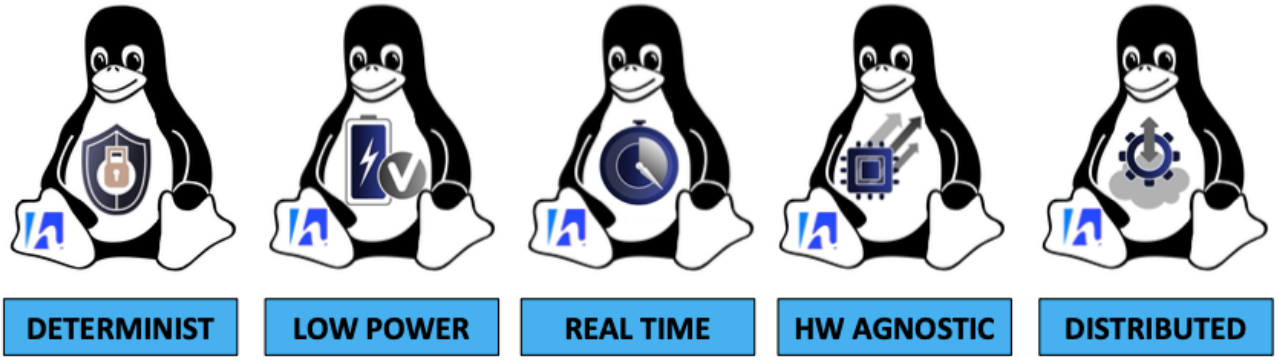
IOT 4.0 COMPUTING ENTAILS RECRAFTING LINUX WITH REALTIME & DETERMINIST CAPABILITIES

HyperPanel Lab



*HYPER*LINUX FUNDAMENTALS

LINUX is a highly popular general-purpose Operating System, but being UNIX based, it is not designed for real-time and deterministic behavior.

So workaround patches, primarily PREEMPT_RT, are gaining momentum from leading Distros as well as the Linux Foundation to further empower embedded LINUX use cases.

Alternatively, HyperPanel Lab is shaping a whole new co-kernel turnkey approach – code named *HYPER*LINUX - that aims at unleashing LINUX real time and latency performances, while achieving formal proof determinism and security by design.

*HYPER*LINUX shall then enable instant and no-code integration modes using turnkey daughter cards and reference designs.

LINUX IS NOT A REAL-TIME OS

Linux is a highly popular general-purpose operating system, but it is not designed for real-time and deterministic behavior. Conversely, because of the IoT computing revolution, there is an ever-increasing need for smart OSes with real-time capabilities, actually even beyond just industrial ecosystems such as automation, medical, data acquisition and measurement. For the purpose, Real-Time Operating Systems (RTOS) do provide a reliable, fast and highly deterministic reaction to external events, unlike regular General-Purpose OSes (GPOS), which provide a non-deterministic response.

In the meantime, Embedded Linux has eventually become one of the most widely used operating system for industrial applications, despite it is not implemented for truly real-time use. Linux is yet multithreaded, and it supports thread priorities, but the kernel is not preemptible, which causes performance issues, specifically on uniprocessor systems. Scheduling latency is also a concern in Linux, which makes it rather difficult to achieve determinism.

The main challenge in using Linux for real-time applications is about configuring the kernel. Developers must patch and customize the kernel to achieve real-time objectives, which can be highly difficult and thus require genuine and therefore rare expertise. So overall, while Linux is a popular choice for industrial real-time applications, it still entails additional work to achieve the level of determinism and low latency required by many real-time applications. That is the reason why Embedded Linux solutions with built-in real time functions and patches such as PREEMPT_RT are essentially commercialized by reputed tier-one Distros being RedHat (IBM), WindRiver, Canonical, Linutronix (Intel) or Suse.

MAKING LINUX REAL-TIME

Today, real-time improvement of Linux systems can be achieved through two main methods.

The first method is the dual-core approach, whereby a real-time kernel is added to the original Linux kernel in order to manage and schedule real-time tasks, whereas the original Linux kernel becomes the lowest priority task of this real-time kernel, meaning that normal Linux processes are preempted when a real-time task becomes ready to run, and the real-time task is executed immediately.

The second method involves directly modifying the Linux kernel itself, which beyond being incredibly complex, both to do and maintain, can then make it suitable for some sort of a general soft type real-time requirement (i.e.: subjective scheduling deadlines, depending on the applications).

Both approaches aim at reducing Linux latencies while providing a deterministic response time to external events. This makes the kernel better appropriate for applications in Industries such as aerospace, automotive, defense, IoT, robotics, telcos, public sector, and retail. Various “embedded Linux distros” are then designed to provide optimal compute and deterministic performance for such time-sensitive applications, while minimizing response time guarantee within a specified deadline.

To fulfil the objective, the real-time applications usually give themselves high priority, a lock in memory as well as a lock-free communication pipe, while avoiding non-deterministic I/O in order to execute a task within a suitably constrained system. Some system calls may even also require special privileges.

Such real-time versions of Linux are tailored to ensure proper and timely execution of selected applications and processes, making it better suited for monitoring IoTs and edge systems requiring quick response times.

In summary, real-time improvement of Linux systems is achieved through modifying the kernel or adding a real-time one to manage and schedule real-time tasks.

METHOD 1 – CO-KERNEL APPROACH

Real-time (RT) Linux is an operating system that was originally designed to provide sub-millisecond precision and reliable timing guarantees for real-time tasks. It works by running a small real-time kernel underneath Linux, with the real-time kernel having a higher priority than the Linux kernel. This allows the real-time operating system to never be blocked from execution by the non-real-time operating system and for components running in the two different environments to share data.

RTLinux was based on a lightweight virtual machine where the Linux "guest" was given a virtualized interrupt controller and timer, and all other hardware access was direct. From the point of view of the real-time "host", the Linux kernel is a thread. Interrupts needed for deterministic processing are processed by the real-time core, while other interrupts are forwarded to Linux, which runs at a lower priority than real-time threads.

Overall, RTLinux was a hard real-time operating system designed to support high-speed interrupt handling and other real-time tasks. It has been used in various applications, including embedded systems, automotive, and aerospace. Although WindRiver Systems acquired RTLinux in 2007 from its FSMLabs creators, it always comprised patented proprietary technologies not under GPL, and moreover, the last GPL version has been retired by WindRiver, so consequently, many RTLinux users had to switch to alternatives such as RTAI or Xenomai.

METHOD 2 – PATCHING THE KERNEL

The Real-Time Linux Preemption Patch (PREEMPT_RT) converts Linux into a preemptible kernel, thus providing faster response times and removing unbounded latencies. The big advantage of PREEMPT_RT over other RT-Linux implementations is that it makes Linux itself real-time, unlike other implementations that use a microkernel.

The patch was originally created in 2004 by Ingo Molnar, a "kernel hacker" known for his contributions to the Linux kernel and now working for Red Hat, and then maintained by Thomas Gleixner, whose Linutronix german company has been acquired by Intel in February 2022. In any case, PREEMPT_RT has become in 2015 an official Linux project, which means it is now within the scope of the Linux Foundation under the so-called Real Time Linux collaborative project. It aims at coordinating the efforts around mainlining PREEMPT_RT while ensuring that maintainers can continue development works, long-term support, and future research over real time.

PREEMPT_RT makes Linux itself real-time, which means that the programming model remains the same. With other real-time Linux implementations, a small micro kernel that runs like a hypervisor is created, and Linux runs as a task, requiring modifications to communicate with the microkernel. However, with PREEMPT_RT, if a program runs on the stock Linux kernel, it then runs on PREEMPT_RT as well. The maximum latency is not currently guaranteed by PREEMPT_RT.

ISSUES & OPTIMUN TO CONSIDER

Linux is a well-tuned Operating System for throughput-limited applications, but it is not designed, neither architected, to provide a deterministic response, which is a key requirement for any real-time application.

The response time of an application is the time interval from when it receives a stimulus to when it actually produces a result based on that stimulus. Nondeterminism is frequently caused by algorithms that do not run in constant time. For an Operating System environment to accommodate a hard real-time application, it must ensure that the application's deadlines can always be met, which implies that all actions within the Operating System itself must be deterministic.

Real-time application developers are mostly focused on interrupt latency, timer granularity, context-switch time, system call overhead, and kernel preemptibility. Linux does not implement interrupt priorities, and most interrupts are blocked when Linux is handling an interrupt, which is why the PREEMPT_RT patch has proven itself as an asset to Linux over the years.

However, in order to truly guarantee deterministic real-time performance, the underlying runtime system that executes the real-time operations must be completely independent from Linux. Isolation is therefore a key requirement to achieve reliable and secure operation of real-time applications alongside Linux.

A genuine real-time Operating System is therefore required to enable real-time behavior of Linux, and indeed, not all real-time extensions are equal. To ensure that user real-time tasks get privileged access to the processor when needed, an RTOS must either isolate the user real-time task from the kernel's own processing, or alternatively, permit the user real-time task to preempt most kernel operations. The latter is then called a preemptible kernel.

A fully preemptible real-time kernel allows specifying which process takes priority over previously uninterruptible Operating System processes, such as spinlocks, or hardware and software interrupts. With such a capability, one can then execute a time-critical operation ahead of all others without delay. However, just having a real-time kernel on its own will not necessarily make the whole system real-time, as the rest of the hardware and configuration must also be set up for it. That is why, again, customers who plan to use a Real Time patch for Linux have to be prudent to budget both for an expert and a Distro licensing cost from a company such as RedHat, Canonical, WindRiver or Linutronix, to help them out as otherwise, real-time could quickly become the gun to shoot yourself in the foot with

ANOTHER WAY TO STREAMLINE LINUX

Within the scope of the Internet of Things, it is already obvious that Software Defined Vehicles and Industry 4.0 automation are triggering the quest for hybrid OSes being both in a row an embedded RTOS (Real Time Operating System), as well as a computing GPOS (General Purpose Operating System). Equally challenging, Consumer based Smart IoT devices will also require - quite often within a rather limited hardware footprint – energy ultra-efficient OS based solutions with up to decades long seamless support and native security by design robustness to comply with both newer applicable regulations, as well as consumers' increasing concerns for a holistic digital protection.

For that purpose, HyperPanel Lab – a French based computer science company founded in 1988 - has created from scratch, over 25 years of self-financed R&D, a 100% sovereign and fully copyrighted whole new GPOS/RTOS breakthrough architecture.

The complete solution called HPL-OS4.0 comprises 2 layered containers being a software extension of the von Neumann hardware architecture to the Operating System itself. The upper layer deals with the applications, while the lower one manages the hardware – i.e.: thus handling both the I/O and the telecom stacks. The latter is therefore acting as a kernel having hands' on all the drivers, communication protocols and memory access. These 2 containers communicate in an asynchronous manner using an optimized messaging mechanism. They can therefore run side by side on any multicore processors, or separated on different physical IoT devices running as if they were one single object dynamically sharing hardware and functional resources at the edge, rather than just data and files as usual.

The technology is leveraging the Turing Machine principles as FSM (Finite State Machine) are implemented for all the drivers and telecom protocols which are then operated by the dedicated FSM engine. Likewise hardware IP blocks implemented in an FPGA or an ASIC, using an FSM based software architecture enables an ultra-efficient interruption- based monitoring of the kernel, thus allowing to drastically lower latency, while matching as close as possible the hardware fastest theoretical processing time.

HPL-OS4.0 suppresses all hardware to software latencies means that whatever hardware event arises (i.e.: an interrupt), its software processing is immediately started (interrupt handler). Then, the next processing steps (pipe) will also be started at the next CPU cycle. To achieve such instant performances, activity switching time has to be reduced by a ratio between 100 to 1.000. Experiments and computations carried on by HyperPanel Lab are showing between 400 % to 1.000 % benefits in terms of I/O throughputs increase when using the exact same hardware platform.

HPL-OS4.0 comes with 4 distinct levels of execution. This allows HPL-OS4.0 to be fastest than most incumbent RTOS, whilst still offering complete and advanced GPOS type functionalities such as the JavaScript interpreter and the HTML browser. These 4 levels of execution are spread into the two specifics containers hereinabove detailed – i.e.: the lower one overseeing the I/O and the telecoms, and the upper one being in charge of managing and rendering the applications.

The 4th level of execution is the GPOS itself. It is loaded in the upper container that de facto comes with its own dedicated real time control unit which can then be interfaced with any third-party GPOS such as Linux, based on POSIX API, thus allowing easy re-use of legacy Apps and software developments, while still offering real-time capabilities, lowest possible latency, as well as preemption and determinism.

Indeed, to ultimately achieve the best-in-class co-kernel possible implementation of Linux within HPL-OS4.0, it shall be necessary to streamline the Linux

drivers in order for the kernel to become an idle renderer of the user experience as well as a versatile API to run native Linux APPs, while leaving all I/O and telecom management down to the RTOS beneath.

HYPERPANEL LAB's 3-STEPS ROADMAP TO ENABLE SEAMLESS REAL-TIME MODE & DETERMINISM FOR LINUX

The dazzling success of the Raspberry Pi, Nucleo and Arduino boards is clearly emphasizing the industry growing appetence for ready to use off the shelves and nearly plug-n-play hardware-based kits. In other words, time to market and easiness of integration often comes down to the ability of leveraging a no-code stand-alone, kit based and stackable solution.

Based on this momentum, HyperPanel Lab has decided to enter the Real Time embedded Linux ecosystem, leveraging its hybrid 2 layers based RTOS, by means of a 3-steps roadmap.

- Step #1 – DUAL BOARD – a Stand-alone Nucleo type Board running HPL-OS4.0 onto an STM32 MCU, together with Linux running onto a separate Raspberry Pi type daughter card connected in ethernet with the base Board beneath.

- Step #2 – SINGLE BOARD / DUAL PROCESSOR – a Single board running both the STM32 MCU fitted with HPL-OS4.0 and a Broadcom or equivalent CPU running Linux, both OSes having still their own RAM. The boot and the Loader will be unified under the HPL-OS4.0 scope.

- Step #3 – SINGLE CHIP – MULTICORE INTEGRATION – a Single chip ASIC based or SIP implementation with in-depth intricating of the 2 OSes kernels, being HPL-OS4.0 and Linux, each one running on a different core of the CPU. Beyond the boot and the loader, the scheduler will also be unified by HPL-OS4.0, and if needs be, the memory could be shared between the 2 OS environments.

In any and all of the 3 above implementations, Linux will be acting as a renderer for the Applications, thus managing the screen output (which does not need real time, nor determinism). But all of the other I/O and telecom resources will be handled, in a truly determinist and real-time ultra-fast latency manner, by HPL-OS4.0.

In practice, as modern Linux device drivers are already written in two halves, the top one serving as an interrupt service routine, while the bottom half is executed in a kernel thread to complete the driver's work, HyperPanel Lab shall modify the relevant parts to make sure drivers' call for I/O and telecom resources are simply redirected to HPL-OS4.0.

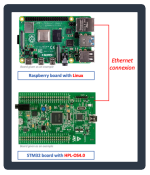
Doing so, HPL-OS4.0 will entirely take care of managing efficiently every resource Linux needs, thus easing the integration effort as Linux original Apps will be working straight away using the driver modified Linux kernel.

In the last and ultimate Step #3 implementation, the ability to run on a single multicore chip will require HyperPanel Lab to also take over the scheduler of Linux by replacing it. But even in that case, the application development environment will remain Linux, thus involving widespread tools like the compiler, linker, and debugger.

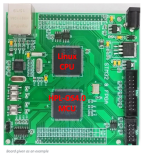
In a nutshell, HyperPanel Lab is planning to push forward the performances, stability and easiness of integration boundaries of Linux co-kernel Real time approach by first, a co-card solution, all the way towards a co-chip fully integrated solution.

That solution to boost, secure and make embedded Linux determinist leveraging HPL-OS4.0 disruptive kernel architecture, is code named ***HYPER*** LINUX.

HYPER **LINUX #1** - DUAL BOARD



HYPER **LINUX #2** - SINGLE BOARD



HYPER **LINUX #3** - SINGLE CHIP