International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 1 131 - 135

Real-time capabilities in the standard Linux Kernel: How to enable and use them?

[1]Luc Perneel, [2]Fei Guan, [3]Long Peng, [4]Hasan Fayyad-Kazan, [5,6]Martin Timmerman

[1]Electronics and Informatics Department
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel-Belgium
{ [1]luc.perneel, [2]feiguan, [3]longpeng, [4]hafayyad, [5]martin.timmerman } @vub.ac.be

[6]CEO Dedicated Systems Experts NV/SA Belgium m.timmerman@dedicated-systems.com

***Abstract*—** Linux was originally designed as a general purpose operating system without consideration for real-time applications. Recently, it has become more attractive to the real-time community due to its low cost and open source approach. In order to help the real-time community, we will present in this paper the practical steps required to achieve a real-time Linux by applying the PREEMPT-RT patches which will provide Linux with these capabilities. We will also focus on some of the kernel configuration that should get attention while building the kernel in order to maintain the real-time behavior of the system during runtime.

*****

I. Introduction

Because of its free open source advantage, stability and supporting multi-processor architecture, Linux operating system (OS) stands high in many (embedded) commercial product developers' favor and becomes the fastest-growing one of (embedded) operation systems [1]. Moreover, its reliability and robustness made it widely used in safety margins and mission critical systems.

In these contexts, time is extremely important: these systems must meet their temporal requirements in every failure and fault scenario. A system where the correctness of an operation depends upon the time in which the operation is completed is called a real-time system. [2]

Real-time is an often misunderstood and/or misapplied property of operating systems. It doesn't mean fast; Real-time deals with *guarantees*, not with *raw speed* [3]. In other words, the system must be deterministic to guarantee timing behavior in the face of varying loads (from minimal to worst case) [4].

Although Linux is a popular and widely used OS, the standard Linux kernel fails to provide the timing guarantees required by critical real-time systems [5]. To circumvent this problem, academic research and industrial efforts have created several real-time Linux implementations [6]. The most adopted solutions are RTLinux/RTCore, RTAI, Xenomai and the PREEMPT-RT [7] patch. Each one of these real-time enhanced kernels has its internal architecture, its strength and weaknesses [8]. All these approaches operate around the periphery of the kernel, except PREEMPT-RT patch which is mainlined in the current kernel and used by great actors such as WindRiver in their Linux4 [9] solution.

We will focus in this paper on the real-time capabilities provided by PREEMPT-RT patch to the standard 2.6 kernel, and show how to enable and use them correctly.

II. Real-time must be either hard or soft

Real-time systems and applications can be classified in several ways. One classification divides them in two classes: "hard" real-time and "soft" real-time.

A *hard real-time* system is characterized by the fact that meeting the applications' deadlines is the primary metric of success [10]. In other words, failing to meet the applications' deadlines can have a catastrophic result.

Conversely, a *soft real-time* system is suitable to run applications whose deadlines must be satisfied "most of the times," [10]. In other words, a soft real-time system can miss deadline without the overall system failing.

III. Linux PREEMPT-RT

Linux Preempt-RT (LinuxPrt) [11, 12] is a Linux real-time patch maintained by Ingo Molnar and Thomas Gleixner [15]. This patch is the most successful Linux modification that transforms the Linux into a fully preemptible kernel without the help of microkernel (the architecture implemented in RTAI or RTLinux) [13]. It allows almost the whole kernel to be preempted, except for a few very small regions of code ("raw_spinlock critical regions"). This is done by replacing most kernel spinlocks with mutexes that support priority inheritance and are preemptive, as well as moving all interrupts to kernel threads [8, 14].

Also, this patch presents new operating system enrichments to reduce both maximum and average response time of the Linux kernel [8]. These enhancements were progressively added to the Linux kernel to offer real-time capabilities. The most important enhancements are: High resolution timers (a patch set, which is independently maintained by Thomas Gliexner [15], which allows precise timed scheduling and removes the dependency of timers on the periodic scheduler tick [16]) , complete kernel preemption, interrupts management as threads, hard and soft IRQ as threads, and

priority inheritance mechanism. Some of these new features like threaded IRQ are currently pushed to the mainline kernel by the patch maintainers [8].

IV. Real-Time in the standard 2.6 kernel

Since the kernel version 2.4, a lot of improvements regarding real-time behaviour found their way into the standard "Vanilla" kernel. In this kernel version (2.4), when a user space process makes a call into the kernel (through a system call), it cannot be preempted. This means that if a low-priority process makes a system call, a high-priority process must wait until that call is complete before it can gain access to the CPU.

But some patches were already there to enable some "voluntary" preemption, in a way that the kernel allowed preemption on locations where it believed it was safe to do so.

Since the Linux 2.6 (Figure 1) releases, these voluntary preemption points were introduced in the Vanilla kernel. Also in Linux version 2.6, a more predictable O(1) scheduler was introduced which is a great benefit for the performance, even when a large number of tasks exists, as it can operate in bounded time regardless of the number of tasks to execute [4]).

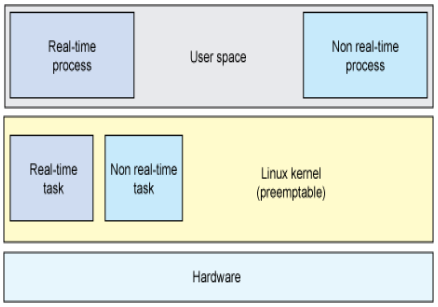

Figure 1: Standard 2.6 Linux kernel with preemption

Remark that for non-real-time threads, the O(1) scheduler has been replaced by the "Completely Fair Scheduler" (CFS) in Linux version 2.6.23, which is a O(log N) scheduler.

Linux is not a hard real-time operating system as it does not guarantee a task to meet strict deadlines [17]. Today, in the 2.6 kernel, you can get soft real-time performance through a simple configuration to make the kernel fully preemptable [4]. This can be achieved by applying the PREEMPT-RT patch that adds the new configuration option `CONFIG_PREEMPT_RT` which tries to minimize the time that preemption is disabled (by locks, interrupts, soft interrupt handling and so on). [18]

Although this configuration option enables soft real-time performance, it does so at a cost. That cost is slightly lower throughput and a small reduction in kernel performance because of the added overhead of the CONFIG_RTPREEMPT option [4]. This is normal and a fundamental rule in real-time software: latency improvements have a negative impact on throughput. We did some quick measurements using an NFS exported partition on the unit under test and reading a huge file from this exported file system mounted on a remote machine. To avoid file system caching effects, the test was done after a reboot of both machines. This very rudimentary test showed a negative throughput impact between 5 and 10% by enabling the PREEMPT_RT patch!

Note that the Linux kernel base code is continuously growing with modifications and enhancements through the Linux evolution. For instance, more and more PREEMPT_RT features made their way into the Vanilla kernel. An example of that is the "priority inheritance mutex" concept which is the solution of the priority inversion avoidance mechanisms on protection mechanisms, where a low priority thread cannot block a high priority one, moved from the PREEMPT_RT patch to the Vanilla kernel in V2.6.18.

Also the "kernel lock" concept (the famous Big Kernel Lock (BKL)) which was introduced for multiprocessor systems support was removed in the kernel version 2.6.39 because of its serious impact on latencies in such systems.

Although a lot of improvements moved from the PREEMPT_RT to the mainline Vanilla kernel, there are still different things lacking and therefore the PREEMPT_RT patch is still required and maintained.

V. Building real-time Linux for x86 platform using PREEMPT_RT

Building a complete Linux environment for a custom board remains a daunting task if you have to do everything by yourself (build cross-tool chains, libraries, kernel, and setup root system). Furthermore, you will need a boot loader (mostly u-boot is used). Luckily, most board vendors deliver their boards with all tools needed and pre-configured kernel / root file system.

Setting up the kernel configuration is needed to define how the kernel will behave. For example, will it be preemptible or not? Also, the kernel configuration should define the hardware it needs to support (linked in the kernel itself or by using loadable modules).

In general, you should know the Linux kernel internals in order to setup a configuration that is specific for your platform. Surely, when the RAM size is limited, disabling some kernel features and modules can decrease the required memory size for running the kernel. Sometimes you need to take a look into the code to understand the impact of some options. Anyhow, setting up a Board Support Package (BSP) configuration is a difficult task for any operating system and requires some (serious) expertise from the developer.

Remark as well that some modules and/or kernel options can affect the real-time performance. So you should be careful in this issue and it is recommended that you start from a specialised vendor release or from an OSADL version.

We built a real-time (RT) Linux version for an x86 platform using the Vanilla Linux 2.6.33.7 and the PREEMPT_RT patch v30. This PREEMPT_RT patch was the latest version officially released by OSADL [19] (the Open Source Automation Development Lab).

We started with the ODSAL kernel configuration. We removed only the unnecessary kernel modules in order to have the minimal kernel footprint. (Ex: USB support was omitted in the kernel). Our configuration is considered as an optimistic "best possible" configuration. Further, we changed some

kernel configurations because their default settings may seriously jeopardize the real-time behaviour.

We are going to highlight on some of the configuration options that should get some care when building a RT kernel in order not to affect the real time performance and behaviour of Linux during the runtime. Those configuration options are the following:

1) Optimize for size

This option is typically used in embedded systems due to the RAM size constraint. Remark that this option can increase cache hits and thus performance as well. But this will largely depend on your application and target.

2) Tickless system

This option generates an operating system clock tick only when a thread wants to be waken-up instead of a periodic one. This improves power consumption and CPU usage (it was mainly introduced for decreasing large virtual machine farms CPU usage). However, the clock tick becomes more complex with this option, and this option does not avoid clock ticks. So in the end, you will have less but longer ticks, which is not good for real-time performance.

3) RT Preemption model

This option should be enabled with the value of real-time to make sure that the kernel is a real-time preemptible kernel.

4) Power management

It should be disabled. We do not want to put the CPU in a lower power state. The reason for this is that it can take some time to throttle back towards full CPU speed. Again this can impact latency on critical moments and is thus bad for real-time purposes.

5) Minimal modules

Do not add modules that you will not use. The less code included in the kernel, the less that can jeopardize the real-time behaviour. Not all modules are already well behaving.

6) Disable memory swapping

When memory is swapped away onto some storage medium, the time to retrieve it when needed is a huge factor slower than when loaded from RAM. So it is not something you want to do when keeping deadlines is your concern.

The selected configuration settings will be finally stored in a ***.config*** file. This ***.config*** file will be used by the ***makefile*** for setting the compile options, creating configuration header files and selecting the files that should be compiled.

This file can be easily found in a running **Linux** distribution in the directory ***/proc*** under the name ***config.gz*** (path: ***/proc/config.gz)*** (at least if this option is built in the kernel).

This file can also be manually adapted (if you are experienced!) and can be used as a base for setting up a new configuration (by using ***make oldconfig***).

VI. Kernel runtime configuration

Building a kernel with the PREEMPT_RT patch and configuring it correctly is not always enough to guarantee real-time latencies. Some precautions still have to be taken at kernel runtime configuration.

The build "run-away" protection in the kernel is an important one. If the real-time priority threads take too much CPU time (default: if more than 950ms used during a second), then all real-time priority scheduling is stopped for 50ms. Although at first sight this looks a nice to have safety system, it is not something you want in a real-time system! The latency is suddenly increased with a horrible 50ms. Real-time systems should keep their deadlines and should go into a safe state when these deadlines are missed (which are typically done by external monitoring and watchdogs). You should disable this feature by setting the kernel variable `/proc/sys/kernel/sched_rt_runtime_us` to -1, and yes very low priority threads might never get the CPU. You should avoid this in your design.

VII. Application configuration

Your applications have to be built and configured correctly also! You should not allow the kernel to allocate real physical memory only upon first use (the default behaviour in the kernel). For instance, assume that you allocate a block of 1MB memory using the `malloc` call. What you get is just the virtual memory region of your process space, linked to the one and only read-only physical memory zero page (which contains zero data). As a result, reading from this allocated memory is not a problem and will always return zero.

However, upon first write access into this virtual memory space, a page fault will occur and a free physical page will be linked to the virtual page. This is called the Copy-On-Write behaviour. These page faults will be added to the worst case latency in your application and that is something one does not want in a RT system.

A solution for that is to use the `mlockall()` system call so that each virtual allocation will be immediately linked with physical memory. This of course slows down the start-up process, but improves latency at later times. Another solution is of course to perform a write on each page after allocation. However, the latter does not help for the allocation of the thread stack segments, something which is handled as well when using `mlockall`().

Still these precautions are not enough! One kernel feature that cannot be disabled is the eviction of read-only memory pages which are loaded from some storage medium. Even when swapping is not enabled in the kernel, this can still happen. Such read-only pages could be for instance the application code! Again this is something one does not want to happen.

The only solution here is to have the application started from a RAM filesystem (to make the read-only pages de-facto present in RAM). This should not only be the case for the real-time application, but as well for the libraries the application uses. As an alternative, you can use a static linked application.

VIII. Testing the built kernel to make sure of the availability of real time capabilities

Taking all the above mentioned precautions into account, we created a testing application, which is intended to test the real-time behaviour of the PREEMPT_RT patch. In this

application, we tested the behaviour of the operating system clock, thread scheduling latency, semaphores, mutexes and finally interrupt latencies and highest sustainable interrupt frequency.The detailed results can be found on [20].

So our answer for the first question: does the PREEMPT_RT patch helps to achieve real-time performance is clearly yes, especially if we compare Linux with for instance a standard Android distribution. In this comparison, we can see the huge difference (multiple factors) between these variants.

The second question is if Linux is a real-time OS? Indeed, there is a good control on worst case latencies. However, these are still a factor away from the commercial real-time operating systems, such as QNX and Windows CE. It largely depends on your requirements and how long the latencies are allowed to be. But as latencies are bound, it is usable for real-time systems.

IX. Can we have a 100% deterministic real-time GNU/Linux kernel?

To answer this you need to keep in mind that the same GNU/Linux kernel code base runs on the world's fastest supercomputers as well as mobile phones. GNU/Linux was not written with real-time in mind - not even to run on supercomputers or mobile phones - but to provide by default maximum raw processing power and throughput shared in a fair way among users and processes, which is typical for multi-user multi-processing operating systems. [18]

GNU/Linux supports more different types of devices and processors than any other operating system ever has in the history of computing [21]. So although it is possible to make the scheduler behave deterministic like one would expect from a real-time operating system you cannot assume that all GNU/Linux drives are written with real-time in mind as well. This means that if you don't use those drivers/configurations which destroy the real-time behavior a “real-time enabled” GNU/Linux does not behave worse than other real-time kernels. Besides you can always run GNU/Linux on top of a real-time operating system. [18]

X. Conclusion

Real-time together with GNU/Linux seems to be on the rise, but there is no “one-fits-it-all” solution to bring real-time capabilities to it. The reason is that there is no silver bullet to make something as big and complex as GNU/Linux 100% real-time aware since this would extremely costly in terms of maintenance. [18]

With our Linux build and the testing application, we tested the real time behavior and performance (like semaphore, thread creation) of a real-time Linux and we concluded that the real-time capabilities are there and working correctly, and Linux can be considered as a real-time operating system if it is correctly configured.

References

[1] Y. Liping and S. Kai, "Improvement and Test of Real-time Performance of Embedded Linux 2.6 Kernel," Digital Content Technology and its Applications, vol. 5, p. 7, 2011.

[2] A. Claudi and A. F. Dragoni, "Lachesis: a testsuite for Linux based real-time systems," in 13th Real-Time Linux Workshop, Prague, 2011.

[3] K. Yaghmor, J. Masters, G. Ben-Yossef and P. Gerum, Building Embedded Linux Systems, 2nd Edition, Building Embedded Linux Systems, 2nd Edition, 2008.

[4] T. Jones, "Anatomy of real-time Linux architectures," IBM, [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-real-time-linux/.

[5] P. Regnier, G. Lima and L. Barreto, "Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems," ACM SIGOPS Operating Systems Review, vol. 42, no. 6, pp. 52-63, 2008.

[6] Z. Chen, X. Luo and Z. Zhang, "Research Reform on Embedded Linux's Hard Real-Time Capability in Application," in Proceedings of the 2008 International Conference on Embedded Software and Systems Symposia, 2008.

[7] M. Mossige, P. Sampath and R. Rao, "Evaluation of Linux rt-preempt for embedded industrial devices for Automation and Power Technologies - A Case Study," in Proceedings of the Ninth Real-Time Linux Workshop, Linz, 2007.

[8] N. Litayem and S. Ben Saoud, "Impact of the Linux Real-time Enhancements on the System Performances for Multi-core Intel Architectures," International Journal of Computer Applications, vol. 17, no. 3, 2011.

[9] W. River, "The First with the Latest: Wind River Linux 4," [Online]. Available: http://www.windriver.com/announces/linux4/. [Accessed 24 Juin 2012].

[10] E. Betti, D. P. Bovet, D. Pierre and R. Gioiosa, "Hard Real-Time Performances in Multiprocessor-Embedded Systems Using ASMP-Linux," EURASIP Journal on Embedded Systems, vol. 2008, p. 16, 2007.

[11] P. McKenney, "A realtime preemption overview," 2005. [Online]. Available: http://lwn.net/Articles/146861/.

[12] S. Rostedt and D. V. Hart, "Internals of the RT Patch," in Proceedings of the Linux Symposium, 2007.

[13] D. Kang, W. Lee and C. Park, "Kernel Thread Scheduling in Real-Time Linux for Wearable Computers," ETRI Jornal, vol. 29, 2007.

[14] S. Arthur, C. Emde and N. Mc Guire, "Assessment of the Realtime Preemption Patches (RT-Preempt) and their impact on the general purpose performance of the system," in Real-Time Linux workshop, Linz, Austria, 2007.

[15] "CONFIG PREEMPT RT Patch-RT wiki," [Online]. Available: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch.

[16] "High resolution timers - RT wiki," [Online]. Available: https://rt.wiki.kernel.org/index.php/High_resolution_timer.

[17] A. S. Ugal, "Hard Real Time Linux using Xenomai on Intel Multi-core processors," Intel corporation, 2009.

[18] R. Berger, "Getting real (time) about embedded GNU/Linux," [Online]. Available: http://www.eetimes.com.

[19] "Open Source Automation Development Lab," [Online]. Available: https://www.osadl.org/.

[20] D. S. Experts, "RTOS evaluation Reports and related papers," [Online]. Available: http://download.dedicated-systems.com.

[21] A. Oram and G. Wilson, Beautiful Code: Leading Programmers Explain How They Think, O'Reilly Media, 2007.