# Review: The Real-Time Linux Kernel: A Survey on PREEMPT_RT

Robert Geil, *UCLA Computer Science Department*

## Abstract

*The Real-Time Linux Kernel,* Reghenzani et al. provides an overview of the history of Linux as a real time system, and specifically describes work on the PREEMPT_RT patch being developed to make Linux a more real time system. In addition to developments in the Linux kernel, Reghenzani provides motivation for the development of a real time Linux for a wide range of industries, and gives an outline of credible research being done within the Linux real time world. The article provides a broad overview while engaging a reader untrained in the semantics of real time systems, performance and goals.

## 1. Summary

The survey article is organized into several subsections, each focusing on a different aspect of Linux as a real time system. Reghenzani opens with a motivation for the development of the PREEMPT_RT patch for the Linux Kernel, giving numerous examples of benefits in industry that a real time Linux system would provide. Following this, he introduces the reader to the concept of a real time system, and explains the place of these systems within industry. Next the authors introduce a brief history and various methods that have been attempted to modify Linux to become a real time system, followed by a performance analysis of Linux in a real time context, focusing on an upper bound for latency. Finally he more deeply describes the applications for real time systems, and concludes with a brief description of related works.

### 1.1. Real Time Systems

A real time system, as cited by Reghenzani is one in which "the correct execution of a real time program depends not only on the logical correctness of the output but also on whether such results are carried out within a given time frame or deadline"[1]. As this new component of correctness exists, there must be new rules to help keep this upper limit on execution. Reghenzani also describes the two options for running a real time program, either as a bootstrapped executable that runs atop the hardware, or as a part of a Real Time Operating System, or RTOS. In addition to how real time systems can run, there is a gradient of the level of real time performance required. At one end is Soft real time, where "the application goal is to maintain a Quality of Service … to ensure an acceptable user experience"[1]. At the other is Hard real time, where "the deadline cannot be missed for any reason, … typical of safety-critical scenarios."[1]. The survey focuses more on softer real time systems, as those requiring stringent proofs and act as life-critical software are beyond the real time abilities of Linux. In addition to the description of real time systems, Reghenzani provides a motivation for the development of Linux as a real time kernel. Many groups and organizations are seeking to minimize costs by using Commercial-off-the-Shelf, or COTS parts. These parts are typically cheaper and more supported. For example, Linux has a wide range of scientific applications as a General Purpose Operating System (GPOS), which would be beneficial to the real time developer. By patching Linux to provide real time support, with use of the PREEMPT_RT patch, fields as varied as Aerospace, robotics, networking and High Performance Computing could see a benefit.

### 1.2. Conflicts between RTOS and GPOS

The goals in creating a Real Time Operating System and a General Purpose Operating System are often competing. In a real time system, the goal is to have a maximum bound on time to complete a given task, meaning that predictability is more important than performance. In a GPOS, however, one of the most important metrics is throughput, and the time to complete an individual task, while still important, is not as important as pushing tasks through the operating system. As such, in a GPOS, there is no bounded maximum time that a given operation may take, especially in the complex environment of interrupts, scheduling and context switching.

### 1.3. Real Time Linux Approaches

In addition to the PREEMPT_RT approach that is focused on by this survey, Reghenzani describes several other approaches to produce a real time Linux system, the most prominent being a cokernel approach. In this case, a small sub-kernel operates between the kernel and the operating system, helping to maintain the flow of operations by catching hardware interrupts and acting as a scheduler, prioritizing real time tasks. The author describes three such systems, RTLinux, Xenomai, and RTAI. Each operate in a similar fashion. RTAI is a little different in that it is a "cokernel approach aimed at ensuring Linux real-time capabilities at the kernel-space level."[1], while the others allow for user-space real time programs. One downside of Xenomai, released in 2002, is that "a provided set of non-POSIX APIs must be used"[1]. This limitation makes it difficult to write programs that are as portable, reducing the benefit that operating in real time Linux provides. Another approach, rather than using a cokernel, is that which PREEMPT_RT uses, namely a semi-preemptible kernel. The original Linux kernel was entirely non-preemptible, which proved a difficulty in developing Linux as a real time operating system. Starting in the early 2000s, however, a series of changes and patches were proposed to add into the kernel, allowing for at least some preemption. As Reghenzani describes, "PREEMPT_RT patch of the Linux kernel is actually a set of patches … to trade the throughput of the system with low latencies and predictability, while maintaining the single-kernel approach, to allow the developers to easily write (user-space) real-time applications."[1]. By patching the kernel, there are some powerful benefits that are gained when compared to the cokernel paradigm. For example, cokernels are build atop, or rather alongside the existing kernel, creating complicated interactions, whereas the PREEMPT_RT patch exists directly within the kernel, meaning it is supported by the community. In addition, PREEMPT_RT allows programs to operate as user-space real time operations, continuing to utilize standard libraries, unlike the approach take by some cokernels. In addition to these two modes of creating real time Linux, Reghenzani lists several others, but is unable to describe them due to intellectual property laws.

### 1.4. Real Time Systems and Kernel Interruption

Because the goal of a real time system is bounded execution time, interrupting the kernel is a necessity of creating such a system. If the bound on an operation is approaching, and the kernel is executing other, non-related tasks, it is imperative that the kernel can be interrupted to return control to the real time task. Historically, the Linux kernel was uninterruptible. However, that has changed over the last several years, as described by Reghenzani. "Over the years, several contributions [to increase the preemptibility] have been integrated into the Linux kernel"[1]. Several prominent changes to increase interruptibility are listed, including preemptible kernel, high resolution timers, and full tickless operation. All of these are on the road to allow for full kernel preemptibility. Currently there are still a couple of areas in which the kernel cannot be preempted, including spin-locks and during the top portion of interrupt handling. Reghenzani goes on to describe further the various advances that have been made with the PREEMPT_RT patch to move toward full kernel preemptibility. High resolution timers have "resolution in the order of nanoseconds"[1], which lets tasks be timed much more precisely, something required of real time systems. Threaded Interrupt Handlers provide a more granular interrupt option, where the top half of the interrupt is critical, and cannot be interrupted, but the bottom half is interruptible. A patch for priority inversion was also a part of early PREEMPT_RT, as "priority inversion is able to cause unbounded and unpredictable latencies in the system"[1]. Another patch allowed for full tickless operation, where there is no timed interrupt. This "decreases the extra latencies due to the rate of preemptions triggered by the periodic timer."[1].

### 1.5. Metrics of Real Time Systems

In the next section, Reghenzani describes how to measure the performance of a real time Linux system. Because of the other interactions that the system has, unlike hard real time systems, there cannot be a mathematically proven upper limit on the time of execution. Therefore, metrics rely more on probability of completing the task before the given time. In particular, if X is a random distribution of time to completion, "Given a deadline D, the task is considered hard real time if $P(X \leq D) = 1$."[1]. As previously stated, this cannot be true in Linux because

there is no hard deadline on task completion due to unbounded times. However, Reghenzani provides an alternate approach of analyzing not just the probability of X, but also other factors including standard deviation, to get a soft probability that the deadline will be met. This area of probabilistic real time is a new field, however. Another factor that must be considered is the jitter, which is a value representing the maximum variation of two times to complete. The existence of this value makes calculating a hard upper bound on compute time much more difficult.

### 1.6. Methods of Performance Analysis

Due to the unique nature of real time tasks, Reghenzani states that "the system performances are usually evaluated in terms of *latency* and the *response-time jitter*"[1]. These two metrics provide a measure of how close a system is to being real time. It is also noted that prioritizing these measures comes at the cost of throughput, a traditional goal of operating systems. Reghenzani then goes on to summarize several methods used to analyze Linux systems, including stresser programs like *dohell* and *cpuburn* which benchmark CPU performance, and a specific tool *cyclictest* intended to track worst case latency. The author goes more in depth with cyclictest and its attempt to "accumulate all the latencies occurring while the system handles the response to an input event"[1]. The author provides some basic pseudo-code simulating how cyclictest works. In addition to these benchmarks, tracers like ftrace and KernelShark can be used to track the scheduler, allowing for analysis of real time system performance.

### 1.7. Classification as a Real Time System

Reghenzani describes how Linux, even with the PREEMPT_RT patch, cannot be classified as a real time system for hard real time operations. Because Linux is at the heart a General Purpose Operating System, there is too much occurring within the kernel to create a static analysis of the timing of the system. Rather, "real-time Linux systems can be considered *95% hard real time*"[1], meaning that deadlines are hit at least 95% of the time, and introducing a more probabilistic view of the system. Some aspects that prevent Linux from being a 100% hard real time system include the scheduler and interrupts, which can be unpredictable as to when they will yield and return the CPU. In addition, I/O is by nature unpredictable, a fact that is further complicated by the fact that jitter appears to depend on hardware configurations of CPU and I/O hardware.

### 1.8. Methods of Performance Analysis

As the author describes, there are many uses for real time Linux across industries. For example, in non-life-critical systems within automotives and aviation, including entertainment systems, Linux with the PREEMPT_RT patch has been utilized. In addition, it has been utilized in network devices like routers and physical simulators. However, in these cases reduced throughput is of concert. Real time systems are also gaining ground in High Performance Computing, where Linux has a near 100% market share. Overall, given the numerous applications and the cost and development savings of using Linux, the PREEMPT_RT patch has proved useful for making soft real time systems on Linux a reality.

## 2. Analysis and Conclusions

The article *The Real-Time Linux Kernel: A Survey on PREEMPT_RT* is a very good general overview. The review appeals to the educated reader by explaining potentially foreign concepts while moving quickly through current research. The topics were well covered, and good motivation for the review was provided, helping to engage the reader. All topics were very up-to-date, and included references to papers written within the last year. The topic was well presented and helped to stir an interest in further research into current real time system development. In terms of a description of an operating system, the paper did an excellent job addressing portions of the system that needed to be modified to support real time operations, including the scheduler, providing kernel interruption and coordinating IO. Overall the review meets its goal of educating and interesting the readers about real time Linux and the PREEMPT_RT patch.

## 3. Citation

[1] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2019. The Real-Time Linux Kernel: A Survey on PREEMPT_RT. ACM Comput. Surv. 52, 1, Article 18 (February 2019), 36 pages. https://doi.org/10.1145/3297714