Linux inside your TV?

Ruud Derwig

The Internet has enabled 'open-source' software development, a process that uses independent peer review and rapid evolution of source code to improve reliability and quality. Open-source licenses allow software modification, (re)use, and redistribution, giving users perpetual, full access to software, independence, choice, and control. This 'collective' ownership motivates people to contribute to the code base. The GNU/Linux operating system is an example of a successful product of the open source community. During the past years Linux has gained a strong position in the (Internet) server domain. Whether Linux and other open-source software is an option for consumer electronics products like television sets, DVD players and set-top boxes is the topic of this article.

Introduction

Consumer electronics equipment found in people's homes today contains more and more software. The functionality and complexity of home products increase steadily and the size of the software needed to realize these products is growing exponentially. Whereas a few years ago the sizes of embedded TV and set-top box software were measured in tens or hundreds of Kbytes, today's high-end products contain Mbytes or tens of Mbytes. And all these Mbytes have to be filled by (expensive) programmers. Wouldn't it be nice to obtain large parts of that software from third parties, and wouldn't it be even more nice to get it for free? This promise - and a dose of Microsoft antipathy - is what lead to many designers and companies in the consumer domain taking an interest in Linux and open-source software. In order to discuss the possibilities of developing consumer products with Linux inside, first we present a number of trends and associated features that will be introduced in new products the coming years. Furthermore, some of the important requirements of the high-volume electronics domain are explained. Then we delve into a number of technical issues that are relevant for deciding on the feasibility of Linux inside your TV or other consumer product. Next to the technical issues, however, it might turn out that non-technical issues like support, licenses, and development process are even more important. After discussing these non-technical issues this article is concluded with the answer to the question: Linux inside your TV?

Tomorrow's products

Trends

The Internet is entering the living room. Although not spreading as fast as people thought - or hoped - some years ago, new features that require Internet connectivity are being introduced right now. Many commercials being broadcast at Dutch television already present a URL pointing to a web page with more information about a certain product. Instead of running to the study, booting your PC, trying to remember the URL, typing it into your browser and finally getting to the content, wouldn't it be nice to push a button on your remote control and get the information on your TV screen? Several digital TV service providers already have a web browser integrated into their high-end digital receivers. But web browsing on TV is not the only feature that needs an

Internet connection. A network connection can also be used for getting data that is not web-related like e.g. the information needed for an electronic program guide. Or it can be used for streaming audio or video content, like for instance in an audio set supporting not only traditional AM and FM bands, but also IM - Internet Modulation - for receiving Internet Radio stations.

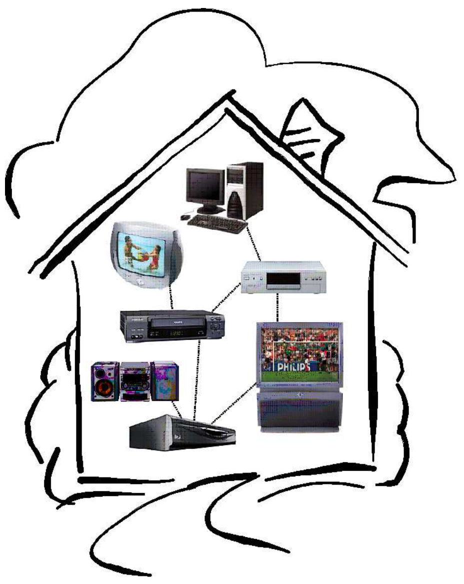


Figure 1: Connected consumer devices form in-home networks.

The Internet is not the only network entering the home. Whereas today devices are operating stand alone, with at most a cable connecting the VCR or DVD player to a TV, in the future devices in the home will be connected - either wired or wireless - to form in-home networks. Instead of always having to watch satellite channel movies in the living room, because there the set-top box is connected to the TV, the set-top box could redirect the movie through the home network to the bedroom and the bedroom TV could forward remote control commands back to the set-top box. The same scenario applies to other devices like for instance a storage device. While watching the forwarded movie in the bedroom, for instance, you get too sleepy to continue watching. Instead of getting out of bed, sleepwalk to the VCR and program it, a storage device connected to the in-home network could be controlled from any place in the house. Although a fully connected home network is probably not going to be there in the near future, the first digital receivers supporting a second TV will enter the market soon.

Another development that is important for the Linux discussion is the (r)evolution of hard-disk technology. Storage capacity of hard-disks is growing exponentially, even faster than Moore's law. Using compression, it is possible to store about an hour of high quality video in a Gbyte. This means that a 60 Gbyte hard-disk can store 60 hours of video. Because hard-disk bandwidth is high enough to support several video streams simultaneously, a hard-disk enables new features like a pause button for live broadcasts. The first hard-disk based video recorders are available today.

The last trend we mention is the fact that consumer systems are becoming more open, like PC platforms. This means that new, third party applications can be downloaded and executed on existing products. An example from the digital video broadcasting domain is the Multimedia Home Platform (MHP) standard. MHP defines a Java based execution platform for interactive applications. Next to a subset of standard Java APIs it defines a number of APIs to control the digital receiver features of a set-top box, like tuner, service information database, and video decoder. MHP applications range from simple teletext like information services as a stock ticker to e-commerce solutions for home shopping.

Requirements

From the trends and features described above a number of new functional product requirements can be derived. Future products must support various forms of networking and connectivity. Furthermore, various mass storage media - both optical and magnetic - and file systems must be supported. But besides these new functions that seem to make consumer products more PC like, the traditional requirements from the high-volume consumer electronics domain still hold. People expect a high level of robustness and ease of use from TVs and DVDs. Frequent user reboots of a TV, because of system crashes, are no option. Given the huge product volumes and strong price erosion, the bill of material of mainstream consumer products is under constant pressure. This translates into a limited CPU cycle budget and a limited memory footprint that make efficient resource management very important. In combination with the required robustness and the real-time nature of audio and video, this leads to the need for efficient, deterministic, predictable CPU scheduling and memory allocation. Traditionally, small and efficient real-time kernels were the only solution for meeting these timeliness and predictability demands in a cost effective, resource constrained way. These small kernels like CMX, pSOS and VxWorks, however, do not offer the rich set of features that ”fat” operating systems like Windows and Linux do. Since future consumer products will depend more and more on a rich set of networking, connectivity, and storage features, the ”fat” operating systems are entering the consumer products market.

Technology

To what extend can Linux and other open-source software meet the requirements of today’s and tomorrow’s products? To answer this question we discuss three topics: real-time performance, memory footprint and functionality.

Real-time performance

The standard Linux kernel provides soft real-time support according to POSIX 1003.1b. A priority based preemptive scheduler is available, with scheduling latencies varying on a standard PC from a few μs to 100 ms or even more. Although average latency is usually very low, responses in the order of 10 ms occur frequently. Compared to for instance 20 ms deadlines for a 50 Hz. video frame rate, it is clear that - unless a lot of expensive buffering is applied - plain Linux cannot cope reliably with the requirements from the video domain. This does not mean that you cannot play DVD movies on a standard Linux PC. Such a standard PC uses several Mbytes for buffering audio and video. And even then, when starting a web browser or receiving an e-mail while playing the movie, on many occasions the video is not displayed smoothly at a constant frame rate. Therefore, from a real-time requirements perspective, the only way to build a robust consumer device based on a standard Linux distribution is by solving all real-time requirements with extra hardware. This can either be more memory for buffering, but a more reliable strategy is adding an extra processor that takes care of the real-time control tasks. In such a dual processor system the processor running Linux can be seen as an application co-processor for performing the non real-time best effort application tasks.

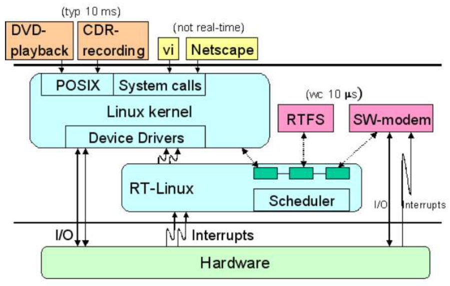


Figure 2: Real-time Linux, a hybrid solution.

But Linux is open-source. So for each problem or lacking feature somewhere in the world someone can be found working on it. Although this is not true for all lacking features, it does hold for the real-time performance of Linux. And - following another open-source tradition, the Darwinian survival of the fittest - it is not a single person or project that is providing a solution for improving the real-time performance of Linux. A number of (sometimes competing) alternative solutions exist. These solutions can be classified in two groups, either enhancing the real-time performance of the Linux kernel itself or enhancing system performance by combining Linux with a small real-time kernel.

Enhanced Linux

The greater part of the large worst-case scheduling latency of standard Linux is caused by its monolithic kernel design. Although user space activities are scheduled preemptible, kernel space activities are not. This means that operations like system calls, ”bottom half” interrupt handlers and process scheduling are completed once started, even if an other higher priority activity is triggered (e.g. by an external interrupt) and ready to be executed. To reduce preemption delay in kernel space two solutions are being worked on. The first one, called low-latency patch, shortens the non-preemptible kernel paths by introducing explicit reschedule points inside the kernel. The second one, called preemption patch, exploits kernel extensions for supporting symmetric multi-processor architectures (spinlocks). Instead of allowing preemption at specific points inside the kernel, like the low-latency patch does, the preemption patch enables preemption for the complete kernel space, except for specific critical sections that should not be entered by more than one thread concurrently. Most of these critical sections are already protected by spinlocks for the multi-processor version of the kernel. By implementing the spinlocks with mutexes in the single processor build - instead of skipping the spinlocks like the standard kernel does - the kernel becomes re-entrant. Of course the complete solution is somewhat more complex than described here, but a detailed discussion of both patches is not the goal of this article. Although it is difficult to give accurate worst case scheduling latency numbers for both patches - numbers depend on different versions of the patches, different hardware and different benchmarks - it is generally assumed that both patches can reduce maximal latency to sub ms numbers in the order of hundreds of μs.

Another approach to enhancing the real-time performance of Linux targets the scheduler. The standard Linux scheduler is optimized for throughput and fairness, not for real-time responsiveness and predictability. Although the scheduling policy for application processes can be set to the traditional priority based preemptive scheduling offered by real-time kernels, the implementation of the Linux scheduler still can be improved for better and deterministic performance. Other improvements to the scheduler include new scheduling policies like the budget based scheduling found in resource kernels.

Hybrid solutions

RTLinux and RTAI are two approaches to achieve hard real-time behavior, generally referred to as 'real-time Linux'. Essentially, these projects are constructing their own real-time kernel. This kernel runs Linux as its lowest priority task. All the standard services of the Linux kernel and Linux applications are available (although without the real-time guarantees), yet real-time threads and handlers can run with minimal, hardware limited, latencies. Real-time Linux solutions can guarantee interrupt latencies of several μs and scheduling latencies in the order of 10 μs The major drawback of these solutions is that the improved real-time performance only holds for the real-time kernel part. Standard Linux device drivers do not become real-time drivers by using a real-time Linux variant. Drivers may disable interrupts for up to hundreds of μs and violate other standard real-time design rules. In order to use those drivers in the real-time domain, they have to be rewritten.

Footprint

Although the memory available in consumer devices is growing, memory remains a scarce resource. The main reason is that the larger part of the Mbytes that are available in high-end digital products are used for buffering and rendering audio and video data. Also graphics consume more and more memory. Besides a framebuffer (sometimes doubly buffered) a considerable amount of memory is spent on built-in fonts and bitmap graphics. Finally, new features like a web browser or Java virtual machine have a large impact on the remainder of the memory that is not used for audio and video data. Traditional real-time kernel sizes are in the order of 100 Kbytes or 10 Kbytes for small kernels. A minimal desktop Linux system requires several Mbytes of memory, without a graphical user interface like X. Of course not all libraries and modules that are needed on a desktop PC are needed on a TV or set-top box. By selecting specific features and disabling others, standard Linux distributions are reasonably scalable. Some commercial embedded Linux distributors even provide special tooling for configuring and scaling down Linux. However, assuming Linux is chosen in favor of a real-time kernel because of its rich set of features, it is fair to state that a minimal embedded Linux solution requires several Mbytes of memory.

And that is just the kernel. When adding applications and services like a desktop window system, web browser and Java virtual machine, total memory requirements are in the order of tens of Mbytes..

But Linux is open-source. Just like on the real-time performance, several projects are working on reducing the memory footprint of Linux based systems. We name two examples here.

The standard GNU C library that most applications dynamically link to requires about 1.3 Mbytes. That's a lot of memory, especially if only a few functions are used - printf does not always make sense inside a TV. When linking statically to the library, code that is not needed can be stripped, resulting in a minimal size of about 300 Kbytes. But then, each application duplicates this code. Several small and scalable C library implementations, like 'uClibC', 'DietLibC', and 'newlib' have been or are being developed for embedded applications, reducing the memory requirements to several Kbytes.

Graphical user interfaces and window systems that are used in standard Linux versions do not meet the requirements of the embedded consumer domain. Besides a large footprint, standard solutions are not optimized for interlaced TV displays or small LCD screens. There are many open-source (and commercial) projects that deal with those issues. Again, we name two examples. The 'MicroWindows' project works on a small modern graphical windowing environment for embedded applications. It supports an API based on Win32 GDI, with a footprint below 100 Kbytes. 'Qt/Embedded' is a solution that is scalable from 800 Kbytes to 4 Mbytes. It aims, among others, at consumer devices like set-top boxes and PDAs.

Functionality

The technical motive for choosing Linux is not its real-time behavior or memory footprint. It is the rich set of features that ease development and enable a short time to market. Linux supports most of the new functions that we see entering consumer products in the coming years. It is a fully featured, high-end operating system that supports multiple processors, processes and users, that supports networking and many file systems, and that provides memory protection and security options. Apart from generic services and implementations of standard protocols, specific support can be found for a wide range of processors and peripherals, including standard (wireless) network cards and IDE drives, but also including a/v consumer domain specific peripherals like (digital) TV cards, IEEE 1394 high speed digital interface, and BlueTooth devices. Additionally, since sources of all drivers are open, the effort required to realize a new driver for not yet supported hardware is less than it would be when starting from scratch.

Next to the features that are built into the Linux kernel, many utilities, middleware and applications exist. Although most of them are targeted at desktop or server systems, a large number of them can be put to account for the consumer domain because of the convergence of PC and consumer functionality. Furthermore, because of the big momentum for embedded Linux solutions, more and more middleware and applications become available that are specifically targeted at embedded devices. One just has to take a quick look at a site like www.linuxdevices.com to realize this. We mention a few examples that are relevant for the consumer domain. Many projects deal with web browsing on embedded platforms. Both open-source solutions, like 'viewML', 'Konqueror/Embedded', and commercial solutions like the 'Opera' web browser that runs on top of the 'MicroWindows' environment are available. Other open-source projects and commercial companies are building (digital) TV receivers including electronic program guides and hard-disk video recording on top of Linux. Activities range from open-source initiatives like the 'vdr/LinuxTV' project - developing 'personal video recorder'[2] software -, via associations like the 'TV Linux Alliance' - standardizing platform interfaces -, to companies like 'TiVo', that offers a personal video recorder service and set-top box on subscription basis.

Figure 3: The TiVo personal video recorder.

[2]Personal video recorder is the phrase used for indicating hard-disk based video recorders that support functionality like a pause-button and automatic recording of all broadcasts of your favorite soap series.

Non-technical issues

How free is free software? Throughout this article, we deliberately use the term open-source instead of the term free software that is commonly adopted too. The free part of free software refers to the freedom towards users that is ensured by licenses. And although the business models associated with open-source software are usually not based on paying for the actual software and intellectual property that the software represents, free software is not free of cost. In order to use open-source software in software intense high-volume electronics products, for instance, professional support and training are important too. Today, the support you can get from companies specialized in embedded or real-time Linux is comparable to what you get from traditional real-time kernel vendors. Quality is good and prices are fair, among others because of the competition between different suppliers that do not hold technology locks on their customers with specific real-time solutions or tools.

Another cost factor that people tend to forget when talking about free software is licensing. Since there are several important licensing related issues, it deserves a section on its own.

Licenses

The idea of open-source software dates back to the software-sharing community of the MIT Artificial Intelligence Labs. When this community dissolved, one of these people, Richard Stallman, continued to write what he called free software. He later formed the Free Software Foundation that is responsible for many of the GNU applications. Several open-source licenses exist nowadays, like e.g. the GNU General Public License (GPL) and Library (or Lesser) General Public License (LPGL), the BSD or Berkeley license, and more recent variations such as the Netscape and Mozilla Public Licenses and the Sun Community Source license. The term 'copyleft' is also used, especially with the GNU licenses, because the central idea is to give everyone permission to run, copy, and modify the program, and to distribute modified versions. It is, however, not permitted to add restrictions. With many of the licenses modified versions of the software must, upon redistribution, provide the same freedom to users, including availability of the modified source code. One of the differences between these licenses is in the terms for combinations with proprietary software. For example, linking proprietary closed-source applications with GPLed libraries is not allowed, but linking them with LGPLed libraries is. Linking binary modules into the Linux kernel is generally allowed due to an explicit license exemption granted by Linus Torvalds.

The availability of modified source code that GPL and GPL-like licenses require, has both advantages and drawbacks. It enables open-source communities and projects and has helped Linux to become the feature rich operation system that it is today. For high-volume consumer electronics manufacturers it means lower bill of material costs, since no run-time license fees are due. But it can also mean that the manufacturer's specific intellectual software property and added value is no longer protected when specific enhancements and additions to the copyleft licensed source code have to be opened up. And when this intellectual property is also protected by patents, the situation becomes very unclear from a legal point of view. By incorporating GPL licensed software into a product, the manufacturer grants users the right to freely use and distribute the manufacturer's modifications and additions. On the other hand, patent laws disallow the free use of these modifications and additions when they are protected by patents.

Because of the fact that open-source licenses have never been tested in court, companies that consider incorporating open-source licensed software into their product should make a trade-off between the time to market and cost advantages and the legal intellectual property risks. Next to this legal risk, also the public image of the company should be considered. In the spirit of freedom of use and sharing, the open-source community has little respect for companies that only use software for their own commercial benefit. Furthermore, the principles and ideas behind open-source software are not in line with traditional patent protection. On the one hand many members of open-source communities feel that software patents, that restrict the free use of software, hinder innovation. On the other hand, patent-minded people feel that the protection and possible financial rewards stimulate companies to invest in innovation. Without proper protection of intellectual property rights, the huge research and development investments that are needed for breakthrough innovations would not be affordable.

To overcome some of the disadvantages of GPL, companies like Sun have introduced semi-open-source licenses to exploit the open-source advantages while simultaneously protecting their intellectual property and taking benefit from community-constructed add-ons. The dual licensing of Qt's graphical windowing environment is another approach. It can either be used free of charge, but protected by GPL, or companies can choose to commercially license the same software that can be combined with own applications without restrictions.

Process

The final subject we discuss does not concern the products of open-source, but the way these products are developed. Given the fact that traditional development processes do not scale up very well to cope with the large team sizes that are needed for complex products, open-source development processes can be an interesting alternative.

Re-use of software is often seen as the solution for coping with the increasing complexity and strong time to market requirements. Software re-use, however, turns out to be very challenging in practice. It is very difficult to take an arbitrary software component from one product and put it to use in another product. Reasons vary from strong context dependencies of a component, quality problems and lacking documentation, to architectural mismatch. One way of eliminating these obstructions for re-use, is to centrally enforce a common architecture and common processes for documentation, quality control, etc. However, when development projects grow in size - a modern, high end TV requires over 100 man years of software - or are executed over different locations and time zones, the overhead of centrally enforcing and checking the architectural and process rules grows exponentially, if possible at all.

The other way of dealing with the re-use obstacles is the open-source way. The distributed nature of open-source projects, with many contributors that communicate through the Internet, scales much better than a centralized approach. The basic idea is very simple. When programmers can read, redistribute, and modify the source code for a piece of software, it evolves. People improve the code, adapt it, and fix bugs. And this can happen at a speed that, in comparison to the pace of conventional software development, may seem astonishing. Several companies are experimenting with open-source like development processes. Either through projects that are truly open to the public community, or through so-called 'inner-source' projects that aim at creating and leveraging communities inside a company. Main challenge for these experimental development processes and organizations is to find the right balance between centrally managed and distributed development activities. Without some central guidance and direction, software will not evolve into the right (commercial) directions.

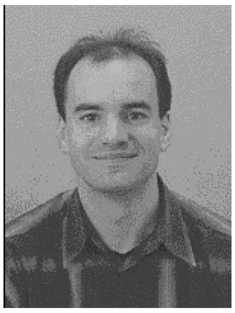
Conclusions

Linux is gaining a lot of momentum in the consumer electronics domain. Although real-time behavior and memory footprint do not yet allow Linux to be used in all mainstream consumer products, the rich featuring makes Linux a serious candidate for today's and a very serious candidate for tomorrow's high-end products.

Using Linux and other open-source software can be very tempting from a technical and time-to-market point of view. The freedom that open-source advocates, however, is mainly freedom to end-users and does not necessarily match with the intellectual property business interest of consumer electronics manufacturers. Given the legal uncertainty - open-source licenses have never been tested in court - attention must be paid to reducing the risks, for instance by avoiding linking to GPL software or only linking binary modules into the Linux kernel.

Open-source software influences the consumer products of tomorrow. If not by being incorporated into products, then by adopting certain aspects of the open-source development process that promotes sharing and re-use of software. This leads us to our final conclusion. Linux inside your TV? Probably sooner than you think!

Ruud Derwig (Ruud.Derwig@philips.com) is working at Philips Research on software platform architectures for resource constrained products in the consumer electronics do-



main. Key areas of expertise are real-time kernels and operating systems, resource aware component architectures, and heterogeneous software architectures. Before joining Philips Research he followed the post-masters Software Technology program (OOTI) at the Eindhoven University of Technology.