Embedded GNU/Linux and Real-Time an executive summary

Robert Berger
Embedded Software Specialist
Stratigou Rogakou 24, GR-15125 Polydrosso/Maroussi, Athens, Greece
Phone : (+ 30) 697 593 3428, Fax: (+ 30) 210 684 7881
email: robert.berger@ReliableEmbeddedSystems.com

Abstract

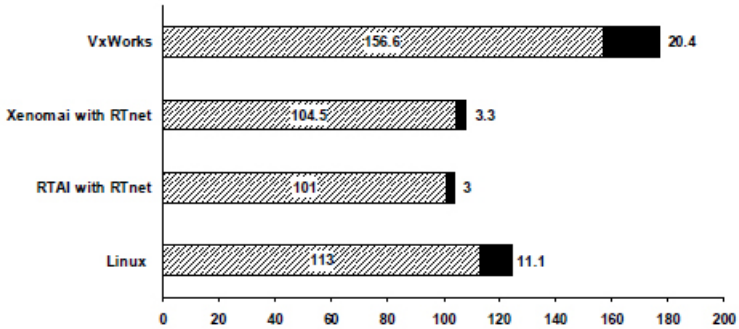
What is real-time and how can it be added to a plain vanilla Linux kernel[1]? Two fundamentally different approaches are described here. One approach attempts to patch the vanilla Linux kernel (PREEMPT_RT[2]) to achieve real-time functionality, while the other one utilizes a dual kernel architecture (RTAI[3], RTLinux/GPL[4], Xenomai[5], XM/eRTL[6], Real-Time Core[7], XtratuM[8], seL4[9], PaRTiKle[10],...), where Linux runs as the idle process on top of the real-time kernel. How do those approaches compare? At the time I started my quest for the holy grail of real-time Linux I hoped that it would be a clear decision wether to use p-rt or dk, but things seem to be less black and white than I initially thought. Bear with me to find out what changed my initial believes.

1 Introduction

I would like to thank Nicholas Mc Guire, Paulo Montegazza, Philippe Gerum[1] and Jan Kiszka from the dual kernel (dk) camp and Thomas Gleixner from the preempt-rt (p-rt) side as well as Paul E. McKenney and Carsten Emde for sharing their precious time with me to provide valuable input and to review this paper.

2 Real-Time

Real-Time has to do with time (timeliness), but most of all with determinism (predictability, time and event determinism). Paul E. McKenney defined it as follows: “A real-time system will pass a specified test suite“[11]. At first sight one might believe that a commercial hard real-time operating system always outperforms (real-time) Linux, but even this is not always true. Performance measurements on Gigabit Ethernet network communication have been carried out on VMEbus MVME5500 boards equipped with MPC7455 PowerPC processors[12]. Comparisons between the Linux IP stack, RTnet[13] and the VxWorks network stack have been performed. One board acquired 64 channels from AD converters via the VME bus and transmitted the data via UDP to another board which wrote data to a DA converter. Please note, that Linux under load would have had much worse performance, while the other systems were only minimally affected by workload. The graph shows that there are valid open source alternatives to VxWorks at least for specific cases.



	Linux	RTAI with RTnet	Xenomai with RTnet	VxWorks
Jitter (µs)	11.1	3	3.3	20.4
Delay (µs)	113	101	104.5	156.6

1 Philippe does not belong to any camp. He will maintain Xenomai dk until it proves totally useless, which, according to his understanding of the facts, is not the case so far.

3 Comparison of dual-kernel (dk) and PREEMPT_RT (p-rt) characteristics

3.1 Latencies

Wind River ran a couple of benchmarks[14] on a Dell D610 laptop with an Intel Pentium M processor to compare Real-Time Core, the Linux PREEMPT_RT patched kernel, and the basic Linux 2.6.27 kernel. The tests used default configurations and fully stressed the non real-time Linux system in order to measure scheduling jitter.

	average latency (µs)	worst case latency (µs)
vanilla Linux	44.000	7827.007
PREEMPT_RT	32.680	109.009
Real-Time Core	27.030	29.200

This shows the general believe that a dk solution always performs better than p-rt. The factor by which it performs better (in case it does) strongly depends on the test cases and the chosen architecture. Please note that p-rt might not be available on as many architectures and kernel versions as dk, so in some cases you cannot perform comparisons.

According to Carsten Emde [15] the peak performance of machines increased by a factor of 20000 in the last 30 years while the worst case performance (WCET) decreased only by a factor of 200 due to caches, shared IRQs, multiprocessing and the like. This means that processors 30 years ago were much better suited for deterministic real-time applications.

Real-time operating systems are usually written with those real-time friendly architectures in mind. Also dk utilizes naïve (naïve meaning simple not bad) locking schemes, which trade off minimal overhead on few CPU cores for scalability on multicore architectures. With sophisticated state of the art processors and from something like 8 cores onwards there is practically no performance penalty for p-rt. (In case you prefer to use dk on multicores you could always use only one or 2 to 4 cores for real-time and the others for non real-time applications. Care needs to be taken if 2 cores share the same caches like with the Intel Core 2 Duo and Quad architectures since the real-time behavior will be affected.)

3.2 Safety and Security

Concerning safety, Linux is nowadays certified to SIL 2[16,17]. The additional (real-time) kernel used by dk is much smaller than the full Linux kernel which might help with respect to certification. There are special dk approaches which target the safety domain and not real-time - see below.

Concerning security, we first need to get the timely correctness right. Once this has been accomplished a mechanism needs to be deployed, which protects applications in the “secure” domain from malicious Linux programs. With paravirtualization[18] (e.g. Xtratum 2 [19]) containers are utilized to protect the various partitions.

In general, systems nowadays have either massive security problems, or real-time problems without additional means/constraints.

3.3 Dual Kernel (dk)

Dk is driven by embedded and industrial communities and enforces by design a semantic separation between real-time and non real-time domains which makes it the preferred solution

wherever this separation is feasible. The real-time domain always has priority over the non real-time one, so funny things happening in the non-real time world have minimum impact on the real-time applications. In order to achieve maximum performance real-time programs need to run in kernel space, where less libraries than in user space are available. There are also limited debugging capabilities, although it might be easier to debug just the real-time application due to the separation mentioned above. The programming model for RTLinux/GPL and PaRTiKle is PSE 51 (Minimal Real-Time POSIX.13 System Profile) and XtratuM2 is based on ARINC 653 (Avionics Application Standard Software Interface) standards for design and BSSC 2000 Issue 1 which is the European Space Agency coding standard. Sometimes a new API needs to be learned to utilize dk, sometimes, like with Xenomai skins, you can use the real-time API that best fits your use cases and that you might already be familiar with. Please note that although POSIX is only used for a small portion of the real-time applications in the field you can use it through a Xenomai skin. (By the way - The main goal of Xenomai is to provide skins to move legacy real-time applications to Linux and this requires real-time capabilities, but the current solution being dk is just a by-product of it.) Your code needs to be licensed under GPL since it runs in kernel space, but you get hard real-time for it. If you go easy with OS sys calls user space programs can be written, which perform similar to those in kernel space and don't need to be under GPL. Besides you can choose to run applications, which require throughput but no real-time in Linux as usual and run your real-time stuff on the real-time kernel. Unless paravirtualisation will be used dk will most likely never be mainline which enables it to fix also those mainline latency issues which would be difficult to push to mainline otherwise, but at the cost for the maintainer of spending time to adapt out of mainline code every time a new kernel version is released. Nevertheless a broad support of kernel versions can be achieved like with Xenomai 2.5.0. It will support kernels from 2.4.25 up to 2.6.32+, which helps in the embedded world, since support for many architectures is still vendor driven and board support packages are not always compatible with the latest and greatest mainline kernel versions.

3.4 PREEMPT_RT (p-rt)

The idea of p-rt is to make the Linux kernel preemptive e.g. by replacing spinlocks by preemptible objects as well as making interrupts preemptible. By enabling p-rt on a single processor SMP bugs can be uncovered, which is an indicator that p-rt acts as a warning mechanism for future mainline kernel problems. It is very well suited for applications, where it's difficult to separate the real-time from the non real-time part. The standard programming model is POSIX which allows reuse of mainline drivers. (Some NPTL issues on multicores still need to be resolved). Since p-rt applications live in user space the same licensing applies as for all other Linux user space applications. It has the potential to be hard real-time, but due to the complexity of the Linux kernel and moreover due to the huge amount of modifications constantly being added and removed to/from it it might be very difficult to guarantee hard real-time while following the Linux way of working (use the latest). This would mean that every patched system would need to be reviewed for its real-time properties, which certainly requires additional expertise. Fortunately, the p-rt kernel is equipped with a number of built-in tests that facilitate the determination of the real-time capabilities of a given system[20]. In case a freeze on a specific "real-time blessed" kernel version is made p-rt should work pretty well, otherwise some people call this approach statistical hard real-time, since all possible execution paths would need to be checked to guarantee quality of service. There is not only interest from the embedded and industrial community for p-rt, but also from the enterprise side, since for some applications determinism is getting more important than throughput. The p-rt patch is more than 80% mainline and there are high hopes that some time around the end of 2010 it will be fully mainline. Nothing is to be disputed for this to happen, it's "just" about getting the bits and pieces into mainline and cleaning up some legacy stuff. So sooner or later p-rt will be the "standard" real-time Linux solution.

3.5 Real-Time Drivers

In case of stringent latency requirements real-time drivers are needed. Drivers which do not utilize the standard Linux drivers are written to the RTDM[21] specification and used for dk solutions (Xenomai, RTAI,...). There are only a few of those dedicated drivers available and they need to be maintained at project level, which is expensive and exposed to little testing compared to mainstream Linux device drivers, although it might pay off like with RTnet as mentioned above. P-rt does not directly support RTDM, but with Xenomai over p-rt those drivers could be used, although UIO[22] and threaded interrupts[23] are the “standard” vehicles for real-time drivers under p-rt with some constraints when shared interrupts[24] are used.

4 Future outlook

It’s always quite tricky to predict what the future will bring, since you never know for sure, but let me give it a try.

P-rt requires a recent glibc[25] with priority inheritance[26,27] enabled and will be used for mainstream and multiprocessor systems, but not in the safety domain. Initially especially people with not so hard real-time requirements (and that’s the majority) will embrace it. Later on, as it proves to be stable and deterministic, others will follow. Please note that there are many more kernel configuration knobs which might affect real-time behavior for p-rt than for dk, but board specific default p-rt kernel configurations might solve this stumbling stone as well.

Dk does not depend on user space features which enables it to support low-end architectures and uClinux[28] (MMU less like e.g. blackfin, nios2) as well as architectures for which no stable mainline real-time support is available either due to a lack of interest or architectural roadblocks as well as orphaned architectures which are not yet mainline. Dk is the only solution for the safety domain[29]. There will be a migration path from Xenomai and RTAI to p-rt, which will be released as soon as p-rt is fully mainline. This will give users a choice to experiment with different technologies and decide what best fits for their particular system.

5 Discussion

Since this paper was supposed to be an executive summary please forgive me for not touching the items mentioned above in sufficient technical depth and check the references for further information. I just wanted to give some overview over the existing technologies and show when to use what. In case only one solution exists the decision is clear. In case the performance and determinism between dk and p-rt is comparable it’s a matter of preference. Philippe Gerum said:”Today, there is absolutely no one-fits-it-all solution, and for a good reason: there is no magic bullet to make a complex Linux system real-time aware in 100% of its code base; doing so is extremely costly in terms of maintenance, especially with such a fast development cycle we have in the Linux ecosystem [30].” Remember also what Paul E. McKenney said:” A real-time system will pass a specified test suite”, which is not a definition for purists, but comes as close as it gets to practical real-life experiences. So what you need to do is to choose significant benchmarks for your system and exercise them for the real-time Linux solution of your choice. Only this will give your customers and yourself the confidence to deploy it with a clear conscience. For this to happen please keep in mind that it’s non trivial to come up with a good test suite that accurately reflects your application requirements and the more closely you are running to the capabilities of the platform, the more care you will need to invest and the more specific the benchmark will be to your application[31].

6 References

[01] http://www.kernel.org/
[02] http://rt.wiki.kernel.org/index.php/Main_Page
[03] https://www.rtai.org/
[04] ftp://ftp.rtlinux-gpl.org/pub/rtlinux
[05] http://www.xenomai.org/index.php/Main_Page
[06] http://dslab.lzu.edu.cn/mediawiki/index.php/Installation
[07] http://www.windriver.com/products/platforms/real-time_core/
[08] http://www.xtratum.org/
[09] http://ertos.nicta.com.au/research/sel4/
[10] http://www.e-rtl.org/partikle/
[11] http://www.rdrop.com/users/paulmck/realtime/paper/RTvsRF.2009.09.30b.pdf
[12] A.Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa and C. Taliercio; Performance Comparison of VxWorks, Linux, RTAI and Xenomai in a Hard Real-time Application
[13] http://www.rtnet.org/
[14] Wind River - Real-Time Options for Linux white paper (ask Wind River for it)
[15] http://www.osadl.org/fileadmin/dam/presentations/RTLWS11/cemde-path-analysis-vs-empirical-latency.pdf
[16] http://www.gmsystemsgroup.com/sil/sil_info_101.html
[17] Eur Ing R H Pierce MSc CEng MBCS, Preliminary assessment of Linux for safety related systems, ISBN 0 7176 2538 9
[18] http://en.wikipedia.org/wiki/Paravirtualization
[19] http://www.xtratum.org/getxmhpv.html
[20] http://www.osadl.org/Single-View.111+M57559f62c88.0.html
[21] http://www.xenomai.org/documentation/branches/v2.3.x/pdf/RTDM-and-Applications.pdf
[22] http://www.osadl.org/UIO.uio0.0.html
[23] http://lwn.net/Articles/302043/
[24] http://lwn.net/Kernel/LDD3/ - Chapter 10: Interrupt Handling
[25] www.gnu.org/software/libc/
[26] http://en.wikipedia.org/wiki/Priority_inheritance
[27] http://lwn.net/images/conf/rtlws11/papers/paper.10.html
[28] http://www.uclinux.org/
[29] http://www.xtratum.org/news.html
[30] Philippe Gerum of http://www.xenomai.org/ in a private email to me
[31] http://www.linuxjournal.com/article/9361

7 Author

Robert Berger is self employed, but also works as an embedded systems software manager for one of the largest multinational groups in Southeast Europe. He is part of the team that deployed 2007 the first commercial HD (high definition) H.264 Iptv solution in the U.S.A and 2008 the first commercial HD H.264 Iptv solution in Canada. He consults and trains people all over the globe and enjoys helping companies improve their development process and creating better embedded software. He is a project manager for the Linux Driver Project, IEEE lecturer for professional activities in the EMEA Region and speaker at various events on Embedded Systems like the Embedded World Conference and the Embedded Software Engineering Congress. He also moderates various Embedded Systems Professionals forums all over the world wide web.