Embedded/Real-Time Linux Survey

Alain Mosnier
July 2005
Version 0.3

1 Executive summary

The aim of this survey is to get a snapshot of the Linux embedded and real-time markets today, and to some extent, its outlook for tomorrow. The type of information present in the report is a blend of market and technical information, though never deeply technical.

In chapter 4 “Questions to be answered in this report“, page 10, we give a list of questions that this report aims to answer. We give hereafter summarized answers to those questions:

- Why choosing Linux as an embedded/real time OS? What are the advantages and drawbacks?

Different embedded system suppliers will find different advantages and drawbacks to using Linux depending, among others, on their own experience and the kind of applications they have. This is why we talk in this report about potential advantages and drawbacks.

Linux is for sure attractive in many ways, including for embedded and real-time applications. However, the arguments of its detractors are relevant and do need to be assessed by embedded system suppliers who are considering Linux. Some issues, like cost and licensing, can be seen as both advantages and drawbacks, which makes them complex issues that need to be addressed carefully.

Some popular reasons of choosing Linux are: cost, adaptability and configurability, portability.

Some often heard reasons not to choose Linux are: legal ambiguity, total cost of development, Microsoft has better alternatives.

- Who is using Linux today as an embedded/real time OS? Why? What is the outlook of Linux as an embedded/real-time OS?

More and more embedded suppliers are using Linux, although few hard-real time applications are running today (but interest for FSMLabs RTLinux seems to be increasing rapidly). Linux can be found in most embedded categories: PDAs, mobile phones, VoIP phones/devices, robots, audio/video devices, thin client devices, tablets/webpads, telecoms, vehicle industry, test and measurement, payment, point of sales, electronics and entertainment, industrial, automation, and process control.

From figures available on the Internet, and not limited to Linux supporters, we can find that up to 20% of the embedded system suppliers are using Linux today, and this figure is increasing. Microsoft has a comparable market share. The reason for this Linux popularity can probably be found in the reasons to choose Linux that we mentioned earlier.

- What are the available distributions for embedded/real-time Linux, including grow-your-own alternatives? What are the advantages, drawbacks and markets shares of the different alternatives?

It is difficult to draw any general conclusion about market shares, since we have a single source of information for those figures. Popular target distributions include: debian, home-grown, OpenEmbedded, Redhat, Montavista, BlueCat Linux, RTLinux, Wind River Platforms Linux Editions, K-Linux, ELinOS, RTAI, DENX.

Advantages for certain types of applications can be found to certain distributions (Carrier Grade Linux distributions for telecoms applications for example).

Home-grown distributions have the advantage of being flexible and adapted, and the drawback of lacking committed support.

- What is the process of building a development environment for a Linux embedded/real-time system?

We can summarize the different steps as follows:

  - Specify the requirements
  - Choose the target hardware
  - Choose the source for the target software
  - Choose the host system
  - Connect the host to the target
  - Install and configure the cross-development environment
  - Creating a target root file system
  - Configure the permanent storage

- Licensing: what obligations are linked with GPL-like licenses

Basically the obligations are the following:

  - For applications running under Linux: no obligation if they cannot be considered as including GPL code or be derived from GPL code, which should not be a problem for most applications. This also applies to applications that link to unmodified versions of LGPL licensed libraries (GNU C library for example).
  - For Linux devices drivers, and more generally Linux loadable modules: they have to be licensed under GPL, since they can be considered as a derived work from the Linux kernel. Some vendors don't respect this, but they have no guarantee not to be sued by the copyright owners of the Linux kernel code some day.

However, licensing issues are complicated and there are many other licenses than GPL and LGPL among open software products. Please refer to a lawyer in case of uncertainty, and assess the license issues before starting any development.

- Pricing: estimated real cost of Linux vs. cost for other embedded/real-time operating systems.

We cannot pretend that we really have given an answer to that question, given the complexity of the picture on the Linux side, and the usual confidentiality of pricing information, not to mention the large ability to negotiate prices and discuss price models that every customer in this business has.

We did make clear that Linux cannot be assumed to be less costly than other operating systems from a total cost perspective during the whole life cycle.

This total cost needs to be estimated based on, among others: the cost of the development seat, training (depending on the experience already available), support and maintenance from the OS/tool supplier(s), cost of developer's time, cost for royalties both for the OS and other potentially necessary modules.

- Which kernel to choose: 2.4 vs 2.6?

Although many performance improvements have been brought into v2.6 compared to v2.4, those improvements happen to be mostly targeted at large systems.

Some tests at DENX software engineering in Germany have proved that the changes in v2.6 actually decrease performance for systems with limited resources, like most embedded systems.

Quite many embedded Linux vendors still have v2.4 available in their products, and at this point, v2.4 seems like a better choice for most embedded systems with limited processor speed, RAM and L1/L2 caches.

Table of Contents

2 Introduction

The aim of this survey is to, based on information available in books and on the Internet, get a snapshot of the Linux embedded and real-time markets today, and to some extent, its outlook for tomorrow. Given this aim and how fast this market is moving, the reader should be aware of that the conclusions in this report will not remain valid very long.

4Real AB being operating system independent, this reports intends to be as objective as possible.

The type of information present in the report is a blend of market and technical information, though never deeply technical.

The audience for the report is people who are considering Linux as an alternative for embedded systems, people who are advising embedded system suppliers, or just people who are curious about Linux in the embedded market.

To some extent, this report is gathering in a single document information already present on various sites on the Internet. Quoted text is indicated between double quotes (“) and is inserted with the permission of their authors and with an indication of the source.

Throughout the report, references are mentioned with a number between brackets (for example: [1]). This number refers to the list of references present in chapter 15 “References“ page 38.

Chapter 3 “Definitions“ defines terms and acronyms used throughout the report.

Chapter 4 “Questions to be answered in this report“ tries to define what this report should deal with as clearly as possible by listing the questions it should answers. Hopefully, these are the topics that the audience of the report is wondering about.

Chapter 5 “Embedded vs real time systems“ defines what we mean respectively by “embedded systems” and “real-time systems”.

Chapter 6 “Linux as an embedded/real time system“, comments usually mentioned advantages and drawbacks of Linux as an embedded or real time operating system.

Chapter 7 “Applications using embedded and real-time Linux“, gives recent and hopefully representative examples of applications using embedded and real-time Linux. A lot of it comes from [4].

Chapter 8 “Today's market and outlook“ looks at the market for Linux compared to other operating systems in embedded and real-time systems.

Chapter 9 “Distributions“ compares the most popular embedded and real-time Linux distributions, including home grown distributions.

Chapter 10 “Process for the development of a Linux embedded system“ summarizes the process that one needs to go through in order to set up a Linux embedded system development environment.

Chapters 11 “Licensing“ and 12 “Total cost of development“ deal with traditional areas of concern when using Linux as an embedded operating system.

Chapter 13 “Which Linux kernel to choose?“ gives some information about the differences between kernel v2.4 and 2.6 and associated performance considerations.

Chapter 14 “Linux for telecoms applications“ presents some standards that have been developed for an important class of embedded Linux systems: telecommunication systems.

3 Definitions

Term or acronym	***Meaning***
ATCA or AdvancedTCA	Advanced Telecommunication Computing Architecture
CGL	Carrier Grade Linux, see chapter 14 “Linux for telecoms applications“ page 35
CRAMFS	Compressed ROM FileSystem: a compressed read-only file system for computers with limited storage capacity.
Cross-compilation or cross-development	Since host and target (see those terms) computers often are different computers with different architectures, the embedded applications need to be compiled on one architecture, for another architecture. This requires a specific configuration of the development tools. This is what we call cross-compilation or cross-development.
DHCP	Dynamic Host Configuration Protocol, used by a client in order to obtain a dynamic TCP/IP configuration from a server
Earliest Deadline First	A popular real-time scheduling algorithm
Host, Target	In the context of an embedded system, we call target the embedded computer itself, as opposed to host, which is the computer on which the embedded application is developed and compiled. Since the target often has very limited resources, host and target are often two different computers with two different architectures.
JFFS2	Journaling Flash File System, 2. A popular file system for flash persistent memory.
OS	Operating System
OSDL	Open Source Development Labs
PICMG	PCI Industrial Computer Manufacturers Group
PXE	Pre-Boot Execution Environment. Can be for example used in order to force an embedded computer to acquire a TCP/IP configuration via DHCP and boot over the network from a TFTP server.
Rate Monotonic Scheduling	The most famous real-time scheduling algorithm
Telecoms	Telecommunications as an industry
TFTP	Trivial File Transfer Protocol. Can be for example used in combination with DHCP and PXE boot (see those terms) in order to load the operating system on an embedded computer from a remote server.

4 Questions to be answered in this report

This report will try to answer the following questions:

- Why choosing Linux as an embedded/real time OS? What are the advantages and drawbacks?
- Who is using Linux today as an embedded/real time OS? Why? What is the outlook of Linux as an embedded/real-time OS?
- What are the available distributions for embedded/real-time Linux, including grow-your-own alternatives? What are the advantages, drawbacks and markets shares of the different alternatives?
- What is the process of building a development environment for a Linux embedded/real-time system?
- Licensing: what obligations are linked with GPL-like licenses
- Pricing: estimated real cost of Linux vs. cost for other embedded/real-time operating systems.
- Which kernel to choose: 2.4 vs 2.6?

5 Embedded vs real time systems

5.1 Embedded systems

Usually agreed characteristics for an embedded system are:

- It is a **special-purpose computer system, which is encapsulated in a device**, and has a specific purpose, unlike a general use computer.
- Embedded systems are expected to **run continuously during long periods of time without errors**. Most embedded systems avoid mechanical moving parts such as disk drives because they are less reliable than alternatives like flash memory.
- Embedded systems **may be outside the reach of humans** (launched into outer space for example), and must must therefore be able to restart themselves in case of major failure. This is usually the role of a piece of hardware called a watchdog timer, that reboots the system when the timer has elapsed (to avoid this, the system itself needs to periodically reset the timer).
- Often, embedded systems are run in **tough environments** (humidity, vibrations, etc.) and need to be rugged.
- Often, a **low production cost** is an important requirement for embedded systems (car industry, mass market products, etc.). This, in turn, implies that the available resources on the target platform, hardware and software, has to be limited to what is strictly required by the application.

As a consequence of the previous requirements, the target system often has neither hard disk, nor keyboard/mouse or screen. The application code, and device drivers if required, are usually developed and compiled on a different computer called "host computer". This technique is called cross-compilation or more generally cross-development. The binaries are then uploaded to the target, often in flash memory. When the systems starts, it expands the OS and the application in RAM.

Embedded systems often have real time requirements but need not to.

5.2 Real-time systems

Finding a rigorous definition of real-time systems is not as easy as one would think, since there are still discussions among specialists about what this term should mean.

However, many of these specialists seem to agree that the following definitions more or less apply.

Hard real-time system [1]: “A [hard] real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred.” An ABS car breaking system and an airplane autopilot system are examples of hard real-time systems.

Soft real-time system: a system that has time constraints, but for which occasional timing misses are acceptable. The average response time has to be within bounds, but a worst case response time guarantee is not required (best effort approach). A codec in a telephony system and a playback client for a streaming video application are examples of soft real-time systems.

Hard real time systems don't necessarily have to be fast but they have to be predictable (in practice they however have to provide good performance, but there are trade-offs to be made, as we explain below). They usually have to guarantee a certain **latency** and **jitter**. The latency is the delay after which the system has to provide the result of a computation in response to a triggering event. The jitter is the delay variation between several occurrences of the event and resulting computation. A hard real-time system guarantees that the computation will always provide a result after a delay comprised between (latency - jitter) and (latency + jitter)[1].

While soft real-time systems may have their requirements satisfied by general purpose operating systems that show a performance which is good enough for their application, hard real-time systems need guarantees that general purpose operating systems usually cannot provide. General purpose operating systems can present the following limitations:

- The task scheduler's overhead increases with the amount of tasks running on the system. In such a case, response times guarantees can only be given if the total amount of processes on the system is limited to a known maximum.
- The OS kernel is not preemptible. When the OS is running a system call on behalf of a user process, it cannot be preempted it in order to run a higher priority real-time task (priority inversion issue).
- The kernel can disable interrupts for a time which has no upper bound. During that time, that can be arbitrarily long, a real-time request for service won't be taken into account.

However, real-time performance requires compromises on other aspects of the

1 The concepts of latency and jitter for real-time systems are not consequently, or even always, defined in the specialized literature. However, we feel that these simple definitions give a good summary of hard real-time constraints. In particular, only guarantying that the delay will be lower than a certain limit is not sufficient in many cases. For example, an airbag system on a car should not inflate too early, since it has to deflate within short, in order to let the driver regain control of the vehicle. If it inflated too early, there would be a risk that it would deflate when the driver still is in danger of hitting the steering wheel.

operating system. As the following table shows, general purpose and real-time operating systems have different objectives [2].

Real-Time OS	General purpose OS
Optimize worst case	Optimize average case
Predictable schedule	Efficient schedule
Simple executive	Wide range of services
Minimize latency	Maximize throughput

Those objectives are often conflicting. For instance, minimizing latency requires more context switching, and therefore more overhead, which can decrease throughput.

Real-time operation systems use special scheduling algorithms that allow them to guarantee a certain latency. **Rate Monotonic Scheduling** is the most famous real-time scheduling algorithm. It allocates a task priority which is proportional to the frequency of occurrence of the task's triggering event. Another popular real-time scheduling algorithm is **Earliest Deadline First**, where the task with the closest deadline always has the highest priority [37].

Real-time systems don't have to be embedded systems but they often are.

6 Linux as an embedded/real time system

6.1 Potential advantages

According to [3], embedded system suppliers have the following reasons for considering Linux:

- Low cost: 68%
- Adaptability/extensibility: 57%
- Personal control of features: 43%
- Avoids commercial alternatives: 38%
- Performance: 37%
- Memory requirements: 13%

Rather than going through these, we choose to comment the reasons we mostly hear about when talking to customers.

6.1.1 Many processor architectures are supported

At least the following process architecture are supported: **x86, ARM, PowerPC, MIPS, Hitachi SuperH, Motorola 68000**.
This makes the system portable and allows reusability of code between different systems.

6.1.2 Source code is available

This gives the application developer a complete control over the functionality, down to the OS level, and allows the adaptation of modules already developed by others, instead of starting the development from scratch.

6.1.3 Modularity and configurability

At the kernel level, one can choose which modules to use depending on the hardware to support, and recompile and adapted kernel according to a well documented and proved procedure. This allows to run Linux in systems with as little as 2 MB persistent storage and 4 MB RAM.

This approach is also valid for the higher software levels, where one can for example choose not to install a graphical interface if it is not necessary.

6.1.4 Quality and reliability of the code

This is mainly due to the community model development of the Linux kernel and other open source products used on Linux based systems. This model invites many parties to identify problems, debate possible solutions, implement fixes, and review the source code.

6.1.5 Support for devices, availability of communication protocols and tools

The large adoption of Linux has lead to the development of drivers for many devices, most communication protocols and many tools. Most of the programs running under UNIX have been ported to Linux. A large number of drivers are maintained by the Linux community itself, which makes the users less dependent on the device vendors.

6.1.6 Licensing

There is no license fee for Linux. The distribution vendors can only charge for the media they provide, possible extensions that they have made, and additional services. This is definitely an advantage over the sometimes very expensive licenses associated with other embedded operating systems. However, suppliers of embedded systems based on Linux need to have a very clear licensing strategy, if they want to charge for the software they provide and protect their potential patents and copyrights. This will be dealt with in chapter 11 “Licensing“, page 30.

6.2 Potential drawbacks

According to [3], embedded system suppliers have the following reasons for not considering Linux:

- Incompatible with existing code: 48%
- Performance or real-time capability: 27%
- Support: 26%
- Memory usage: 23%
- Development tools: 19%
- Legal ambiguity: 13%
- Cost: 8%
- Other: 17%

Rather than going through these, we choose to comment the reasons we mostly hear about when talking to customers.

6.2.1 Licensing

Open Source licensing is sometimes seen as a drawback since providers of Linux based systems can be worried about being forced to loose control over their copyrights and patents. However, there are ways to avoid these issues. This implies to get a full understanding of the implications of the Open Source licenses. This will be dealt with in chapter 11 “Licensing“, page 30.

6.2.2 Total cost of development

While Linux enthusiasm is often associated with the idea that "it is for free", a more reasonable approach is to list the actual costs associated with the choice of different operating systems during the lifetime of a product.

Indeed, while many Linux components, including development tools, can be downloaded for free, building a whole development environment in this way can cost a lot of time both for startup and maintenance. Many companies developing Linux embedded systems prefer to choose an off-the-shelf distribution and buy maintenance and support services from the distribution vendor.

We will present conclusions from total development time comparisons in chapter 12 “Total cost of development“ page 32.

6.2.3 Difficulty to choose a Linux distribution

One argument against Linux is that choosing Linux is just the first step in making a choice.

With a commercial operating system, a company developing embedded systems can just send an order to the vendor after the choice is made. They will usually receive a standard development environment, with hopefully clear instructions to install it and get started with the development.

When choosing Linux, one has to go through additional steps in the choice process:

- Build a development environment from individual packages from the Internet. If that choice is made, one will have to choose each individual component or group of components, install them, and probably test them individually and together (kernel, development tool chain, C library, standard Linux tools, free BSP or individual drivers, etc.). Or,
- Choose a distribution vendor. If that choice is made, one will have then to select a distribution, depending on several parameters: costs (tools, maintenance, support, etc.), level of support required, features and performance, etc. Depending on what vendor is selected, the rest of the procedure might be more or less similar to choosing a commercial operating system.

This choice process will be even more complex if the system has some hard real time requirements. Some of the embedded Linux vendors provide real-time performance, whereas others don't, and there are also several real-time Linux open source alternatives.

It is clear that choosing Linux requires more expertise during the choice process than choosing a pure commercial operating system. Linux offers the largest choice, but choosing costs time and expertise.

6.2.4 Real time performance concern

The official Linux kernel available at kernel.org is not a hard real-time operating system kernel. There are several alternatives to obtain a Linux based real-time operating system.

We will give more information about these alternatives in chapter 9 “Distributions“ page 22.

6.2.5 Linux is an uncertain world

Compared to a purely commercial operating system, support for Linux is split between a large numbers of companies and organizations. Certain projects, organizations, and the support for certain modules can be discontinued, and "free of charge" implies "no warranty".

Embedded systems suppliers can get the impression that although a lot of hardware is supported and many applications are available, they will never know whether the modules they need really will work individually and together until they try them for themselves. If the modules work together, they might feel uncertain whether some of the modules will be supported in the future or not.

However, companies adopting Linux do have the choice of ordering a complete development package with support and maintenance from Linux distribution vendors. Those vendors can give a clear specification of the features, performance and support that they provide. Of course, the choice is more complex with Linux than with a single vendor system.

6.2.6 Linux can be more expensive and is more complex to learn than Windows

We will come back to comparisons between total costs of development for different systems in chapter 12 “Total cost of development“ page 32.

Microsoft and Linux based operating systems are both complex worlds. However, Microsoft puts a lot of efforts into simplifying the process of development of new products, including embedded systems. The tradition of graphical oriented development tools in Microsoft is older than in the UNIX and the Linux world. Since there are many different tools from many different suppliers under Linux, it is probably easier for a company to get external help on Microsoft's systems than on Linux.

Also, Microsoft's enthusiasts will say that: Microsoft based development is easier to integrate in a usual office environment, Windows CE and Windows Embedded are now stable operating systems, Visual Studio .NET includes embedded features that are easy to integrate, and a lot of software is available for Microsoft's operating systems.

These are relevant arguments, and embedded system suppliers need to take them into account when evaluating their requirements against the different OS alternatives. Once again, Microsoft, if considered as a serious alternative, has to be compared in a fair manner with Linux distribution vendors, based on the most important criteria for the embedded systems supplier (total development cost including royalties, learning curve for the developers and impact on the time to market, etc.).

7 Applications using embedded and real-time Linux

7.1 Embedded product categories

Embedded products using Linux fall into the following categories. Each category is illustrated with examples, as recent and demonstrative as possible [4].

- **PDAs, handhelds**
  - 2005. Archos PMA400 Pocket Media Assistant. 320x240 color LCD, 30GB hard drive, WiFi, Qtopia PIM suite, MPEG-4 video playback, etc.
  - 2004. Sharp Zaurus SL-C3000 based on a 416MHz Intel XScale PXA270 processor. 16MB of ROM and 64MB of SDRAM. 4GB hard drive. Trackball-like pointing device. 3.7in 640 x 480 LCD touchscreen, SD and CompactFlash expansion slots, IrDA, USB, earphones, QWERTY keypad.

Mobile phones

- 2005. Motorola E680i is a Linux-based MP3 player and cameraphone that support stereo bluetooth audio connectivity, an improved interface with full HTML browser, and user-upgradable storage (GB range).

- 2005. Samsung Qtopia is a Linux-based phone, that runs Qtopia Phone Edition, an off-the-shelf Linux application environment for mobile phones from Trolltech. The Samsung phone also supports UWB (ultra wideband) networking.

- **VoIP phones/devices**
  - 2004. Amstrad's E3, based on embedded Linux, is a video conference, email, WEB and picture sending terminal targeted at the private market in the UK.

- **Robots**
  - 2004. ActivMedia's Patrolbot uses Embedded Linux and a high-end Versalogic single board computer (SBC) based on a Embedded 1.6GHz Pentium M. Patrolbot is an intelligent mobile enterprise robot ready for 24X7 autonomous applications. PatrolBot includes laser, sonar, bumpers, dock/charge station and software.

  - 2005. Sony Ericsson. The ROB-1 is a yo-yo-sized robot that can be controlled from certain mobile phones. Compatible phones can display and capture images from its VGA camera.

- **Audio/video devices**
  - 2004. i3 Micro's Mood Box is a set-top box that delivers IP-based TV, VOD, PPV, music-on-demand, Internet browsing, email, and other interactive services to users' TVs. The device is part of i3 Micro's Mood solution, which includes set-top boxes, streaming servers (for IP TV and video-on-demand), and system management servers.

  - 2005. SmartVue's S2 surveillance cam/server runs Debian GNU/Linux and supports DVD-resolution video capture from secure, wireless IP-based cameras. The S2 system uses WiFi and H.264 standards, along with high quality digital optics.

  - 2005. The Siemens Gigaset M740 AV is a Linux-based digital set-top box available in Europe, that had gained some popularity amongst Linux hackers. The Siemens Gigaset M740 AV can decode satellite and terrestrial digital video broadcasts.

- **Thin client devices**

- 2004. Fujitsu-Siemens Futro S. Fujitsu Siemens has launched a line of low power consumption thin clients that requires only 25 W, or 80 percent less power than PCs. The Futro S clients run embedded Linux on Transmeta processors, and are no larger than medium-sized books (24,6 x 4,8 x 17,7 cm).

- **Tablets/webpads**
  - 2004. Kontron V Panels. A high performance Human Machine Interface Panel PC based on a Pentium M processor. The V Panel is rugged and can be configured according to the customer's needs in terms of size, memory and interfaces.

- **Switches, gateways, servers, access points**
  - 2004. 3Com Office Connect VPN Firewall and SecureServer. 3Com's OfficeConnect SecureRouter (3CR860-95) and OfficeConnect VPN Firewall (3CR870-95) are Linux-based VPN routers for the SOHO (small office, home office) market. 3Com offers a free but unsupported Linux-MIPS kernel source tree and cross-compiling tool chain for the devices.

  - 2004. D-Link DSL-G604T. A MIPS-based ADSL router with four 10/100Mbps Ethernet switch ports and an 802.11g high-speed wireless LAN. The DSL-G604T has been marketed primarily in Australia, Great Britain, and Russia. D-Link has released the source code, under the GNU GPL license.

  - Ericsson should be mentioned as a major telecoms vendor who is using Linux in one of it major platforms: Ericsson Telecom Server Platform (TSP) [5]. “TSP is Ericsson's choice of carrier-class server technology for all new multimedia applications and control functionality” [6]. As illustration 1 below shows, Linux is one of the chosen operating systems for TSP. Also, Ericsson seems to be very active in the Carrier Grade Linux (see chapter 14 “Linux for telecoms applications“ page 35) standardization work and is especially visible on the international scene through Dr. Ibrahim Haddad, a Researcher at the Ericsson Corporate Research division’s Open System Lab, located in Montreal, Canada.

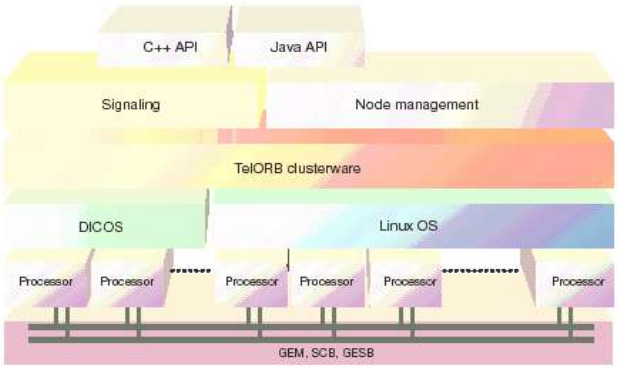

Illustration 1: Software architecture of the Telecom Server platform. The TelORB clusterware also controls the Linux processors in the cluster [5].

- Ericsson Enterprise's Telephony System MX-ONE is a server based IP switch based on Linux and is optimized for real-time applications like telephony. It includes two parts: Telephony Server 1.0 och Media Gateway [7].

- **Vehicle industry**
  - 2004. Daimler-Chrysler StarScan. StarScan is a portable diagnostics tool used by repair shops for Dodge, Chrysler, and Jeep vehicles incorporating CAN. It uses real-time embedded Linux.

  - 2004 . Kontron CV Server Car PC. A rugged computer for use in vehicles with a variety of available CPU modules, from a 300MHz Pentium III to a fanless 1.4GHz Pentium M. Supports Linux as an operating system.

  - 2004. TomTom Go. The TomTom Go is a Linux-based in-car GPS navigation system available with or without preloaded maps. TomTom provides a full set of source code under the GPL license: the compiler tool chain used to build all the software, the Linux kernel for ARM, with modifications by TomTom, BlueZ libraries and utilities (official Linux Bluetooth protocol stack), TomTom software under GPL license, other third party software under GPL license.

  - 2003. Volvo Mobility Systems ITS4mobility. ITS4mobility is an intelligent transport system for public transports that covers the functional areas of traffic control, travel information and vehicle management. It allows integration with subsystems such as traffic signal priority and passenger counting. It enables tracking of bus locations, two-way messaging, bus arrival forecasts, and other valuable functions. ITS4mobility runs a 58MHz Motorola MPC 823e, with MontaVista Professional Edition, including a 2.4.17 Linux kernel.

- **Test and measurement**
- 2004. Fluke Networks EtherScope. The EtherScope can quickly map, scan, and test 10/100/Gigabit Ethernet LANs for configuration, wiring, and other problems. It's running Montavista Professional Edition, kernel 2.4.18, with Qt embedded. The EtherScope has 256 MB of RAM, 32 MB ROM, PCMCIA and CompactFlash slots, USB port, and audio jacks.

- **Payment, Point of Sales**
  - 2004. Squirrelsystems SquirrelOne. "The SquirrelOne is a restaurant POS (point-of-sales/service) system that uses Linux-based POS clients along with a Windows-based server.

  - 2004. Trintech Smart 5000 PIN Pad with integrated hybrid card reader, rolled out by Shell in the UK and Ireland. Embedded LINUX – kernel V2.4, ARM 7, Flash 4MB (optional 8MB), SDRAM 16MB, SRAM 512KB, standard serial (RS 232), USB Support, display 132x64 pixel (8 line), keyboard 15 keys + 6 programmable keys, dimensions 205x110x120 mm.

7.2 Hard real-time Linux applications

Applications mentioned on FSMLabs' (see chapter 9 “Distributions“ page 22) WEB site include: development and test of a jet engine at Pratt & Whitney and Hamilton Sundstrand, Unmanned Aerial Vehicles at The University of Alabama, and control of advanced coal power plants at Beijing Northern Power Institute.

8 Today's market and outlook

In this chapter, we try to draw conclusions on the current market for Linux as an embedded/real OS today and its outlook for tomorrow based on two surveys made on that topic: Linuxdevices.com's 2005 Embedded Linux Market Survey and Embedded.com's Embedded systems survey, also from 2005 ([4] and [3] respectively).

Additional information can also be found on www.designnews.com in an article entitled "Embedded's New Star" about Linux from June 2005 [8].

8.1 Today's market

According to Embedded.com's survey, 44% of embedded system's developers use a commercial OS today, 19% no OS at all, 17% an internally developed OS and 20% an Open-Source OS.

Unfortunately, Embedded.com doesn't mention the amount of respondents to the survey, and it is therefore difficult to judge how representative these figures are. Furthermore, Open-Source doesn't have to be opposed to Commercial, Montavista Linux being one example of an operating system that is both Open-Source and commercial.

However, Embedded.com has excluded all Linux based commercial operating systems from the Open-Source category. According to Embedded.com, commercial variants of Linux represent 19.3% of commercial operating systems in embedded systems. If we add 19.3%x44% of commercial Linux to the 20% of Open Source operating systems which are also claimed to be Linux based in the article, we obtain a total of **28.5% for Linux based operating systems**. From the same survey **VxWorks represents around 19%** of the market and **Microsoft's operating systems around 22%**.

In comparison, Linuxdevices.com's recipients, who are self-selected among the site's readers, and therefore probably not representative of the embedded market, claim to have used during the latest 2 years (only the first four choices in volume are mentioned here):

- **Linux: 42.7 %** of the respondents
- **Microsoft's operating systems: 15.6%** of the respondents
- **VxWorks: 8.3%** of the respondents
- **Proprietary or in-house OS: 9.3%** of the respondents

8.2 Outlook

According to [3], embedded system suppliers will use the following operating systems for their next project (percentage of increase or decrease compared with current OS is indicated):

- Commercial: -0.5%
- Open-Source: +8.5%
- Internal: -2.5%
- None: -4%

From those figures, Embedded.com derives that the use of open source operating systems will increase in embedded development, whereas other alternatives will decrease (commercial operating systems, internal operating systems and systems without an operating system).

Additionally, **41% of Embedded.com's recipients are using Linux today or are likely to use it soon**.

If we add the recipients who say that they are likely to use it, but not soon, we reach the 74% of all recipients!

Embedded.com also tells us that at least 18% of the Embedded.com's recipients who want to rely on commercial operating systems will consider one that is based on Linux.

Linuxdevices.com's recipients say that they will use during the next two years:

- **Linux: 54.5 %** of the respondents
- **Microsoft's operating systems: 12.9%** of the respondents
- **VxWorks: 6%** of the respondents

- **Proprietary or in-house OS: 6.7%** of the respondents

8.3 CPUs

For information, Linuxdevices.com's recipients have used the during the last two years and plan to use during the next two years, the following CPUs:

- ARM: 31% and 32%
- x86: 29% and 23%
- PowerPC: 14% and 16%
- MIPS: 7% and 8%
- Coldfire: 6% and 5%
- DSP: 5% and 6%
- Other: 2.5% and 1.5%
- SH: 2% and 2%

9 Distributions

This chapter deals with target distributions, i.e. distributions that are used to build the target software. This is pretty much independent from the choice of a host distribution, i.e. distributions that are used on the host system. We won't deal with host distributions in this document.

9.1 Distributions' popularity

According to [4], more than 60% of Linuxdevices.com's recipients use free and open source tools and Linux distributions for their embedded development. The rest, less than 40%, use commercial distributions or tools.

Among the first group, Debian is the most popular distribution, followed by home-grown (assembled from freely downloadable, non-commercial sources) and OpenEmbedded (actually more a development environment that especially supports handhelds-oriented distributions, like Familiar or OpenZaurus)[2].

Among the embedded commercial alternatives for Linuxdevices' recipients (that excludes Redhat, which is apparently considered as a non-embedded commercial alternative), Montavista is the most popular, followed by Koan (Italian vendor providing the K-Linux embedded distribution) and Sysgo (German vendor providing ElinOS embedded distribution). FSMLabs's (American vendor profiled in hard real-time systems and providing RTLinux-based distributions) appears to become almost as popular as Montavista for the near future among Linuxdevices.com's recipients. Wind River, the vendor behind VxWorks, should also be noticed as a Linux distribution vendor with increasing popularity.

2 Linuxdevices.com mentions an unfortunate mailing list post encouraging users to vote [4], that could partly explain OpenEmbedded's surprising popularity in the survey.

In comparison, [3] shows the following list of commercial Linux vendors in decreasing order of popularity for Embedded.com's recipients:

- Redhat, Montavista, Lynuxworks (LynxOS and Blue Cat Linux), TimeSys (Linux/RT), FSMLabs (RT-Linux), Wind River (NE Linux = Network Equipment Platform, Linux edition)

9.2 Advantages and drawbacks of different approaches

Chapter 10 “Process for the development of a Linux embedded system“ on page 28 will give a feeling of what is required to set up a Linux embedded development environment. One of the steps is the choice of a target distribution. As we have seen earlier in this chapter, growing one's own distribution is a surprisingly popular alternative.

The main advantage of such an approach is control and flexibility. The main drawback is the lack of committed support, with the impacts it can have on time to market.

We can guess that one of the explanations for the popularity of own grown distributions is that embedded Linux is still a young market, and so are the distribution vendors in this area. The growing experience of those vendors will probably make a difference in their understanding of their customers' issues, and going with a distribution vendor will probably mean limiting the time spent solving unexpected problems, which in turn will reduce time to market.

Still, the freedom paradigm behind open software will still remain, and there will always be a share of embedded developers who will prefer to choose, integrate and test freely downloadable components by themselves.

We will now give more information about the following vendors and distributions:

- debian, home-grown, OpenEmbedded, Redhat, Montavista, Lynuxworks (LynxOS and Blue Cat Linux), TimeSys (Linux/RT), FSMLabs (RT-Linux), Wind River (NE Linux = Network Equipment Platform, Linux edition), Koan (Klinux), Sysgo (ElinOS), RTAI and Denx.

9.3 debian

Debian is the most famous non-commercial general purpose Linux distribution, featuring a large number of software packages (15490). The debian community is putting some efforts into making debian the first choice distribution for embedded systems [9]. However, while emdebian describes a cross-compiling strategy with debian as a host, it doesn't provide as of today an actual distribution that can easily be installed on the target. Under those circumstances, the main difference between choosing debian and growing a home distribution for an embedded development would be that in the debian case, the components chosen for the target are taken from the debian source packages instead of being taken from their original supplier (GNU, etc.).

9.4 Home-grown

The idea of a home-grown distribution is to fetch the different components needed on the target in source code, including the Linux kernel, and to compile them on a host in a cross-compiling environment. We will give more details about that procedure in chapter 10 “Process for the development of a Linux embedded system“, page 28.

9.5 OpenEmbedded

“Basically OpenEmbedded is a build system that can generate (cross-compile) Software packages for embedded targets. This may include Bootloader, Linux and Applications.” [10]

“It has been designed to be able to handle different hardware architectures, support multiple releases for those architectures, and utilize tools for speeding up the process of recreating the base after changes have been made.” [10]

OpenEmbedded seems especially targeted at software development for PDAs.

9.6 Redhat

Redhat is the most famous commercial general purposes Linux distribution, and is mostly profiled towards the enterprise server segment. Redhat has also an offering for the embedded segment, that includes the following components:

- GNUPro Multi-platform Software Development Tools: this is a tool chain that is common to both non-embedded and embedded development.
- Custom Engineering for Embedded Linux: this is mainly a service offering. It also includes a product: a bootloader for embedded systems called RedBoot.

9.7 Montavista

Montavista is today marketing four different products.

- MontaVista Linux for Embedded Devices (Professional Edition)
- MontaVista Linux for Communications Infrastructure (Carrier Grade Edition)
- MontaVista Linux for Consumer Electronics (Consumer Electronics Edition)
- MontaVista Linux for Mobile Devices (Mobilinux)

These are packages based on open source components, adapted to the needs of different segments. Montavista use the performance benefits of the standard Linux kernels (2.4 and 2.6) and have themselves made contributions to these kernels. For example, “the MontaVista Linux Preemptible Kernel received the Editor’s Choice Award from Linux Journal and was adopted by the Linux kernel project as a mainstream feature [11].” This feature is part of the 2.6 mainstream kernel.

“Improvements in task-level response from Preemptible Linux Kernel are dramatic: worst-case preemption latency on typical embedded CPUs like PowerPC and IA-32 processors drop from hundreds of milliseconds to under one millisecond; typical pre-emption latency (99.9% or more) falls to a few hundred microseconds; and average preemption latency falls to tens of microseconds.” [11]

Although “MontaVista Software has made pioneering contributions to the responsiveness and determinism of embedded Linux” [11] and “MontaVista has sponsored projects and implemented technologies to enhance Linux real-time capabilities” [11], they don't make a clear commitment to provide hard real-time performance on their WEB site.

Montavista's different distributions include: Board Support Packages for “nearly 100 popular COTS, Evaluation, and Reference boards” [11] and “seven target CPU families with more than 25 CPU variants” [11], an IDE called Montavista DevRocket, and numerous libraries and utilities that can be used on target.

Montavista has lately focused on the telecoms vendors and mobile phone vendors with their latest versions of CGE (Carrier Grade Edition) and Mobilinux. They have implemented SAF's APIs OpenAIS and OpenHPI (see chapter 14 “Linux for telecoms applications“ page 35) as part of CGE 4.0, which also conforms to Open Source Development Labs’ Carrier Grade Linux Requirements 2.0.

9.8 Lynuxworks

9.8.1 LynxOS

LynxOS has been a proprietary hard-real time operating system for 16 years, that has always provided a binary compatibility with UNIX and POSIX. The latest version, 4.0, provides a Linux ABI compatibility. More specifically, this ABI compatibility has been tested against Linux kernel v2.4.x/GLIBC v2.2, meaning that applications which are running in such an environment can run on LynxOS v4.0 without recompiling.

There is a version of LynxOS for safety-critical systems called LynxOS-178.

BSPs for PowerPC, IA-32 and Xscale boards, as well as a development tool chain are provided with the LynxOS distributions.

9.8.2 BlueCat Linux

BlueCat Linux is LynuxWorks' own Linux distribution. The latest version is 5.0, which is based on the 2.6 version of the mainstream Linux kernel.

According to LynuxWorks, the source code for the GPL licensed components of BlueCat Linux is available upon request from LynuxWorks Support.

BSPs for ARM, PowerPC, x86, MIPS and SH3 based boards, and optional IDEs and debug tools are provided with BlueCat Linux.

9.9 TimeSys

Linux/RT, a product that was released in version 2.0 at the beginning of 2001, is not mentioned on TimeSys' WEB page any longer.

TimeSys is now focusing on the customization of Linux for various embedded target platforms. TimeSys has the following software modules available:

- TimeStorm® Linux Development Kit: BSPs including root file systems - for PowerPC, ARM, MIPS, XScale, and IA-32 - and a graphical IDE called TimeStorm® Linux Development Suite.

- LinuxDepot, which is a Linux Component Repository and a Developer Exchange. The Developer Exchange is a kind of common development forum for TimeSys experts and embedded developers using TimeSys' products.
- Linux Verification Suite, which is an Eclipse-based framework for locally testing and validating a customized Linux. [12]
- LinuxEngine, which is a tool to build a complete target distribution and manage build versions.

9.10 FSMLabs

FSMLabs has one main product called RTLinux, that includes a hard real-time sub-kernel (RTCore) and runs the Linux kernel as its lowest priority task.

The sub-kernel prevents Linux from disabling interrupts. In order to do so, the Linux kernel is patched to replace the calls to the interrupt disabling and enabling instructions by alternate functions, and the sub-kernel emulates the hardware interrupts. This way, the hard real-time tasks running under the sub-kernel can be guaranteed a given latency, while non-real time tasks can run under Linux and use its full set of functionality. Rate Monotonic Scheduling and Earliest Deadline First are two of the real-time scheduling policies available in RTCore [14] (see paragraph 5.2 “Real-time systems“ page 11).

FSMLabs is giving worst case “thread jitter(s)” (this is what they call the latency for a real-time task, including interrupt latency, scheduling and context switch) in the range 10 microseconds to 100 microseconds depending on the hardware architecture.

RTLinux is protected by a software patent, and FSMLabs is licensing it both commercially (RTLinuxPro) and under the GPL (RTLinuxFree). They indicate explicitly that all software that includes RTLinuxFree must be distributed under the GPL. [15]

RTCore provides the POSIX API 1003.13 PSE51, designed for a “minimal real-time system profile”. [16]

Linux kernel versions 2.4 and 2.6 are available with RTLinux and a variant with the same RTCore and a BSD kernel instead of Linux, called RTCore BSD, is also available.

9.11 Wind River

Wind River offers 6 runtime platforms targeted at different types of applications: General Purpose Platform, Platform for Automotive Devices, Platform for Consumer Devices, Platform for Industrial Devices, Platform for Network Equipment, and Platform Safety Critical.

Among those, both the General Purpose Platform and the Platform for Network Equipment have a Linux Edition and a VxWorks edition. The others only have a VxWorks Edition.

Of course, it is significant that the vendor of one of the largest embedded operating systems in terms of market share, namely VxWorks, has taken a clear step towards Linux and has now added it to its portfolio.

According to Wind River, if the device requires capabilities provided by Linux and is not constrained by memory capacity or real-time requirements, the General Purpose Platform, Linux Edition meets the need. [17] It is based on the Linux kernel version 2.6.

The Network Equipment, Linux Edition includes a carrier Grade Linux Kernel based on 2.6, as defined per Open Source Development Labs' specifications. [18]

For the Linux runtime platforms, Wind River provides a cross-build system, and IDE called Workbench Development Suite based on Eclipse 3.0 and BSPs for target boards based on x86, ARM, PowerPC and MIPS processors.

9.12 Koan

Koan Software appears to offer a distribution called K-Linux, made from GPL licensed components, and has ready-to-use BSPs for various boards based on ARM, PowerPC, Coldfire, MIPS and Hitachi processor. It includes a GUI called K-GUI an IDE called Ktx the Toolkit, and the royalty free real-time sub-kernel RTAI. [19]

A version called Klinux Live 2.0.0 is available free of charge for evaluation purposes at www.klinux.org.

9.13 Sysgo

Sysgo offers the Linux based ElinOS line of products, and has the reputation of having very affordable solutions, compared to the other established commercial Linux embedded vendors.

The FreeToolbox product is downloadable for free and includes BSPs for x86, PowerPC and ARM based target platforms, and a cross-development platform.

ELinOS commercial DevelopmentKit includes the Embedded Linux Konfigurator, which is a “Graphical user interface for the creation of directly bootable (ROM-) images with integrated kernel- and system-software configuration”, as well as RTAI. [20]

They also offer an Eclipse based IDE called CODEO and and target system analysis tool called COGNITO.

Linux kernel versions 2.4 and 2.6 are available in ELinOS.

Besides, Sysgo is also a reseller of LynuxWorks' LynxOS.

9.14 RTAI

The basic principles of RTAI are similar to those of RTLinux (see paragraph 9.10 “FSMLabs“ page 26). However, the development community behind RTAI is very active and there are currently several development tracks for RTAI. Please refer to [21] for more information about the technology behind RTAI.

RTAI is contributed in accordance with the Free Software guidelines. The project has been founded by the Department of Aerospace Engineering of Politecnico di Milano (DIAPM). Over the years, it has become a community effort involving international developers, coordinated by DIAPM's Prof. Paolo Mantegazza. [21]

9.15 DENX

DENX is the company behind Das U-Boot, a universal boot loader that is very popular for Linux based embedded systems.

They have also ported RTAI to the PowerPC.

They provide these software packages, together with their Embedded Linux Development Kit (ELDK) , for free download or for purchase on CD-ROM. The ELDK includes BSPs for PowerPC, ARM and MIPS based boards, RTAI, Das U-Boot, and some cross-development tools.

10 Process for the development of a Linux embedded system

For a detailed presentation of the process to start the development of a Linux embedded system, please refer to [22].

We can summarize the necessary steps necessary to have a development environment up and running as follows.

10.1 Specify the requirements

Specify the requirements, in terms of data input, output and processing. Translate them into data flows, storage and processing capacity. For a good example of requirement analysis and design of an embedded system, including real-time requirements, please refer to [23] (only in Swedish unfortunately).

10.2 Choose the target hardware

Based on the previous step and other requirements, **choose the target hardware**. Of course, there are many other important factors that will influence this choice: economical, mechanical, etc. We won't deal with these aspects here. Also, availability of the software (BSPs, tools, etc.) and support from the suppliers is an important aspect. Choice of the hardware and type of source for the software are really connected choices. Choice of target hardware and software is more a matrix choice than two independent choices, but we won't take that discussion any further in this report.

Important parameters for the target hardware are: processor speed and processing capacity (in Mips or Mflops), main memory, permanent storage, I/O ports, support for peripherals. In order to run Linux, the system will probably have **at least 2 MB of ROM and 4 MB of RAM**. Running Linux on a smaller system will require particular efforts [22].

10.3 Choose the source for the target software

Choose the source for the target software: rely on a supplier, or “roll your own”. Chapter 9 “Distributions“ page 22 is probably a good place to start when choosing a software vendor or source of target software. In this chapter, we will mostly assume that we are in a roll-your-own scenario.

10.4 Choose the host system

Choose the host system, if not already done. This doesn't even have to be Linux, although having Linux as an operating system for the host system will often ease the development process.

10.5 Connect the host to the target

Although a standalone development directly on the target is possible if the target is powerful enough (including the ability to be connected to a keyboard, mouse and monitor), the common case is to perform the actual development on a separate computer, typically a PC running Linux with a cross development environment. The host is connected to the target via a serial port, or Ethernet, or both. Alternatively, the software is loaded from the host onto a flash device, which is physically inserted in the target when testing the target software. Both the host and the target could also connect to a common NFS file system during development, thereby removing the need for transferring binaries from the host to the target. There could also be a specific debugging connection (JTAG, BDM, etc.) between the host and the target.

10.6 Install and configure the cross-development environment

The idea of this step is to configure a development tool chain that is able to build binaries for the target platform. Since the host platform is often not based on the same architecture and processor as the host, binaries built with the unmodified native development tool chain would most likely not run satisfyingly on the target, if at all. According to [22], if we limit ourselves to the traditional **GNU tool chain**, this involves: kernel headers setup, binary utilities setup, bootstrap compiler setup, C library setup, full compiler setup. Often, we will choose a C library that is smaller than the standard GNU C library, in order to reduce the size of binaries on the target system. **uClibc** is a popular alternative, that maintains compatibility with C99 and SUSv3, but is less ambitious than glibc in complying to other standards. Often, a terminal emulator will be part of the development environment, in order to communicate with the target system through a serial port. **Minicom** is a popular terminal emulator under Linux.

10.7 Creating a target root file system

The basic structure will be quite standard. However, we will here as well limit the space taken by tools to the necessary minimum. A popular source for the main system applications is a package called **BusyBox**.

10.8 Configure the permanent storage

Under this heading, we gather several steps of a process that can be quite complex:

- Decide the **booting scheme** for the target system: booting from an embedded flash permanent storage, booting hard disk, booting from a remote DHCP/TFTP server via a PXE enabled network interface, etc.
- **Partition** permanent storage. Decide and implement the **file system** (read only or read/write, JFFS2, CRAMFS, etc.).
- Choose and implement a **boot loader**. **U-Boot** is a popular choice for systems without a hard disk. How this can be implemented depends a lot on the target hardware. It can involve connecting to the target platform via a terminal emulator an transferring the boot loader to the flash memory.

11 Licensing

There are complex licensing issues around free software and embedded systems providers should get help from a lawyer if they have some doubts about what those issues imply for their own products. We will however try to give simple principles about those issues in this chapter.

11.1 GPL and LGPL

The GPL (General Public License) is used by the free software community as a copyright license mainly for applications, and implies a freedom to copy and change the software, but also that the source code needs to be made available even if the software is delivered in binary form. Additionally, any piece of software that includes GPL software must itself be licensed under the GPL. This also applies to software which is derived from GPL software, the word “derived” being the source of controversies. The Linux kernel is licensed under GPL v2.

LGPL (Lesser General Public License) is used by the free software community as a copyright license mainly for software libraries. It gives the same freedom of copy and modification as GPL, but doesn't force software that includes LPGL software to itself be licensed under the LGPL, as long as the LGPL software is not modified. As an example, the GNU C library is licensed under LGPL

Other types of licenses can be found among open software like the BSD license, the Apache license, etc. We will only focus on GPL and LGPL here.

For embedded Linux systems, the consequences of GPL and LGPL are the following.

- **Applications running under Linux are in general not forced to be licensed under the GPL**, as long as they don't include any GPL code. Most applications can for example very easily be distinguished from the Linux kernel, and cannot be considered as including it or part of it. Nor can they be considered as derived from

the Linux kernel as long as they for example use standard POSIX system calls and don't have to be linked with kernel headers.

- **Applications that include LGPL libraries are in general not forced to be licensed under the LGPL or GPL**, as long as they don't include any GPL code, and the libraries are not modified. For example, applications including the GNU C library can be delivered as commercial products without application source code, since GNU C library is licensed under the LGPL, provided the applications don't include other code or library that have other types of licenses.
- **Linux devices drivers can be considered as derived from the Linux kernel, and should therefore be licensed under the GPL.** Nevertheless, that rule is often not applied by a number of vendors who don't want to provide the source code for their drivers, probably in order to avoid revealing secrets about their hardware. However, a company delivering Linux device drivers, and more generally Linux kernel loadable modules or software that is linked to the Linux kernel should weigh the possible consequences of not licensing that software under GPL. There is at least a theoretical risk that they would be sued by one of the copyright owners of the Linux kernel.

11.2 RTLinux patent and RTAI's license

Victor J. Yodaiken owns a US Patent for the technology used in RTLinux (US5995745: Adding real-time support to general purpose operating systems). Although this technology can be used without royalty for software delivered under the GPL, according to a license called “Open RTLinux Patent License” from FSMLabs, the patent has created controversy, especially since it is so general that it could encompass RTAI.

RTAI itself is licensed partly under GPL and partly under LGPL. Therefore, what was said above about applications under Linux also applies to applications under RTAI/Linux.

11.3 SCO's lawsuits

In 2003, SCO, who is the current owner of the UNIX System V Release 4 intellectual property assets, has filed several lawsuits against, among others, IBM and Novell, for copyright infringement and in the scope of other conflicts. These lawsuits are still pending.

The base for some of the claims in the lawsuits is an underlying claim from SCO that some of UNIX System V's code that SCO owns copyrights for is included in the Linux kernel without their permission. However, no court has so far confirmed that this really was the case.

SCO has also started SCOsource, a business division in charge of licensing SCO's UNIX intellectual property. SCOsource is requiring the payment of a license fee for any server running Linux, currently 699 USD for a server with one CPU.

Of course, these issues are a possible risk for any company using Linux in its products. However, if it was confirmed that SCO has copyrights on some code that is part of the Linux kernel, one can guess that the Linux kernel community could replace this code by some new and fresh code providing the same functionality (the claims don't seem to concern large volumes of code). The copyrights would then

remain an issue for products relying on former versions of the kernel, and Linux embedded developers should certainly keep an eye on the evolution of these issues in the future.

12 Total cost of development

12.1 Embedded Market forecasters' report from 2003

Embedded Market forecasters has published a free report in July 2003 entitled “Total cost of development” [24], that compares Linux to Microsoft's operating systems for embedded development. That report concluded that the total cost of using Linux was between three and four times more expensive than using Windows XP embedded or Windows CE.NET, only taking into account the manpower necessary for the development. It also suggested that development tools for Linux were more expensive, and license fees for modules that are not included in royalty free GNU/Linux are in total more expensive than licenses for Windows CE.NET (codecs for various media formats for instance).

This report has provoked many reactions, admittedly many of them from Linux followers. As a matter of fact, the author of the report himself has publicly admitted that the study had been partially financed by Microsoft [25]. Given this fact, it is difficult to give the report much credit. Besides, the reports compares embedded project development times without taking possible large differences between the devices being developed into account. Other valid criticisms have been made on the contents of the report.

However, the cost estimation framework developed for the report makes a lot of sense. In fact, we can hardly imagine how a company developing embedded systems could make a choice of an OS without doing a total cost estimation for the product's life time. As the report suggests, this should include development tools, support, maintenance, development time and royalties for the OS and any non-free module.

For information the report quotes a royalty cost between 2.6 USD and 9 USD for Windows CE.NET. It doesn't give any such information for Windows XP embedded.

12.2 VDC's report from 2005

VDC has published a report called “The Mobile Software Stack for Voice, Data, and Converged Handheld Devices” in April 2005. “Based on a survey of over 1000 global mobile developers”, [26] of which 118 are using Linux. According to this report, Linux incurs longer (two to three weeks) and less predictable development times than Microsoft, Symbian and PalmOS.

Linuxdevices.com proposes some possible explanations to this bad performance [26] (almost half of the developers were using freely available, unsupported distributions; neither hardware drivers nor test and validation tools were available as much as now when the developments started), but such results cannot be ignored, and they raise once more the necessity of making objective estimations of the total cost associated with the OS for the whole product life cycle.

12.3 An OS cost comparison strategy

Most embedded suppliers have by now realized that Linux is not for free from a product developer's point of view. However, we can't either assume that the cost for tools associated with Linux will be prohibitive and that the development will necessarily be longer.

We know for sure that Linux is not an easy choice, because it leads to many different alternatives, from a completely free set of tools, to a turnkey environment from one single supplier.

Parameters to take into account when making a comparison include: cost of the development seat, training (depending on the experience already available), support and maintenance from the OS/tool supplier(s), cost of developer's time, cost for royalties both for the OS and other potentially necessary modules.

With the difficulty of choice also comes the freedom of choice: it is actually possible to start the development of a prototype under Linux with entirely free tools, including an IDE like Eclipse, without even needing to contact a vendor's sales person. This approach can be interesting if time-to-market constraints and available staff allow it and can help to foresee critical issues and forecast costs.

13 Which Linux kernel to choose?

13.1 Changes in kernel v2.6 compared to v2.4

There has been many changes in the kernel between versions 2.4 and 2.6 [27]. It is difficult to list the most important changes since this depends a lot on the application, but if we still try, the list of the three most important changes could be the following:

- **Kernel preemption**. In v2.4, the kernel used a preemption called “user preemption”. This means that the Linux scheduler was able to interrupt a user task in order to run a higher priority task, or if the time slice of the currently running task had elapsed. This is how most multitasking systems work. However, the kernel itself could not be preempted, which had an impact on latency. In order to improve latency, the v2.6 included “kernel preemption”. This means that code running in kernel space can also be preempted, at least in areas where it is safe, i.e. as long as it doesn't hold a lock. [28]
- **O(1) scheduler**. In v2.4, the task scheduler had to go through the list of all tasks in ready state in order to decide which to run next. This algorithm had a running time that was therefore proportional to the amount of ready tasks. This is called O (n) complexity. This means that the scheduler overhead was increasing without limit with the amount of tasks. Instead, v2.6 includes a scheduler which uses a priority array and a list of ready tasks sorted by priority, that allows the scheduler to find the next task to run in a constant time, independently from the amount of ready tasks. This is called a O(1) complexity (constant running time, independent from the amount of tasks). With the O(1) scheduler, the amount of tasks can be arbitrarily high, the scheduler overhead will still be the same. [28]
- **Threading improvements**. NPTL (Native POSIX Thread Library) is a new implementation of POSIX threads in v2.6 that provides a much better performance

of threads, especially creation and destruction time, than the version present in v2.4.

The kernel v2.6 has shown to provide performance improvement over the kernel v2.4 in some cases, for example for servers running a database application. [29]

13.2 Performance of the kernel v2.6 in embedded systems

Wolfgang Denk at DENX software engineering in Germany has made a comparison of kernels v2.4 and v2.6 in April 2005 on two PowerPC-based boards, with processors commonly used in embedded systems. [30]

His management summary is the following:

“Using the 2.6 kernel on embedded systems implicates the following disadvantages:

- Slow to build: 2.6 takes 30...40% longer to compile
- Big memory footprint in flash: the 2.6 compressed kernel image is 30...40% bigger
- Big memory footprint in RAM: the 2.6 kernel needs 30...40% more RAM; the available RAM size for applications is 700kB smaller
- Slow to boot: 2.6 takes 5...15% longer to boot into multi-user mode
- Slow to run: context switches up to 96% slower, local communication latencies up to 80% slower, file system latencies up to 76% slower, local communication bandwidth less than 50% in some cases.

There may be a few cases where the use of Linux kernel version 2.6 makes sense even for embedded systems (typically on "bigger" systems with more powerful processors), but **if memory footprint or system performance are important you probably want to stick with a 2.4 kernel for now.**”

I quote hereafter Wolfgang's answer to a question about why he believes that the kernel v2.6 is showing worse performance than v2.4 on embedded systems, with his permission:

“The fundamental problem as I see it is as follows:

- Embedded processors have small caches (in the order or 4...16 kB) and almost never you can find a L2 cache. This means that they are very susceptible to code size changes.
- In 2.4 the clock tick was 10 milliseconds (100 Hz), while in 2.6 it is 1 millisecond (1 kHz). This means that all actions triggered by the clock tick now run ten times as often as before.
- The new high-performance scheduler has a code size which is more than 3 times that of the 2.4 scheduler.

If we consider just these facts alone, we see that the scheduler code that needs to be run in 2.6 is approx. 10 x 3 = 30 the amount we had in 2.4. Say the overhead was 0.5 % in 2.4 - so now we have: WOW!

And many other of the "optimizations" that went into 2.6 are for high-end systems, too: servers with thousands of processes, 64 bit systems with many gigabytes of system memory and CPU clock frequencies in the GHz ranges. Guess how this matches for example a MPC860 processor running at 50 MHz with a total of for example 16 MB RAM...

On a small embedded system with 5 or maybe 10 or maybe even 20 processes it does not matter at all if the scheduling algorithm has a O(1) or O(N) or even O(N*N) characteristics - it just has to be small and fast. 2.6 was designed with a completely different point of view in mind."

For information, we list below the kernel versions that are included in the current distributions of some embedded Linux vendors at this point in time:

Montavista: 2.4 and 2.6.

LynuxWorks BlueCat: 2.6

SysGo ELinOS: 2.6

FSMLabs RTLinux Pro: 2.4 and 2.6

Wind River General Purpose Platform, Linux edition: 2.6

Koan Klinux: 2.4

RTAI: support for both 2.4 and 2.6

DENX: 2.4

14 Linux for telecoms applications

A general presentation of standards based on Linux to meet the needs of the telecoms industry can be found at [31].

14.1 ATCA

ATCA is not specific to Linux, but it is often the physical layer used to build Linux based telecoms systems.

ATCA or AdvancedTCA (Advanced Telecommunication Computing Architecture), is a standard specified by the PICMG (PCI Industrial Computer Manufacturers Group) and covers shelves, boards, mezzanines and management. [32]

The current version of the ATCA standard is called PICMG3.0 and was ratified in December 2002.

It covers the following identified market needs: [32]

- Scalable shelf capacity to 2.5Tb/s
- Scalable system availability to 99.999%
- Multi-protocol support for interfaces up to 40Gb/s
- Robust power infrastructure and large cooling capacity
- High levels of modularity and configurability

- Ease of integration of multiple functions and new features
- The ability to host large pools of DSPs, NPs ,processors, and storage
- Convergence of telecom access, core, optical, and datacenter functions
- Advanced software infrastructure providing APIs and OAM&P
- High security and regulatory conformance
- World-class element cost and operational expense
- A healthy, dynamic, multi-vendor, interoperable eco-system
- Available now, with a design-in life through at least 2010

14.2 Carrier Grade Linux

CGL (Carrier Grade Linux) is a working group which is part of the OSDL (Open Source Development Labs). “OSDL is a non-profit organization founded in 2000 to accelerate the growth and adoption of Linux in the enterprise.” [33]

CGL specifies capabilities in several areas: [*33*]

- Standard compliance: Linux Standard Base, POSIX, SA Forum, PICMG, etc.
- Hardware: CPU Blade Hot Swap, hardware platform management, trusted hardware power management, etc.
- Availability: five nines availability (99.999%), online operations, redundancy, monitoring, etc.
- Serviceability: remote management, diagnostic monitoring, failure analysis, debugging tools, etc.
- Clustering.
- Security.
- Performance: efficient load balancing/SMP support, optimized protocol stacks, large physical memory support, more real-time support with low predictable responsiveness.

Mostly, conformance to CGL is provided by vendors who are active on the telecoms market. Those vendors will contribute to the mainstream Linux kernel to a certain extent, but some features might never be integrated in the main stream kernel, if the kernel community doesn't consider them as beneficial for the vast majority of the kernel's users.

The current list of CGL members is: 10art-ni, Aduva, Alcatel, BakBone, Cisco, Comverse, Ericsson, Fujitsu, Hitachi, HP, IBM, Intel, Lynuxworks, MontaVista Software, NEC, Nokia, Novell, NTT Corporation, NTT Data Intellilink, Red Hat, Sun Microsystems, Timesys, TurboLinux, Wind River.

Version 3.0 of the CGL specification currently pertains.

14.3 SA Forum

“The Service Availability™ Forum is a consortium of industry-leading communications and computing companies working together to develop and publish high availability and management software interface specifications. The SA Forum then promotes and facilitates specification adoption by the industry.” [34]

The current list of members is: Alcatel, Artesyn Technologies, Augmentix Corporation, Clovis Solutions, Continuous Computing, Ericsson, ESO Technologies, Fujitsu Limited, Fujitsu Siemens Computers, GNP, GoAhead Software, Hewlett-Packard, IBM, Intel, Kontron, Lucent Technologies, MontaVista Software, Motorola, MySQL AB, NEC, Nokia, Nortel Networks, NTT, Oracle Corporation, OSA Technologies, Phoenix Technologies, Radisys, Siemens, Solid Information Technology, Sun Microsystems, TietoEnator, UXComm, Veritas Software. [34]

SA Forum has specified two interfaces so far: the **HPI** (Hardware Platform Interface) and the Application Interface (referred to as the **AIS**, for Application Interface Specification).

“[**The HPI**] specifies a generic mechanism to monitor and control highly available systems. The ability to monitor and control these systems is provided through a consistent, platform independent set of programmatic interfaces. The HPI specification provides data structures and functional definitions that can be used to interact with manageable subsets of a platform or system. The HPI allows applications and middleware (HPI User) to access and manage hardware components via a standardized interface. Its primary goal is to allow for portability of HPI User code across a variety of hardware platforms.” [35]

The **AIS** “provides a view of one logical cluster that consists of a number of cluster nodes.” [36] It “determines the states of a component and monitors the health of components.” [36]

The SA Forum has also produced SNMP experimental MIBs for inclusion in a System Management Specification to be published.

Versions B.01.01 of HPI and AIS are active right now.

14.4 Implementations

Most hardware vendors who are active in the telecommunication market provide hardware that is compliant to ATCA.

When it comes to Linux Carrier Grade and SA Forum, the following companies, among others, have made some announcements in the area: Montavista, IBM, Veritas, Wind River, Oracle, Motorola, HP, Fujitsu Siemens Computers, Intel.

15 References

1. Comp.realtime: Frequently Asked Questions (FAQs) (version 3.6), http://www.faqs.org/faqs/realtime-computing/faq/
2. FSMLabs, Inc., Real-time Linux Applications and Design, http://www.fsmlabs.com/images/stories/pdf/archive/RTL_app_design.ps
3. Embedded.com, Embedded systems survey: Operating systems up for grabs, May 2005, http://www.embedded.com//showArticle.jhtml?articleID=163700590
4. Linuxdevices.com, Great Gadget Smack-Down, http://linuxdevices.com/articles/AT2631993452.html
5. Telefonaktiebolaget LM Ericsson, Ericsson Telecom Server Platform 4, 2002, http://www.ericsson.com/about/publications/review/2002_03/files/2002032.pdf
6. Telefonaktiebolaget LM Ericsson, Ericsson's family of carrier-class technologies, 2001, http://www.ericsson.com/about/publications/review/2001_04/files/2001045.pdf
7. Ericsson Enterprise, MX-ONE Telephony System 1.0, http://www.ericsson.com/enterprise/library/brochures_datasheets/mxone/mxone_1023714.pdf
8. design news, Embedded's new star, June 6, 2005, http://www.designnews.com/article/CA603743.html
9. emdebian, http://www.emdebian.org
10. OpenEmbedded, http://www.openembedded.org
11. MontaVista software, http://www.mvista.com
12. TimeSys, http://www.timesys.com
13. ELJonline, Real Time and Linux, Part 3: Sub-Kernels and Benchmarks, http://linuxdevices.com/articles/AT6320079446.html
14. FSMLabs Inc., Micheal Barabanov, A Linux-based Real-Time Operating System, June 1, 1997, http://www.fsmlabs.com/images/stories/pdf/archive/thesis.ps
15. FSMLabs Inc., RTLinuxFree, http://www.fsmlabs.com/rtlinuxfree.html
16. Linux Journal, Real-Time Applications with RTLinux, January 20, 2001, http://www.linuxjournal.com/article/4444
17. Wind River, Product Overview of the General Purpose Platform, Linux Edition, https://portal.windriver.com/portal/server.pt/gateway/PTARGS_0_1_353835_0_0_18/General-Purpose-Platform-overview.pdf
18. Wind River, Product Overview of the Platform for Network Equipment, Linux Edition, https://portal.windriver.com/portal/server.pt/gateway/PTARGS_0_1_353860_0_0_18/Platform-for-Network-Equipment-overview.pdf
19. Koan Software Engineering, http://www.koansoftware.com/
20. SYSGO, http://www.sysgo.com/

21. RTAI, http://www.rtai.org

22. Karim Yaghmour, Building Embedded Linux Systems, 2003 (O'Reilly)

23. Patrik Lantto, Kravanalys och implementering av ett realtidssystem med höga dataprestanda, March 4, 1997, www.snart.org/docs/Exjobb97/lanttoexj.pdf

24. EmbeddedMarket forecasters, Jerry Krashner, Total Cost of Development, July 2003, www.embeddedforecast.com/EMFTCD2003v3.pdf

25. WindowsForDevices.com, SPECIAL REPORT: Study says embedding Windows costs less than embedding Linux, July 16, 2003, http://www.windowsfordevices.com/news/NS3540266275.html

26. LinuxDevices.com, VDC: mobile dev times longer, less predictable with Linux, April 20, 2005, http://www.linuxdevices.com/news/NS3477393296.html

27. Dave Jones, The post-halloween document. V0.50 (aka, 2.6 - what to expect), http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt

28. Robert Love, Linux Kernel Development, Second Edition, 2005 (Novel Press)

29. 2CPU.com, Linux Kernel Comparison: 2.6.4 vs. 2.4.25, March 29, 2004, http://www.2cpu.com/articles/98_1.html

30. DENX software engineering, WolfgangDenk, Comparing Linux 2.4 and Linux 2.6 Kernels, April 24, 2005, http://www.denx.de/twiki/bin/view/Know/Linux24vs26

31. Telefonaktiebolaget LM Ericsson, Ibrahim Haddad, Towards Carrier Grade Linux Platforms, 2004, http://www.linux.ericsson.ca/visibility/Linux_Konferenca_2004.pdf

32. PCI Industrial Computer Manafacturers Group, AdvancedTCA Tutorial, June 23, 2004, http://www.picmg.org/pdf/Supercomm_Tutorial.pdf

33. OSDL, CGL data sheet, http://www.osdl.org/docs/cgl_data_sheet.pdf

34. SA Forum, http://www.saforum.org

35. SA Forum, Service Availability Forum Hardware Platform, Interface Specification, SAI-HPI-B.01.01, http://www.saforum.org

36. SA Forum, Service Availability Forum Application Interface Specification, Volume 1: Overview and Models, SAI-AIS-B.01.01, http://www.saforum.org

37. Andrew S. Tanenbaum, Albert S. Woodhull, Operating Systems, Design and Implementation, Second Edition, 1997 (Prentice Hall)

16 About 4Real AB and the author

4Real AB is a Real Time and Hardware consultant company located in Danderyd 9 km outside Stockholm, Sweden.

We work primarily with product development of Embedded Systems. Our 30 consultants have between 5 and 25 years of experience in the field of electronic circuit design, embedded software and project management.

We can take full responsibility for a product development, from idea to production, or we can take part in your project in miscellaneous ways, e.g. mentoring, technical investigations or project/product reviews. All the consultants are partners which implicates a strong commitment and active interest in the company's development.

Alain Mosnier, the author of this report, holds a Master of Science in Computer Science from the School of Mines of Nancy, in France (Ecole Nationale Supérieure des Mines de Nancy). He has 11 years of professional experience from the IT and telecommunications industry, in both technology and sales. He is currently a consultant and partner at 4Real AB.