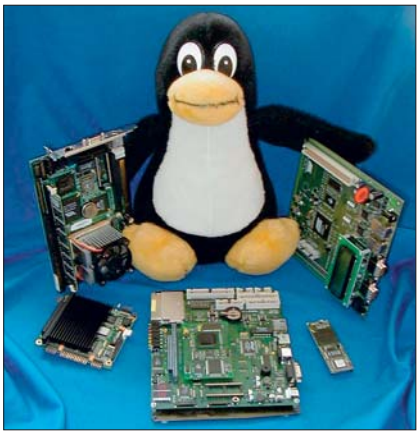
Next-generation hard real-time on POSIX-based Linux

by Robert Schwebel, Pengutronix

The POSIX standards make a good choice for building long-term solutions for industrial applications. They offer a framework which has already proved itself over more than twenty years. With the recently added pre-emption patch, Linux is fully able to provide a hard real-time POSIX runtime environment.



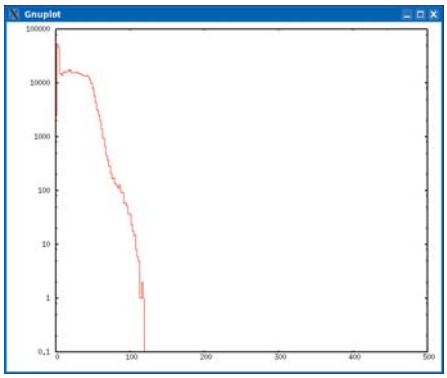
Industrial applications often need hard guarantees for system response times. Whereas operating systems on a standard PC usually try to be fair when giving out CPU time to the processes running on a machine, control applications cannot accept that, for example, an accidently running visualisation process may defer the execution of a high priority motion controller task for an undefined time span. Literature often differentiates between: hard real-time, soft real-time and no real-time operating systems. Historically, hard real-time (HRT) applications have often been developed with specialised hardware (microcontrollers, signal processors), being dedicated to one single task. On the other hand, commercial off-the-shelf PC hardware was never the best hardware platform for HRT applications: neither the old-fashioned interrupt architecture of the x86 processors nor the bus architecture made them the first choice for hardware developers.

However, there are three arguments driving developers into using standard PC hardware for industrial applications: 1) PC hardware is inexpensive, compared to customised industrial components; 2) the commercial power behind the x86 architecture is extraordinary and leads to a massive performance growth for modern PC processors; 3) HRT applications have an ever-increasing demand for communication to the outside world, and the PC is the platform with the most communication possibilities.

Over the last ten years it has become more and more evident throughout the industry that Linux would be a first-class player in the operating system market. Due to its open nature, Linux quickly became one of the best-supported systems, being commercially supported by most major IT and server companies. So it was no big surprise that, during the second half of the nineties, industrial users also noticed that Linux could be an enterprise-class operating system for control and automation applications. But as it was aimed at server and desktop scenarios, the Linux mainstream developers never came up with the aim to support HRT requirements. Nevertheless, there have been several pioneer projects which brought HRT scenarios to the open-source operating system.

Hard real-time solutions for Linux first came up during the end of the nineties, when Victor Yodaiken, a New Mexico Institute of Technology professor, and Paolo Mantegazza, professor at Politechnico di Milano, published their RT-Linux and RTAI patches to the Linux kernel. Both solutions were based on a "dual kernel" approach: the idea was to run a small HRT kernel with complete control over the interrupts "below" a standard Linux, so that the Linux kernel would get CPU resources only if there were no other high-priority tasks on the schedule. The dual kernel approach worked in several industrial real-life projects, so it quickly became the standard solution for developers who wanted to combine the communication skills of Linux with HRT requirements. The author was part of the RTAI development team for some years and successfully finished several industry projects based on RTAI.

Starting sometime around 2002, it became more and more obvious to several industrial RTAI developers, that the inner software structure, the maintenance model and the dual kernel approach in general would not be able to solve tomorrows, hard real-time problems. The software structure would probably have been a solvable task, taken that the core of RTAI – dating back to the DOS era – would have been rewritten from scratch. But the maintenance model couldn't easily be solved: the mainline Linux developers have often stated that no dual kernel approach would be accepted to be part of the "official" Linux kernel. The consequence was that the automation developers would have the long-term burden to let the RTAI kernel follow the rapidly moving Linux target. Linux is developing quicker than ever: since the 2.6.15 kernel was released at the beginning of 2006, more than 4000 patches have been integrated into the 2.6.16 kernel and that pace would have to be followed by the real-time folks.



Latency plot; x-axis: delay in µs; y-axis: histogram/number of events

Following an operating system which is developing at this high rate is each project manager's nightmare and an often-heard argument against using recent kernels in industrial projects. In reality things work differently: the people who actually develop a hard real-time extension must follow the latest kernel development, because the more they stay behind the top-of tree-activities the more they have to care about long solved problems and cannot participate in the high bug-fixing and feature improvement rate of the Linux kernel. Taking a project-centric view there always is the point where the kernel has to be frozen, but the usual strategy in an open-source-based industrial project is to do that due to project reasons, not because of some random decisions of real-time kernel extension developers. Due to the fact that the Linux kernel has several 100,000 testers worldwide, the latest kernel usually is the best one.

Another argument became more important in recent times: the investment goods industry builds machines with an estimated life span of 15 to 20 years, and the software part of the development cost is constantly increasing. Taking the fact that this software part also has to be maintained and supported for these 15 to 20 years, there is a high demand for finding a software framework able to survive that time span. Whereas probably nobody can imagine professionally supporting software technology from the eighties today (remember, that was the C64 and IBM-XT era), we now have software infrastructure that is able to provide a solid base for future development.

Since 1986, the POSIX (portable operating system interface for Unix, DIN/EN/ISO/IEC 9945/IEEE1003.x) standard provides a solid base for all kinds of industrial strength problems which happen to exist in today's embedded software. The standard describes the interface between the operating system and the application software. It is actively maintained by The Open Group, the latest revision is the 2004 edition of the IEEE1003.x standard series. All maintenance effort aims at fixing problems with the standard, without breaking compatibility. For timing-critical applications, POSIX has everything one might expect: a thread model with locking, timers with nanosecond resolution and mechanisms for asynchronous notification. As the POSIX standards have already survived 20 years of software innovation, it looks like a good base for industrial applications. And, by the way, the Linux kernel aims at being POSIX-compliant.

In 2005, Linux kernel developers Ingo Molnar and Thomas Gleixner revealed two advances to the kernel community: the "real-time preemption" and the "high resolution timer" patches. The idea behind their work is simple. Why should we add a dual kernel approach below the kernel when we can fix the reasons why Linux was not HRT up to now? If that approach worked, we had a Linux kernel which could run all kinds of "normal" userspace POSIX code under hard real-time conditions.

From the API point-of-view, Linux has all necessary interfaces. The problem is that the implementations behind the APIs add several restrictions to what the APIs provide: for example, POSIX timers have historically always been limited by the internal scheduler resolution of 10ms. So even if the programmer wrote a POSIX program and set up a timer with a period of 500µs, running at the highest POSIX real-time priority 99, that timer would probably have expired after 10ms – or even much later, if the system has other important things to do. So the task was to analyse the reasons for the restrictions of the Linux kernel, which is currently an ongoing task worked on by the core developers, supported by the rest of the Linux community.

In the meantime, the real-time preemption patch has made quite some progress. On the technical side, a recent RT-preempt patched Linux kernel has several advantages over a "vanilla" kernel: first it is fully preemptable. There are no long-lasting locks any more, so if the scheduler decides that a high priority POSIX task is to be run, the task switch happens without longer scheduling latencies. Because locks in scheduled operating systems always have the "priority inversion" problem (a high priority task may wait for a lock being held by a low priority task), Molnar and Gleixner developed priority inversion (PI) support for the in-kernel locking mechanisms, and, down that road, cleaned up the whole kernel locking infrastructure. Finally it came out that the timer infrastructure of the Linux kernel was optimized for timeouts, not for timers. Timeouts are usually optimized not to expire (think of a hard disc block read, where a timeout should almost never happen), whereas timers are optimized to expire. Because the overwhelming majority of timers in the Linux kernel have been used for timeouts, it was necessary to completely rewrite the timing infrastructure and separate out timers from timeouts. A side effect is that, after that cleanup, there is much better support for highly specialised timers which are for example found on common system-on-chip CPUs.

As of today, major parts of Molnar's and Gleixner's work have already made it into the mainline kernel, others will probably follow when the remaining issues the kernel community might have are resolved. These activities clearly show why the kernel development is so successful in the first place: in contrast to what the RTAI, RT-Linux and other communites have done in former years, this time the developers analysed the real reasons for design weaknesses and fixed their origins.

In December 2005, a group led by some large-scale machine building companies, seconded by hardware manufacturers and Linux technology companies formed the Open Source Automation Development Lab, OSADL. One of the aims of OSADL is to support the real-time preemption activities and to offer an infrastructure which can provide first-class support and test infrastructure for the automation industry, so that users from the industry can certify their hardware and software products against community-proven standard systems. ■