Myths and Realities of Real-Time Linux Software Systems

Kushal Koolwal
VersaLogic Corporation
3888 Stewart Road, Eugene, OR 97402 USA
kushalk@versalogic.com

Abstract

This document addresses some of the differences between real-time and general purpose operating systems. This includes an analysis of several common misconceptions including performance issues, latency, hard vs. soft real-time systems, programming APIs and the software kernel. In addition, there are suggestions for implementation options to convert Linux to a real-time operating system.

1 What is real-time system?

According to the Real-time Computing FAQ[1], a real-time operating system (RTOS) is defined as follows:

"A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation, but also upon the time at which the result is produced. If the timing constraints are not met, system failure is said to have occurred."

In other words, a system should be able to perform computations deterministically - an important concept in real-time systems. A system is said to be deterministic if it's possible to predict the timings of its computation precisely, and computations are logically correct. For example, a real-time system should give the result of "2+2" not only as "4", but it should also produce the correct result within a specified time range.

1.1 Analogy for real-time systems

To understand the concept of a real-time system, consider the example of an automobile airbag system, one of the most critical systems in a modern car. An airbag should be deployed within a particular time interval (for example, within 1 second[1]) after the sensors in the car detect a collision.

A car equipped with a real-time computer system will always guarantee that the airbag will be deployed within 1 second, no matter what happens (see worst-case scenario below). The real-time system will put all the non-critical tasks, such as powering the stereo, switching on the air-conditioning, activating cruise control, etc. on hold and will immediately deploy the airbag as soon as it receives the signal from the sensors.

If a car is equipped with a non real-time system, however, there is no guarantee that the airbag will be deployed exactly within 1 second after the sensor detects a collision. Such a non real-time system might deploy the airbag after finishing the request to activate the cruise control, which happened to come before the request to deploy the airbag. An airbag system that deploys even 0.1 second later than the expected time is as bad as not deploying at all. Why? Because it might already be too late to save the life of the passenger. In fact, in real-time systems, a difference of even 0.000001 second may be important.

2 Metrics of real-time

A number of parameters are required to quantify real-time systems. To understand these parameters, it is important to define the term "latency."

[1] Usually the time interval is much less than 1 sec., but we use 1 sec. for simplicity.

Latency: In computing, latency is "the time that elapses between a stimulus and the response to it[2]."

Using this definition and our airbag example, latency in real-time systems is defined as the time elapsed between the triggering of an event (signal sent by sensors), requesting a particular task (deploying the airbag), and the actual task being executed (airbag deployed). In other words, it is the delay between an action and a response to that particular action.

Following are some important metrics that help us to quantify a real-time system:

a) Interrupt Latency: The time elapsed between the generation of an interrupt and the start of the execution of the corresponding interrupt handler.

Example: When a hardware device performs a task, it generates an interrupt (an IRQ). This interrupt has the information about the task to be performed and about the interrupt handler (ISR) to be executed. The interrupt handler then performs the particular task.

b) Scheduling Latency: According to Clark Williams of RedHat[3], "it is the time between a wakeup (the stimulus) signaling that an event has occurred and the kernel scheduler getting an opportunity to schedule the thread that is waiting for the wakeup to occur (the response). Wakeups can be caused by hardware interrupts, or by other threads." Scheduling latency is also known as **task-response latency** or **dispatch latency**.

An extension of this concept is the **priority inversion**. Here is one example: Sometimes while scheduling different threads (or tasks) a situation might arise in which a higher priority thread might be waiting to obtain a resource that is held by a lower priority thread. Here a kernel has to reach a safe point, named a **rescheduling point**[4], before it can give the resource to the higher priority task. In fact, one of the failures of the Mars Pathfinder was caused by an unbounded priority inversion[5].

Example: Suppose your car (a low priority task) enters a busy traffic intersection (CPU of the system) and you guess that you might be able to cross the intersection (use the CPU to process), but then you realize that all of the traffic in front of you has abruptly come to a stop (an unexpected event occurs in the system), and you are stuck right in the middle of the intersection. Suddenly you hear the siren of a fire truck (a high priority task) heading toward you to cross the intersection (use CPU to process). Unfortunately, you cannot move, as you are stuck, and the fire truck also has to stop until you are out of its way (a high priority task has to wait for a low priority task). As a result, a building might continue burning and lives may be lost.

In the real-life scenario described above, the fire truck has to wait, but in a real-time system a real-time kernel can create paths (roads) on-the-fly which allows a high priority task (fire truck) to bypass a low priority task (traffic jam). These are named rescheduling points.

The figure below illustrates Interrupt Latency and Scheduling Latency[4].

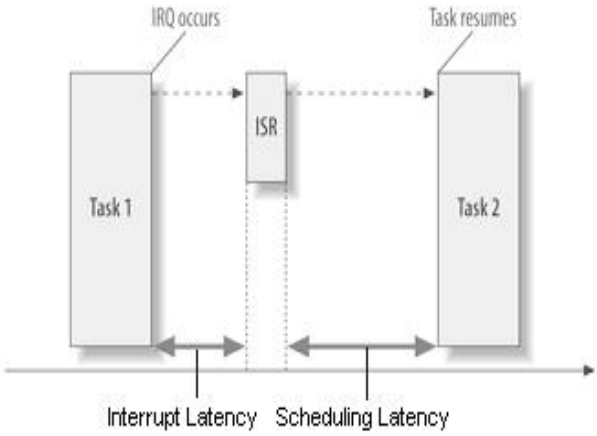

FIGURE 1: *Interrupt and Scheduling Latencies*

c) **Worst-case Latency**: This is defined as the maximum time that can lapse before the desired event occurs.

Worst-case refers to the condition when the system is under heavy load - more CPU load and I/O operations occurring than are typical for the particular system. When we talk about real-time it is very important that we define our results not only on an average basis, but in a worst-case scenario as well.

Example: The 1 second (maximum) limit discussed in the airbag example is the worst-case scenario. It means that, on average, the airbag will be deployed in, say, 0.3 seconds, but in a worst-case scenario (system under heavy load), the airbag will be deployed within 1 second.

Latency requirements: In general, a real-time system should be able to give an average latency in the range of 10-20[2] μs. Under a worst-case scenario,

[2]1 second = 1000 milliseconds = 1000000 μs.

[3]These latency numbers are not set in stone. It really depends upon your needs, the underlying application and the system. We found these numbers based on our research [10] and [14]. Also, these numbers are with respect to Linux OS. Your mileage

the latency should be within a few hundred microseconds[3].

3 Hard vs. Soft real-time

Most people use the terms hard real-time and soft real-time loosely. However, it is essential to distinguish between the two terms.

Hard real-time: A system that can meet the desired deadlines at all times (100 percent), even under a worst-case system load. In hard real-time systems, missing a deadline, even a single time, can have fatal consequences. Some examples are airbag systems, ABS (anti-lock braking systems), missile systems, aircraft controls, etc. Hard real-time systems are used in cases where a particular task, usually involving life safety issues, needs to be performed within a particular time frame, otherwise a catastrophic event will occur.

Soft real-time: A system that can meet the desired deadlines on average. Some examples are online audio/video broadcasts, stock market software, cell phone display, etc. In other words, a soft real-time system will give reduced average latency but not a guaranteed maximum response time. Due to the lack of deadline support, a soft RTOS is risky to use for industrial control and robotics[6].

A soft RTOS can miss a few deadlines (such as dropping a few frames in a video application) without failing the overall system. A hard RTOS cannot afford to miss a single deadline at any point because of its mission critical application nature (such as deploying an airbag).

4 GPOS vs. RTOS

Since the basics of an RTOS have been discussed, now is a good time to compare some of the important characteristics of an RTOS to those of a general-purpose operating system (GPOS). A GPOS is used by average computer users for day-to-day activities like checking e-mail, typing documents, listening to music, watching videos, etc. The table below compares GPOS and RTOS systems:

Parameters	**GPOS**	**RTOS**
Primary function	Performance	Responsiveness
Time bounded	No	Yes
Guaranteed Calculation (Worst-case)	No	Yes
Calculation Example (Max = Worst-case)	Avg: 5 secs (99.9%) Max: 22 secs (0.1%)[0] Deadline: 20 secs	Avg: 7 secs Max: 15 secs Deadline: 20 secs
Examples	WinXP, WinXPe, Linux, DOS, MacOS	WinCE, VxWorks, QNX, Linux (RT patch)

5 Myths of real-time: Real-time = Performance?

It is not surprising how often people confuse real-time with performance. A real-time system does not mean faster performance. In fact, a real-time system may or may not lead to deterioration in performance. Application developers who think they need a real-time system for better performance of their application can actually get the performance they require by simply upgrading their hardware with faster processors, RAM, high speed buses, etc.

According to Yaghmour[4], "Real-time deals with guarantees, not with raw speed." However, a quality RTOS will still deliver decent overall throughput (performance) but can sacrifice throughput for being deterministic (or predictable). In fact, many benchmarking studies on the performance of a real-time kernel against a standard kernel have found that overall performance does not decrease significantly[7].

6 Who needs real-time?

As mentioned above, an RTOS does not necessarily mean a faster machine. Not everyone needs a real-time system. In fact, almost 90 percent of embedded applications do not require a hard real-time system. It depends upon the specific needs and the underlying application. The key to determining whether

will vary if you use a different OS or a different real-time approach (see below).

[0] 22 secs is not acceptable in this case and hence it does not qualify as a hard RTOS.

you really need a real-time system is to ask the right questions:

a) What type of latencies are required? What is the maximum latency that your application can tolerate without failing the system?

b) Does the application involve any life-safety issues? Or, in other words, does it need to perform a certain task within a particular deadline for the task to be of any value?

Example: A car manufacturer is designing a new car and requires the airbag system to deploy within 50 μs, at any given cost, after collision has occurred, or else the application (deploying of airbag) won't be of any value.

c) Is it absolutely necessary to process all of the data in the system, or is it acceptable for a small amount of data to remain unprocessed?

d) Is the application dependent upon an external device that needs a response within a particular time frame or else the entire system will fail?

The following table lists examples of some hard and soft real-time applications by industry:

Industry	**Hard Real-time**	**Soft Real-time**
Aerospace	Fault detection, Aircraft control	Display Screen
Finance	ATMs	Stock Market Websites
Industrial	Robotics, DSP	Temperature monitoring
Medical	CT Scan, MRI, fMRI	Blood extraction, Surgical Assist.
Communications	QoS	Audio/Video streaming, Networking, Camcorders

7 Real-time in Linux

Traditionally, Linux was designed to be a GPOS. However, many projects have been started to convert the Linux operating system into an RTOS: RTLinux (Wind River), Xenomai, RTAI, RT Preemption Patch, etc. A complete list of implementations is available on the Real Time Linux Foundation Web site[8].

There are two fundamental approaches to making the Linux kernel real-time:

a) Improve the kernel preemption itself; that is, make the parts of the code as preemptible as possible.

b) Introduce a new software layer (a sub-kernel) beneath (or along with) the actual Linux kernel, called the Co-kernel approach.

It is beyond the scope of this paper to talk about all the different types of Linux RTOS implementations. The approaches mentioned above, except the RT Preemption (PREEMPT_RT) approach, are fundamentally based on the sub-kernel concept, whereas the (PREEMPT_RT) approach is purely based on improving kernel preemption. The RT Preemption approach is described below.

7.1 Advantages of the PREEMPT_RT approach

One of the primary advantages of this approach is the availability of the same APIs that exist in the non real-time kernel. Application programmers can use the existing POSIX compliant APIs (available under Linux) that they have already used in writing their applications[9]. Developers don't need to learn additional APIs that might be introduced if they were to go with a Co-kernel approach.

7.2 Is the PREEMPT_RT patch a hard real-time or soft real-time system?

There has been a lot of debate in the form of technical discussions over this issue. This document will try to describe the current scenario. When a generic Linux kernel is downloaded from www.kernel.org, or when a Linux distribution like Debian, Fedora, Ubuntu, etc. that comes with a pre-packaged kernel is used, it is not a fully real-time kernel. However, the kernel can be configured to a soft real-time kernel by choosing the "Preemptible Kernel" (PREEMPT_DESKTOP) option under the kernel configuration menu.

Moreover, if the soft real-time kernel is not desirable, then the Linux kernel can be made into a generic real-time kernel[10] by applying Ingo Molnar's PREEMPT_RT patch[11]. This patch makes a regular Linux OS (kernel) into a fully Preemptible OS - an OS that has the capability to stop/hold a low priority task in favor of a high priority task. This patch is not part of the mainline kernel as of yet. However, the team behind this project is constantly making efforts to make it a part of the mainline kernel. In fact, some parts of the PREEMPT_RT patch

have already made it into the mainline kernel. For example, the "Voluntary Kernel Preemption" (PREEMPT_VOLUNTARY) option in the mainline kernel is the result of one such effort.

As mentioned above, it is still not clear whether the Linux kernel with the PREEMPT_RT patch applied can be called a hard real-time system or not. According to Building Embedded Linux Systems[4], "For most applications that need real-time determinism, the RT-patched Linux kernel provides adequate service. But for those real-time applications that need more than low latencies and actually have a system that can be vigorously audited against bugs, the Linux kernel, with or without the RT patch, is not sufficient."

7.3 Overview of PREEMPT_RT patch

What are the changes that are made to the mainline kernel with the PREEMPT_RT patch?

Giving a detailed technical explanation of every change made to the kernel code is beyond the scope of this document. However, here is a high-level overview of the changes made by this patch:

a) Converts all Interrupt Service Routines (ISRs) to kernel threads, known as "Threaded Interrupt Service Routines."

b) Replaces kernel "spinlocks" with "mutexes" that support priority inheritance and are preemptive.

c) Adds High Resolution Timer (HRT) support, which allows the timers to operate at a resolution of 1s.

d) Disables unbounded priority inversion.

(For more technical details about the above issues, please refer to [12] and [13].)

7.4 Device drivers and the PREEMT_RT patch

A common question from users is the following: "Is there a real-time driver for my ethernet controller that enables me to use the real-time Linux kernel (PREEMPT_RT) patch?" At present, most of the drivers available under the Linux kernel tree are quite robust and ready to be used with the real-time kernel (PREEMPT_RT patch). They are tightly written and highly scrutinized, so usually there is no room for further improvement. If any change has to be made in order to further reduce the latency, then the change has to be made to the sub-system.

For example, to reduce network operation related latency, the change has to be made in the networking stack (TCP/IP protocol, etc.) and not necessarily in the network card driver. It would still be good practice to have your device driver inspected by a real-time expert to see if there is further possibility to tighten the code to make it more preemptible. The real improvement will still come from making changes to the core code (networking stack), as explained above. However, keep in mind that if you decide to do this, the reduced latency will come at the cost of performance.

8 Conclusion

While this document presents an overview of the issues and concerns commonly experienced by deterministic computer systems, it is not intended to be a substitute for a detailed analysis of your specific system requirements or operational priorities.

In summary, a thorough examination of the capabilities and limitations of using Linux as a real-time operating system requires a detailed analysis of your specific system performance and application requirements. These include, but are not limited to, an assessment of the mission critical nature of your system, acceptable risks and the level and quality of performance that is expected.

References

[1] *http://www.faqs.org/faqs/realtime-computing/faq/*

[2] *http://dictionary.reference.com/browse/latency*

[3] *http://www.linuxdevices.com/files/article027/rh-rtpaper.pdf*

[4] *Building Embedded Linux Systems*, 2nd Edition, Karim Yaghmour, Jonathan Masters and Gilad Ben-Yossef, Aug. 2008,O'Reilly Media

[5] *http://catless.ncl.ac.uk/Risks/19.54.html#subj6*

[6] *Operating System Concepts*, 6th Edition, Silberschatz, A., Galvin, P. B., and Gagne, G., 2001,John Wiley and Sons

[7] *http://dslab.lzu.edu.cn:8080/docs/publications/Siro.pdf*

[8] *http://www.realtimelinuxfoundation.org/*

[9] *http://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions#List_of_realtime_APIs.3F*

[10] *http://www.ibm.com/developerworks/linux/library/l-real-time-linux/*

[11] *http://www.kernel.org/pub/linux/kernel/projects/rt/*

[12] *http://lwn.net/Articles/146861/*

[13] *http://www.linuxinsight.com/files/ols2007/rostedt-reprint.pdf*

[14] *http://www.mvista.com/download/videos/playvideo.php?video=16*