Editor: Christof Ebert ■ Alcatel ■ christof.ebert@alcatel.com

Using Linux for Real-Time Applications

Armand Marchesin

An open source column talking about Linux? Aren't there enough such articles around? Indeed, tons of information are available on Linux. But that's not what we are presenting here. We've selected a topic for practitioners in the embedded and real-time domains, namely how to use Linux for real-time applications. The article is hands-on: it not only summarizes selection criteria and introduction schemes for RTLinux but also shows how Linux was actually integrated in an existing (legacy) architecture.

I look forward to hearing from you, both readers and prospective column authors, about this column and the products and tools you want to know more about. If you'd like to write for this column, see the author guidelines or contact me. —Christof Ebert

For small and medium-sized businesses to stay competitive, they need communication systems that offer not only excellent telephone service but also Internet and data-handling capability. As a solution to this problem, we at Alcatel have built a comprehensive e-communication appliance called OmniPCX Office, or OXO—a preconfigured server integrating data, Internet, and voice communication capabilities.[1] The system offers a cost-effective voice solution including IP telephony as well as secure, high-speed, shared Internet access, built-in email, security and control applications, and Internet-based remote-access facilities. OXO has been in the field for nearly three years.

Rather than using proprietary software to support the OXO platform, we chose Linux with an RTLinux extension. This column explains why we chose it and how OXO uses it, focusing particularly on architecture and real-time aspects.

Why Linux/RTLinux?

The ever-increasing complexity of devices and systems, the ever-accelerating pressure for a shorter time-to-market, and reduced development costs necessitate using rich software platforms. As we began to develop the OXO system, it quickly became clear that our existing real-time, kernel-based infrastructure was inadequate to support all the expected services. We needed a standard, general-purpose software platform such as Linux or Windows to take advantage of the available applications. With the exception of telephony, this would cover most of the system's requirements.

OXO belongs to the *hard real-time* systems category; this means that whatever load stress the system is experiencing, it must respond to an event within a predefined time. For example, low-level Integrated Services Digital Network (ISDN) signaling protocols demand responses to inquiries within precise (and generally very short) time limits.

Unfortunately, Linux and Windows can't handle hard real-time tasks; they can usually respond to events within a predetermined time, but this isn't always the case. The technical literature refers to this as *soft real-time*. Linux experiences this phenomenon for several reasons, mainly because it can't preempt certain kernel operations.

Getting the determinism you need

There are several ways to make Linux achieve the required level of determinism. Some approaches involve creating opportunities to run the scheduler more often, thereby minimizing the delay between an event's occurrence and its processing.[2] Although this improves the system's responsiveness for interrupts and tasks, Linux is still not hard real-time; critical sections of code can produce long latencies. Other approaches modify the system's original design in ways that jeopardize compatibility with standard Linux and thus with the existing applications base, negating all expected benefits.

Another is the dual-kernel approach.[3] Think of this as a simple real-time executable rather than a full-blown operating system, in which the lowest-priority task (in the executable sense) is Linux as a whole (see Figure 1). The real-time executive takes control of the machine, mainly by intercepting all the interrupts that Linux normally handles; the system immediately services the interrupts that are meant for real-time. The executive then reassigns the interrupts to Linux only after the system has processed the real-time tasks. In this way, applications requiring real-time processing (like managing signaling protocols) can cohabit with other, less demanding applications (such as voice mail or Internet access) on the same machine. Moreover, Linux remains almost intact, thus offering compatibility with the current and future applications base.

There are, however, problems with this approach. One issue is that real-time tasks actually share Linux kernel space, making debugging tricky. Another major drawback is that real-time tasks can't directly access Linux services, for the same reasons that prohibit Linux processes from behaving deterministically. If real-time tasks could use Linux services directly, they might preempt some of the nonpreemptable operations I mentioned earlier, leading to an unstable system. This required us to design the system in two separate (although collaborating) worlds: "pure" real-time tasks with no access to Linux services, and non-real-time applications with full access to Linux. Communication resources such as shared memory or FIFOs allow interchanges between these two worlds.

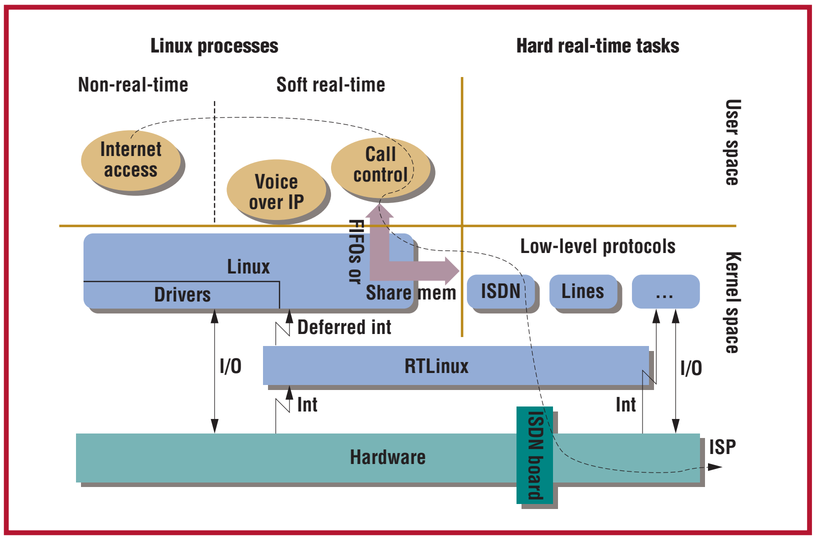

Figure 1. The OXO software is divided into three classes: non-real-time and soft real-time Linux processes in user space and hard real-time tasks in kernel space. The RTLinux executive shares the kernel space as well.

Author Guidelines for Future Columns

Select one open source software component, tool, product, or product group that you've used. Focus on why you chose it, what its benefits are, how it contrasts with competing components (OSS or commercial off-the-shelf), and what lessons learned might aid other practitioners. For your column to be useful, your report must come from in-depth experience rather than a superficial, one-weekend evaluation.

You may review a single tool or product (such as a GNU or Eclipse item) or a vertical product group (such as defect-tracking tools, middleware, or workflow management tools).

Evaluate the OSS components in a concrete, fairly broad application context and present the information in a neutral style. If authors follow a similar style, readers will have easier access to the information.

Preferably, you are an OSS user rather than a primary author or key contributor. Typically, authors don't work for an independent software vendor or packaging company. You can't be zealous nor hostile toward OSS, as this would bias the evaluation and reduce credibility.

Present feature comparisons of different OSS products and other data in chart format, if possible.

Send your column proposal and author qualifications to Christof Ebert at christof.ebert@alcatel.com.

Table 1

Comparing Windows and Linux for voice and data services

Service	Windows + RT extension	Linux + RT extension
Voice services	Sufficient	Good
Data plus Internet services	Very good	Very good
Software development environment	Good	Sufficient
Support	Good	Good
Costs (including hardward impact)	High	Low

Commercial extensions such as RTX from Venturcom and InTime from TenAsys can help enhance Windows so that it can support hard real-time operations. Both systems use the dual-kernel approach. However, we had already developed an almost equivalent product based on Windows NT and commercial applications. This product ended up with an excessive end user price due to the expensive hardware platform that was necessary and the global cost of software. Also, making some of our proprietary hardware run under Windows control was very difficult.

Because Linux is open source, we had better control and could adapt the entire system to our specific needs. Linux is also efficient, robust, modular, and configurable—all important characteristics for mission-critical embedded systems. Moreover, it's royalty free.

Table 1 compares Windows and Linux features for implementing voice and data services. Although they are basically equivalent, we found it easier to develop Linux drivers for our proprietary hardware, thanks to source availability. The software development environment is better for Windows, and both systems offer good support. But when it comes to royalties, software licenses, and the cost of the hardware resources necessary to run the software platform, Linux clearly wins.

Two open source real-time Linux extensions were available: Real-Time Application Interface and RTLinux.[3] For historical reasons, we chose to use RTLinux for OXO.

Implementation

To fit with the dual-kernel approach's rules, we divided the OXO software into three main classes of tasks on the basis of timing constraints: hard real-time, soft real-time, and non-real-time (see Figure 1).

Its hard real-time tasks are RTLinux based, with strict, short deadlines and mainly network or terminal signaling protocols. Actually, these protocols—which are among the software's most delicate parts—were already isolated in the existing system. Moreover, the system had implemented them as finite state machines, making the port straightforward and avoiding the huge debugging effort we feared.

OXO's soft real-time high-priority Linux processes include call control, voice over IP, the H.323 or Session Initiation Protocol, and so on. These processes have flexible deadlines and can live with things like delayed tone generation.

OXO's non-real-time low-priority processes are Linux based, with no real-time constraints. These include mainly Internet applications such as Internet access, email, firewalls, and so on.

Figure 1 shows an example of a non-real-time Linux application that provides Internet access through ISDN. The application asks the call control function (soft real-time) to control the ISDN signaling protocols (hard real-time) in order to set up a connection with the service provider.

Today's communication systems require advanced services that proprietary platforms can't provide without considerable development effort. Consequently, standard platforms have become essential. However, these platforms usually lack the real-time support that such embedded systems generally require.

Although it doesn't support real-time natively, with help from available extensions and careful design, Linux provides an effective way of building such systems. The recommended reading in the sidebar will give you more detailed information.

Recommended Reading

- *Building Embedded Linux Systems*, by Karim Yaghmour, O'Reilly, 2003, ISBN 0-596-00222-x, 391 pp., $44.95.
- *Linux for Embedded and Real-Time Applications*, by Doug Abbott, Newnes, 2002, ISBN 0-7506-7546-2, 256 pp., $33.99.
- *Linux Device Drivers*, 2nd ed., by Alessandro Rubini and Jonathan Corbet, O'Reilly, 2001, ISBN 0-596-00008-1, 586 pp., www.xml.com/ldd/chapter/book, $39.95.
- *Linux embarqué* [French], by Pierre Ficheux, Eyrolles, 2002, ISBN 2-212-11024-3, 326 pp., €39.
- "Books on Embedded Linux," *Linux Devices.com*, 2004, www.linuxdevices.com/articles/AT2969812114.html.

References

1. "Alcatel OmniPCX Office," *Alcatel*, www.alcatel.com/smb/Pages/Products/OmniPCXOffice/index.htm.
2. C. Williams, "Linux Scheduler Latency," *Red Hat*, Mar. 2002, www.linuxdevices.com/files/article027/rh-rtpaper.pdf.
3. *Real-Time Linux Foundation*, 2004, www.realtimelinuxfoundation.org.

Armand Marchesin is a software architect manager in the Software Department of the Enterprise Solution Division, Private Communication Group, at Alcatel. Contact him at Armand.Marchesin@alcatel.fr.