

Real-Time Linux — Preempt-RT

General-purpose operating systems such as Linux are designed for optimal performance throughput and high flexibility. However, they normally do not guarantee deterministic timing. In applications where meeting strict timing deadlines is critical, unpredictable latency can lead to system failure. Real-time or deterministic behavior of applications needs to be supported by the operating system. While traditional real-time operating systems exist, they target smaller microcontrollers and not powerful application cores. Preempt-RT addresses this limitation and turns the Linux kernel into a real-time operating system. In other words, it guarantees that a Linux system reacts to an external event within a certain time. Linutronix was a driving force behind creating and mainlining Preempt-RT and is still actively improving it.

Historically there were two approaches to make Linux real-time capable:

- Linux runs on top of a microkernel
- Realize real-time behavior in the kernel itself

The microkernel approach — also called the dual-kernel approach — handles all real-time tasks in a separate RTOS. Linux is scheduled within this RTOS as a low-priority task. This approach might seem appealing at first but lacks substantial hardware support, good abstractions are hard to design, and it is difficult to use. The in-kernel approach makes Linux intrinsically real-time capable.

This implies that the familiar Linux API can be used, the broad hardware compatibility of Linux can be leveraged, and non-real-time and time-sensitive applications can run in parallel on top of a single kernel.

Since 2024, the Linux mainline kernel now includes the `PREEMPT_RT` option, so the Preempt-RT patch series that was maintained out-of-tree by Linutronix for years is no longer needed to get a real-time capable Linux system. The kernel already provides different preemption models that are optimized, for example, for desktop PCs or cloud servers. With Preempt-RT the additional model “Fully Preemptible Kernel” turns Linux into an RTOS.

Foundations

With the "Fully Preemptible Kernel" all kernel code is preemptible, except for a handful of carefully selected critical sections that must run to completion.

For this, interrupts are turned into threaded interrupts: the interrupt itself is acknowledged in a small, non-preemptible stub, and the bulk of the handler runs in a kernel thread that can be preempted just like any other user process. This allows the handler to sleep on lock contention and run scheduler-aware code without blocking other high-priority tasks.

Linux normally uses a fair-share, not immediately preempting scheduling algorithm. Real-time policies preempt immediately when a task with higher priority than the current one becomes runnable and run it until it voluntarily blocks.

Ordinary locks in the Linux kernel, such as spinlocks, disable preemption and block higher-priority tasks. For Preempt-RT spinlocks are implemented as sleeping spinlocks that do not disable preemption. Instead of spinning, a task attempting to acquire a contended lock disables CPU migration, donates its priority to the lock owner (priority inheritance), and voluntarily schedules out while waiting for the lock to become available.

Priority inheritance is used when a task blocks on a lock (e.g., `spinlock_t` or `mutex_t`); the lock holder adopts the waiting task's priority. For example:

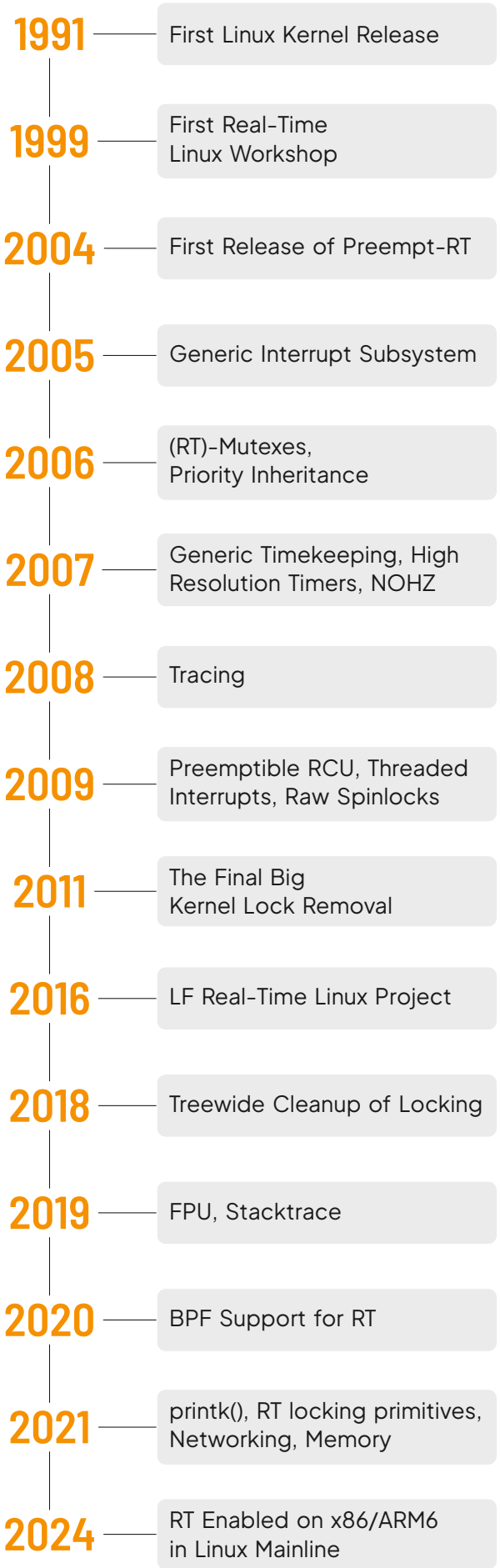
1. A high-priority task A blocks on a lock held by a lower-priority task B.
2. B inherits A's priority, runs to completion, releases the lock, and then returns to its original priority.

By replacing spinlocks with a `rtmutex` (preemptible, priority-inheritance aware) and enforcing threaded interrupts, the number of preemption points rises sharply, cutting the latency between a high-priority task becoming runnable and its actual execution.

Development over the Years

From the beginning of the development, Preempt-RT required extensive modifications across nearly every part of the system. Almost all subsystems — including interrupt handling, locking mechanisms, timekeeping, and memory management — had to be adjusted or redesigned. By the time the work was complete, every file in the kernel had been touched.

The following timeline represents the evolution of the development over more than three decades, highlighting major milestones. From early experiments to production-ready RT releases, to subsystem modifications until full integration.



Applications

One advantage of Preempt-RT is that real-time applications can be implemented entirely from Linux user space using standard interfaces. This means you do not need to write kernel or other specialized code to achieve deterministic behavior. However, there are several important considerations when developing Real-Time Linux applications as outlined below.

Scheduling	Use the correct scheduling policy and priority. Typically, the `SCHED_FIFO` or `SCHED_RR` policies are used, and their priority is set using `sched_setscheduler` or the `chrt` command. Also, avoid system level priority inversion dangers by disabling real-time throttling and the fair deadline server.
Isolation	Pin each task to a dedicated CPU core using `sched_setaffinity` or `taskset` to prevent interference when multiple real-time tasks run on the same CPU. Interrupts that a task depends on should be restricted to the same CPU using `smp_affinity` settings under `/proc/irq/IRQ-NUMBER/`.
Page faults	To avoid severe delays in real-time applications, all memory for the real-time task should be pre-allocated and locked into RAM. Stacks and heaps should be ”pre-faulted” by accessing all allocated memory.
Synchronization	Proper resource synchronization is crucial to prevent delays. Use `pthread_mutex` with `PTHREAD_PRIO_INHERIT` to prevent priority inversion.
Notification	Mutexes should be used with `pthread_cond` variables to ensure that high-priority tasks are woken immediately and without unnecessary delay once the lock is released.
Cyclic tasks	Periodic or cyclic tasks should use dedicated threads and absolute timers with `clock_nanosleep (CLOCK_MONOTONIC, TIMER_ABSTIME, ...)` for sleeping.

Hardware Considerations

Even with Preempt-RT and RT-optimized applications, modern hardware — designed to maximize average-case performance and minimize energy consumption through features such as CPU frequency scaling and complex cache architectures — can significantly compromise real-time determinism. While less complex systems intended primarily for embedded use cases, such as many ARM-based devices, can be easier optimized for real-time operation, modern Intel systems include a lot of features that must be tuned or even disabled to avoid unintended side effects. To simplify this setup, Intel introduced the TCC mode (for Time Coordinated Computing) that can be enabled in the BIOS for a good real-time baseline configuration.

Real-Time Performance Verification

Analyzing the real-time performance of the system is crucial to ensure it is performing within the specified boundaries. Primarily this involves identifying the maximum response latencies at various priority levels to provide a necessary understanding of the types of latencies that different components of the system could experience. Latency refers to the delay between when an event occurs and when the system responds to it. This includes the time taken for the hardware to signal an event, for the kernel to schedule the appropriate task, and for the task to complete its action. Reducing latency is critical for ensuring that Linux can meet the strict timing requirements of real-time applications, from industrial control to audio processing and robotics.

As mentioned before, Preempt-RT reduces the outliers in latencies by making most kernel code preemptible, converting interrupt handlers into threaded interrupts that can be scheduled according to priority, and improving locking mechanisms through techniques such as priority inheritance. These modifications allow high-priority real-time tasks to respond deterministically, even under heavy system load.

Cyclictest, part of the rt-tests suite, is the industry-standard tool for RT analysis. It operates by creating real-time threads (with `SCHED_FIFO` policy) that are periodically woken up and the measured difference between expected and actual wake-up time is recorded, representing the scheduling latency introduced by the kernel and system load. It often gives a good first impression of the real-time behavior of a system. Many further tools exist to verify certain properties of the system. For example, the RTC-Testbench `https://github.com/Linutronix/RTC-Testbench` is specifically tailored to assess real-time network communication, e.g., in Time Sensitive Networking contexts.

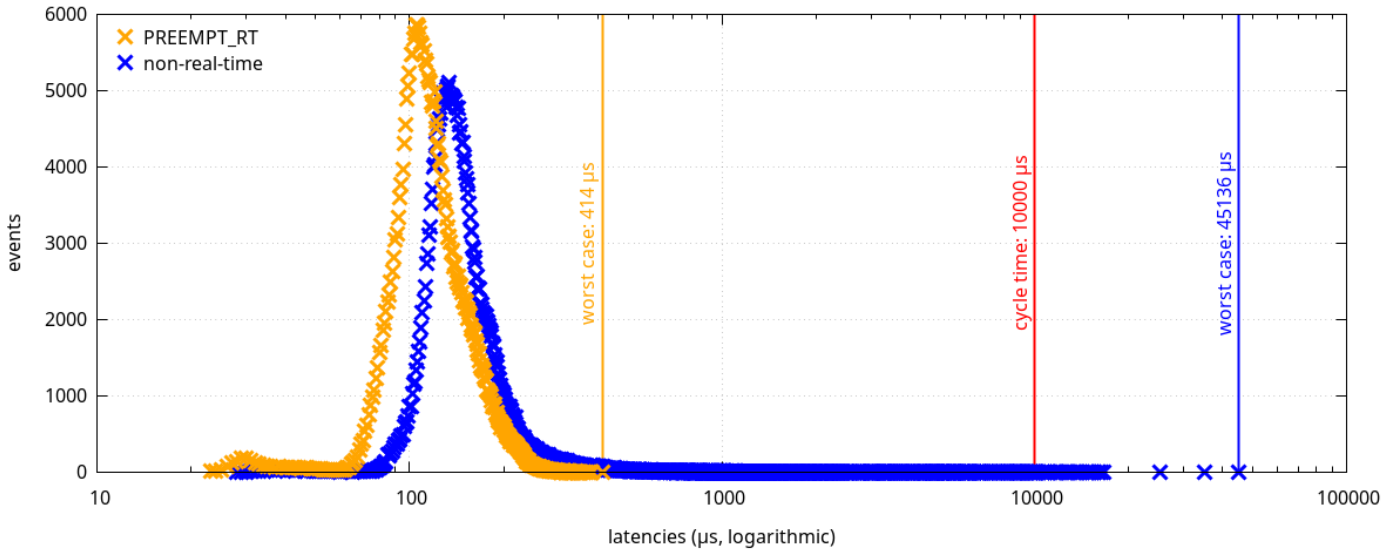


Figure 1: Linux Real-Time Application Latency Test Results

Figure 1 shows the latency (in µs) measured by a representational cyclic real-time application running on a PHYTEC phyBOARD-Pollux i.MX 8M Plus with a Debian trixie system. It compares identical conditions and tests using a kernel with Preempt-RT enabled and one without. In both cases the latency was measured under heavy system load. The Preempt-RT kernel reduced worst-case latency by a factor of 100+ (notice the logarithmic scaling) and brings the maximum latency down from totally failed responses of 450% the cycle time, to a maximum latency of only 4% the cycle time, even under heavy system load.

Linutronix Services

With Preempt-RT now part of the mainline kernel, many operating system vendors offer real-time capable systems. At Linutronix, however, you get unmatched real-time expertise from the very engineers who shaped the vision of Real-Time Linux, backed by experience in hundreds of industrial deployments.

- Custom Linux kernels with Preempt-RT
- Support with real-time application development
- Optimization of hardware for RT performance
- Analysis and remediation of latency problems
- Holistic real-time system design
- Support of real-time communication systems, for example with TSN or DetNet
- Comprehensive training for Preempt-RT delivered by the makers of Real-Time Linux
- Complementary offers such as our security-hardened and CRA-compliant operating system IGLOS

Conclusion

By carefully considering scheduling, CPU assignment, memory management, synchronization, and timers, Linux systems with Preempt-RT can achieve deterministic behavior suitable for real-time workloads, while still retaining the flexibility and rich feature set of Linux.

This also enables applications such as edge cloud scenarios where multiple real-time applications run in parallel as containers on a single cluster node or mixed-criticality systems using hypervisors to divide the CPU for a very flexible solution that can, for example, combine vanilla and Preempt-RT Linux with an RTOS like Zephyr and a bare-metal application.

With decades of expertise in Real-Time Linux, Linutronix is ready to guide you to the best solution — and to implement it side-by-side with you.

Linutronix GmbH
Bahnhofstraße 3 | D-88690 Uhldingen-Mühlhofen
Phone: +49 7556 25 999 0 | sales@linutronix.de
www.linutronix.de | Visit us

Are you interested?
Would you like to learn more about our products and solutions?
Simply contact us via phone or email.