemptibility and timing guarantees.

In this approach, a small OS real-time kernel run the Linux kernel as a low priority task. The key idea behind this approach is to not trying to fix the Linux kernel to attain better real-time performance, but instead introduce a small real-time kernel to handle the scheduling of tasks, handling of interrupts, providing a good set of task synchronization and communication mechanisms, and also interfacing to the Linux kernel which is running as one of its task. Any real-time task will be run separately by the sub-kernel in parallel to the Linux Kernel. The Linux kernel is suspended when the real-time sub-kernel deals with interrupts or handles task scheduling and communication issues. As such, the Linux kernel is not allowed to disable interrupts. Hence, it has to be modified to certain extend to prevent disabling of interrupts.

There are pros and cons between these two approaches, but the sub-kernel approach is the more popular methods used by in the open source domain, as will be discussed in the next section.

III. COMPARISON

This section compare the issues involves in using the two different approaches normally adopted to enhance the real-time performance of Linux.

i) Latency guarantees

One of the limitation of the kernel patching approach is that it is really not possible to claim guarantees on the latencies of the patched kernel. The latencies that are published on the patched kernel are usually statistically measured, and are the worst-case measurements under various load and stress conditions. For systems that require hard real-time performance, these patched kernels may not be able to satisfy the requirements due to a lack of timing guarantees. On the other hand, the real-time sub-kernel approach can provide guarantees on the latencies of the system due to the much smaller sub-kernel code (30 KB to 64 KB) involved. This allows a complete code analysis on the sub-kernel and map out the longest path during the design time.

ii) Maintenance

Patching of Kernel inevitably imposes an additional burden on all contributing kernel developers to ensure that any new code that is added to the kernel, including loadable driver modules, must meet the strict requirement of not introducing long non-preemptible code paths. As it is generally more difficult to write preemptible code than non-preemptible code, this may result in a slower rate of code development for the kernel as well as new Linux drivers.

iii*) Bugs*

Patching the kernel for preemptibility introduces substantial change in many areas of the kernel code. This increases the risk of breaking something in the kernel as the inner workings of the kernel is complex and not easily understood. In contrast, the real-time sub-kernel approach usually leaves most of the Linux kernel untouched, with the kernel patch needed to be of minimum. In the case of RTAI, the patch only involves the modification and addition less than 100 lines in the kernel code. Hence, the risk of introducing bugs is kept to a minimum.

iv) Unique Programming Model

In the real-time sub-kernel approach, the real-time task is now handled by the sub-kernel instead of the Linux kernel. As such, these tasks have to be programmed as kernel modules instead of normal user programs. Kernel modules use a slightly different set of APIs compared to normal user space programs, with additional set of POSIX real-time API to allow the real-time modules to take advantage of the real-time capabilities of the system.

This unique programming model does not really apply to the approach of patching the Linux kernel to make it preemptible. Furthermore, the patching of the kernel has the benefit of providing real-time response behavior to all user space programs. Nevertheless, this problem is in Xenomai, a sub-kernel based approach, through its real-time shadow services concept.

IV. REAL-TIME LINUX PLATFORMS

As can be seen from the last section, patching of kernel is a much more involved and challenging approach than using the sub-kernel approach, and generally best performed by parties who have the resources and commitments like the Linux OS commercial vendors. This section lists some of the main real-time Linux solutions available in the market today. Of more interest for education purposes are those developed by the open source community, which all happen to employ the real-time sub-kernel approach.

a) Patching Kernel Approach

This is usually coupled with an improved scheduler, high resolution timers and additional thread synchronization mechanisms.

i) MontaVista Linux

MontaVista[8] Linux is probably the most well known and one of the pioneers in providing commercial grade real-time Linux. Its Linux-based RTOS is used in consumer electronics devices like a number of smartphones in Japan as well as mobile phones made by Motorola. Most of the core changes introduced by MontaVista to the Linux kernel were later submitted back to the Linux open source community and maintained as a preemptible kernel patch which are included in the Linux 2.6 stable kernel.

ii) TimeSys Linux

TimeSys[9] Linux is a derivative of the standard Linux kernel that is fully preemptible and is specifically aims for the embedded Linux domains. It is available in four flavors, including one for real-time. It includes a constant time scheduler, has fully schedulable interrupt handlers, fully schedulable soft interrupt requests (IRQs) and reduced interrupt disable times.

iii) Linux Real-Time Kernel Patch

The Linux real-time kernel patch[10] is a cumulative result of several previous efforts to reduce the Linux kernel latency. This patch has many contributors and is currently available as the rt patch for the 2.6 kernel. Some of the main features of this real-time kernel patch are:

- Replacement of spinlocks to mutexes
- Running ISRs as kernel tasks
- Preemptible soft IRQs
- Preemptible RCU

b) Real-Time Sub-kernel Approach

i) RTLinux

RTLinux[11] was originally an open source development, but is now being acquired and available through Wind River[12]. It utilized a small real-time sub-kernel to intercepts all the hardware interrupts. For those interrupts that are appropriate for the real-time executive, the associated ISR is executed. All other interrupts are passed on to the Linux kernel as software interrupts and is done only when the real-time executive is idle. To minimize the changes needed for the Linux kernel, an emulation of the interrupt control hardware is provided to the Linux kernel. Communication between the real-time tasks and Linux user tasks is accomplished through the use of message queues and shared memory.

ii) RTAI

RTAI[6] (Real-Time Application Interface) is an open source project supported by the open source community with similar concept to those of RTLinux. RTAI uses the concept of a hardware abstraction layer (HAL), simulating a hardware platform for Linux to run on. In essence, the Linux kernel thinks it is running on a real hardware when in actual fact, the hardware environment is a software simulated one. This HAL is also known as ADEOS[13] (Adaptive Domain Environment for Operating Systems). As a result of this approach, the modifications needed to the Linux kernel are kept to the minimum. RTAI also provides an extension known as LXRT (Linux Real-Time) which allows real-time applications to run in user space instead of kernel space. This extension provides an option for developers to either code their real-time applications as kernel modules (which used to be the only way) or to code them as normal user space programs.

iii) Xenomai

Xenomai[7] was originally developed together with the RTAI but separated in 2005. Hence it also uses the ADEOS that acts as a resource virtualization layer to allow sharing of hardware resources among multiple kernel components (Domains), with Xenomai as the highest priority in the ADEOS domain. Compare to RTAI, the Xenomai's main goal is to be more focus on the clean extensibility (RTOS skins), portability, and maintainability rather than fast dispatching. It is also better structured and is available for a larger number of platforms. One of the feature of interest in Xenomai is the real-time shadow services that enable the real-time thread to be migrated between the Xenomai and Linux domain. When a real-time task (e.g. migrated from the Xenomai Domain) executes into the Linux domain on a given processor, the Linux kernel as a whole inherits the real-time priority of such task, and thus competes for the CPU resource by priority with other real-time tasks regardless of the domain (i.e. Linux or Xenomai) they happen to belong to.

Performance comparison[14] between Xenomai, RTAI & mainline Linux Kernel (v2.6.14) shows that Xenomai is slightly less performing than RTAI, mainly because of its layered approach that introduces overhead in the interrupt management. The comparison test results between Xenomai and Linux (under no load condition) show that they are similar in term of interrupt latency, rescheduling delay and network communication performance. However, Xenomai would have the edge once the load is increased, and will have much better determinism than the mainline Linux.

V. SUMMARY

This paper describes the various real-time issues when using Linux for embedded systems. It discusses the main real-time deficiencies of the Linux 2.4 Kernel and the improvements that are introduced in the Linux 2.6 Kernel. It then compares the two main approaches used to further enhance the Linux kernel to meet hard real-time requirement. While the kernel patch approach has the benefit of bringing real-time response behaviour to all user space programs, the real-time sub-kernel approach provides better control of latencies and hence determinism.

A number of real-time enhanced Linux are compared. Of particular interest for education purpose are the open source project types where the availability of the program code would allow the student to explore and understand the various features required of an RTOS.

REFERENCES

[1] Richard Nass (2008, Sept), An insider's view of the 2008 Embedded Market Study [online]. http://www.embedded.com/products/softwaretools/210200580

[2] Carrier Grade Linux [online]. http://www.linuxfoundation.org/en/Carrier_Grade_Linux

[3] CE Linux Forum [online]. http://tree.celinuxforum.org/CelfPubWiki/RealTimeResources

[4] Robert Love, Linux Kernel Development, 2003

[5] Aseem R. Deshpande, Linux Kernel 2.6: the Future of Embedded Computing [online]. http://www.linuxjournal.com/article/7477

[6] RTAI - the Real-Time Application Interface for Linux from DIAPM [online]. https://www.rtai.org/

[7] The Xenomai Project [online]. http://www.xenomai.org/

[8] MontaVista, Real-Time Linux Development from MontaVista [online]. http://www.mvista.com/real_time_linux.php

[9] Dr. Doug Locke, A TimeSys perspective on the Linux preemptible kernel [online]. http://www.linuxdevices.com/articles/AT6106723802.html

[10] Linux rt kernel patch project [online]. http://www.kernel.org/pub/linux/kernel/projects/rt/

[11] Victor Yodaiken, An Introduction to RTLinux [online]. http://www.linuxdevices.com/articles/AT3694406595.html

[12] RTLinuxFree, What Is RTLinux? [online]. http://www.rtlinuxfree.com/

[13] The ADEOS Project [online]. http://home.gna.org/adeos/

[14] Barbalace A, Luchetta A, Manduchi G, Moro M, Soppelsa A, Taliercio C. Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application**.** IEEE Transactions on Nuclear Science, Vol 55, Issue 1, page 435-439