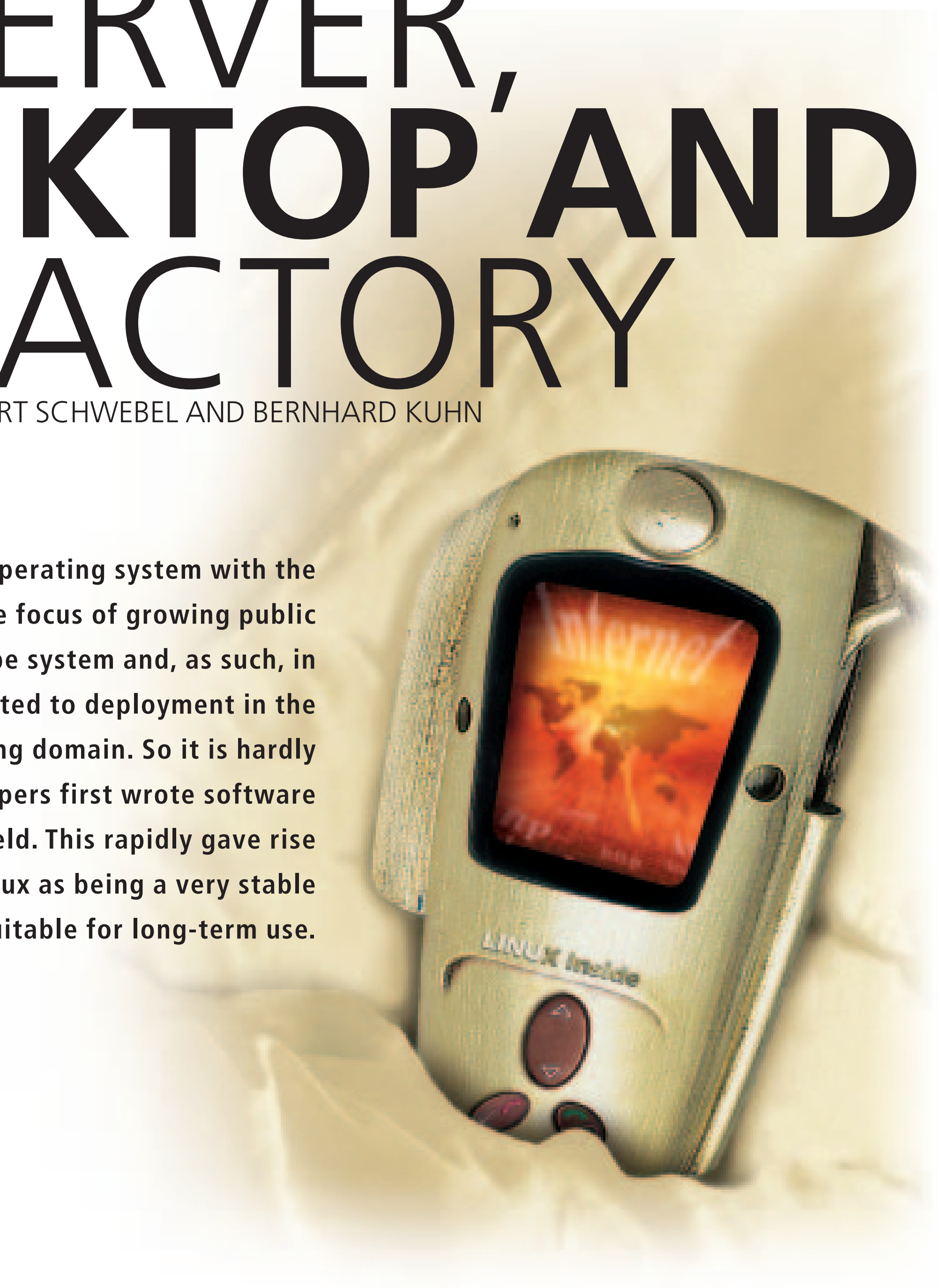
Linux in Industry
SERVER, DESKTOP AND FACTORY

ROBERT SCHWEBEL AND BERNHARD KUHN



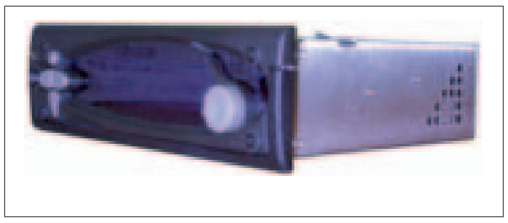
For some years now, the operating system with the penguin has been the focus of growing public interest. Linux is a Unix-type system and, as such, in the early days it was well suited to deployment in the server and networking domain. So it is hardly surprising that many developers first wrote software which was useful for this field. This rapidly gave rise to the reputation of Linux as being a very stable operating system that was suitable for long-term use.

Many users and programmers soon began to create solutions for everyday office applications so that now Linux increasingly covers this market too. But what are the advantages of using Linux in industrial plants?

Special requirements

It is hard for proper operating systems to establish themselves in the world of microcontrollers and programmable control systems. The requirements are usually such that an ordinary office PC could not meet them. Extreme fault tolerance, hard real-time demands, communication capability, integration into large measuring and control equipment, high temperature and impact requirements are not conditions with which you would expect your desk PC to cope. A single general protection fault would give rise to enormous expense and serious damage and would not be tolerable. So it is no surprise that microcontrollers and customer-specific computer systems have so far occupied this market.

In fact, industrial problems are frequently highly specialised. Many computers work as embedded systems without graphical interfaces and are deeply implanted inside measuring devices, switchgear cabinets or plant, without being visible as computers from the outside. It is precisely the


Empeg Auto-MP3-Player: *http://www.empeg.com*

requirement for high serviceability of the systems which means a controller should, if at all possible, only fulfil such functions as those for which it is needed. All components that are not in use could act as a potential source of error and also cost money unnecessarily.

So why an operating system for industrial applications, and why Linux? Because of its very special genesis and the philosophies behind it, we now have access to a system that in many cases can fulfil the aforementioned requirements very well and which fits in perfectly with the modern structures in industrial plants.

The modular heart of the matter

Strictly speaking, Linux means only the operating system's kernel, which is responsible for all basic administration tasks. These include the organisation of program runs, the allocation of processor time to individual processes, access to files, and the network and other input and output components. In short, all of the processes that matter in a system when it is running.

One of the central concepts of Linux is modularity. Not even the kernel of the operating system is constructed as a monolith. In fact, most of it has been realised in the form of so-called kernel modules. The developer who installs a Linux kernel on his system can decide, for each module, whether he wants to have it permanently (static), only temporarily (dynamic) or not at all, in his individual installation. Unwanted parts can be left out without affecting the stability of the system.

But this modularity does not end with the operating system kernel – it continues in the domain of systems software. Functions that are not directly part of the kernel are obtainable as independent program packets and strictly speaking they do not really come under Linux either. The principal one is the GNU Project, in the frame of which practically all the essential tools and systems-related components were developed.

Examples of such components are the management and synchronisation of system time, starting of timed programs, the provision of network services such as mail, access to ISDN and modem lines, embedded webservers, databanks and much more.

There are great advantages in the fact that these components are not part of the kernel. The kernel already had the ability to monitor system-related components in operation and to limit their access. No program is completely error-free, but this structure ensures that the kernel catches errors and that their effects are limited. For instance, if the keyboard and mouse of a computer are no longer usable, in most cases it is still possible to log in via the network and sort out the system.

The single-chip computer to the mainframe: same sources

When, at the start of the nineties, Linus Torvalds began to develop the kernel of Linux as a programming exercise, he did not think that his code would ever run on a processor architecture other than his 386. Many developers have now made their contribution to porting the kernel onto other families of processors.

Nowadays Linux can be used on most of the available 32-bit architectures. This means that there are platforms available from the tiny on-chip system (e.g. Axis Etrax100) in the embedded application up to large multiprocessor computers or mainframes (IBM S/390). The secret behind this is, again, modularity: as only a relatively small part of the total system has to be adapted for the respective processor, most of the rest of the components can be reused directly. It is only the hardware-dependent segments that really have to be re-programmed, which essentially means hardware drivers and the boot procedure. All higher-level services are not in fact part of the kernel and most run unchanged.

For the programmer in development the saying: write once, run everywhere, holds true. Software developed on an ordinary PC can usually be converted for the embedded platform without any alteration, as long as attention is paid to the equipment limitations to which the embedded computer is subject.

[top]
Kerbango Internet Radio

[below]
Digital video recorder: Philips TiVO Personal TV Receiver *http://www.tivo.com*

Hardware drivers made easy

Access to hardware is a privileged action. This means that not every application program under Linux can access an input/output port, perform DMA or interrupt activities. In fact, hardware drivers are realised as kernel modules. If a driver is to be used for a self-developed card, this does not mean the Linux kernel has to be re-converted. Modules can be taken as completely independent programs, which can be loaded into the system at run time and removed from it again later. They run only with the privileges of the system administrator (root) and thus have full access to the hardware.

For drivers, the Unix concept 'Everything is a file' applies. So-called device files are closely linked with the actual Linux driver, and these are found in the directory */dev*. This is not a real file on the hard drive, but a virtual interface between application program and kernel driver. Assuming a driver is to be developed for a measuring card named 'measuring card', which digitises analogue signals, then the pseudo file for the measuring card would be called something like '/dev/measuringcard'. Access to this file is now possible in the usual way: The C programmer will open it like an ordinary file using *open()*, read from its data using *read()* and finally close it again using *close()*.

The application program does not notice the difference between the drive interface and a real file. This means it is easy, in the test phase, to provide a file with simulated data and thereby test the behaviour of a program.

Software development with respect to the driver is really simple. It is only necessary to define which activities the driver is to perform when the user opens or closes the pseudo file or reads out its data. The least realistic devices, though, deliver only pure data. Usual control commands are also necessary, for example to switch the measurement ranges in the aforementioned measuring card. To do this there is an additional function called *ioctl()* (Input / Output Control). This can be used to implement an interface for any additional commands for initialisation, control and monitoring of the card. There is minimal expense involved in integration in the operating system.

FrontPath ProGear Webpad
http://www.frontpath.com/progear.htm

Communication with the environment included

Control of hardware components is important, but equally important is the communication of an embedded system with its environment. If a man-machine interface in the form of a GUI is not necessary for this, the network capabilities of Linux come into play.

The simplest way to communicate with an embedded system is via a serial port. All components of this are available on every Linux system, when support for the interface has been integrated into the structure of the kernel. It is up to the developer to choose which program monitors the interface.

Free networkability

In the case of embedded network applications in particular (firewall, VPN-bridge, intrusion detection system), the Linux Kernel 2.4 in combination with the corresponding programs is hard to beat. One additional aspect: A few small and medium-sized enterprises are currently facing the challenge of having to make their products Internet-capable. Whether it's a lift system or a CNC milling machine – it has to have an integrated webserver for remote status queries.

In many mechanical engineering firms, this is actually turning into a serious problem: electronics and software play a subordinate role here and often Assembler language and Dos are still being used. So the path to Linux and GNU Tools is often shorter than that to an overloaded integrated development environment.

One interesting option is the use of a so-called 'getty'. These programs usually serve to ensure that a local user can log onto a system and work on the computer in text mode. Because of the modular structure, however, there is nothing wrong with starting a 'getty', not on the local console (embedded systems may not have a graphics card and no keyboard), but on the serial port. If a modem is also used, the device can be serviced remotely without any additional actions.

Any terminal program (e.g. Hyperterm under Windows, which comes with the package) can be used to dial into the industrial computer from anywhere in the world and work on it exactly as if one were sitting at a local console. This functions because of the concept mentioned above of the pseudo files. Ultimately, the local console (*/dev/console*), to the 'getty' program, is nothing but a serial port (*/dev/ttyS0*), namely a file.

In principle, this means that any option supported by Linux is open for communication. In recent years the trend has been increasingly in the direction of using Ethernet as 'unified field bus' for

[left]
Bluetooth/DSL-Webpad/Telefon Ericson H610

[right]
Indrema Entertainment System games console

industrial communication. Linux is ideally prepared for this trend. Since its origins lie in the network communication of the classic Unix computer, drivers and all-important protocols for Ethernet are already included. But this extends far beyond the systems-related part. There is nothing wrong with installing small webservers on a measurement computer, via which data from measurement points can be accessed directly from the company's own Intranet. So there is no obstacle to integration in a company-wide network structure. All standard protocols from the IT world such as HTTP, ftp, Telnet etc. can be used.

The integration of other field buses has now become a reality: there are now drivers available for almost all the major field buses (Profibus, CAN, RS485, ...), so that a Linux PC can also be deployed at the interface between classic field bus technology and modern industrial Ethernet.

The more communication options there are available, the greater the role-played by the question of access security. After all, the competition must not for example gain access over the Internet to measurement values from a quality control system. But here, too, conventional IT technology can be of further assistance. With 'secure shell' there is a protocol available with which one can make fully encrypted connections.

With this, authentication of users and the entire communication are performed with the latest encryption technologies. Lastly each network service can be encapsulated into a 'secure shell' and tunnelled through what may be an insecure network.

Resource saving

It's hard to believe how small Linux can be: Uncompressed kernels with 256K are quite possible, such as with uClinux on Motorola 68K derivatives. In this case, however, there is no TCP/IP stack, which makes for limited use.

Usually, depending on the application and target platform, you should reckon on 1 to 4Mb flash and 2 to 8Mb Ram requirement. Other embedded operating systems are (assuming the same requirements) not much more economical.

Development tools en masse

In most cases, applications for embedded controllers are created using a cross-development system. This usually contains an integrated development environment with compiler, debugger and all the necessary tools. In addition to these, an in-circuit emulator is often needed if the application is to be tested directly in the target circuit.

In Linux-based systems modularisation also extends to the domain of development systems. The most frequently used GNU Compiler Suite GCC provides front-ends and back-ends. The former are there to translate a programming language, while the latter create machine code for a specific processor. If, for example, code is to be created on an Intel PC, which is running on a PowerPC, only the right back-end needs to be chosen. The Intel back-end can be used in the test phase, which allows the code to be translated, tested and debugged in one's own time on the development computer. Conversely, with different front-ends a wide variety of programming languages can be used (such as C and Pascal combined in one project).

But there are also other advantages to network capability. If software is to be developed for a small processor board, which has no hard disk of its own, but is running directly from the Flash Rom, the following procedure would be suitable. The embedded system does not boot a kernel located on the flash disk in the development phase, but the complete file system together with all operating system parts and application programs is integrated

Adomo Webpad

via the network. The processor board does not 'notice' any difference here, as to whether the files are present locally on the Flash disk or via the network connection. Developers have the option of inserting new program versions for the running system directly onto its hard drive. The GNU debugger 'gdb' which is part of GCC is equally network-capable. This means any application can be remotely debugged without using any special tools. If programming is complete, all you need do is transfer the complete development data tree from the large computer onto the Flash disk.

Real time with Linux

Hard real-time demands are very important for many applications in the field of measuring and control technology. If a control device does not respond within precisely defined time limits, vital measurements could be lost or control commands be missed, which might very well lead to damage to the plant or production stoppages.

There are a number of real-time expansions available for Linux. With these, firstly a small, hard real-time capable kernel runs on the hardware, which then only assigns the actual Linux kernel processor time and resources when there are no real-time programs waiting which need them. This ensures that the real-time programs, completely independently of events at Linux level, always get the scheduled processor time, thus guaranteeing that response times to interrupts of approx. 10µs can be achieved without any problem.

The advantage of this concept is that all normal operating system services are still available on the real-time capable computer. So such things as status information can be viewed via a webserver or a graphical display, or data can be saved to a hard disk.

Graphical interfaces

Depending on the equipment on a computer, there are some very different options for visualisation. Control devices can send textual outputs to a two-line display with serial port. If a real graphical user interface is desired there and then, there are various toolkits available for this, which offer all the usual widgets of modern window interfaces. The X-window system normally used on Unix systems is network-transparent. This makes it possible to send the output of a controller together with all control options via the network into a control room or even to a far remote computer via the Internet. For systems with very modest resources there are also a variety of graphics toolkits available which offer fewer features but on the other hand can be accommodated in a few hundred kilobytes of space.

Double investment protection based on free licences

By now many of the options available under Linux have been introduced. One great peculiarity here are the licence models which are used for most

Mini-Linux Distributions

Name	URL	Floppy/ HDD	RAM requirement	Kernel	Content / Comment
hal91	http://home.sol.no/~okolaas/hal91.html	1.44Mb	not known	not known	Ancestor of all mini and embedded Linuxes, designed as rescue diskette and 'mobile Linux'
Floppyfw	http://www.zelow.no/floppyfw/	1.44Mb	8Mb	2.2.13	Offers static router and firewall functionality, Floppyfw stems from hal91
LOAF	http://loaf.ecks.org	1.44Mb	4Mb	2.0.36	Linux on a floppy with diverse network clients such Telnet, ssh, nfs, ftp, lynx and network configuration tools
Tomsrtbt	http://www.toms.net/rb	1.722Mb	not known	2.0.36	Contains over 100 Unix commands, most important man-pages and some kernel modules for network cards and SCSI adapter on a reformatted diskette (82 tracks, 21 sectors)
Trinux	http://www.trinux.org	2 x 1.44Mb	12-16Mb	2.2.x	Contains various tools for network monitoring and detection of network errors
mulinux	http://mulinux.firenze.linux.it/	1.722Mb	4Mb + Swap	not known	Contains various text-based network clients such as mail and newsreader and the Web browser Lynx and the necessary network tools for making connections
LRP	http://www.linuxrouter.org	1.44Mb - 120Mb	12-64Mb	not known	The Linux Router Project is scalable from a simple IP-masquerader or Remote Access Server up to a WAN router – can also be used for X-terminals.
SmallLinux	http://smalllinux.netpedia.net/smallin.htm	1.44Mb	2Mb-4Mb 1.0.9		Small Linux basic system with especially low memory requirement due to the use of the 1.0.9 kernel

parts of Linux. All parts of the operating system itself are Free Software. Kernels and the overwhelming majority of the system-related tools as well as a great many programs are licensed under the GNU General Public Licence (GPL). The purpose of this is to protect Free Software. Software licensed under the GPL can be altered and even sold, without licence fees being payable. The only condition is that code derived from GPL-licensed code must be released again under the GPL.

There are some advantages to this procedure. GPL software is mostly developed by many programmers scattered all over the world, collaborating over the Internet. Many programs (for which, with other operating systems, you would have to reinvent the wheel) are already available as GPL software and can be used immediately. Developers can devote their entire concentration to their own tasks. For the software components created by the Free Developer Community, you will usually receive excellent support by e-mail, at no cost. The channels of communication are short, and suggested changes are often realised very rapidly by the author. Since the worldwide developer community is very large, for any problem someone will quickly be found who has a ready solution – or is creating one.

If software is developed solely for Linux, but uses no GPL code, then obviously it is possible to produce non-free software. Nor is the interaction between free and non-free programs a problem, so those proprietary algorithms can also receive full protection. The often-quoted saying that under Linux you always have to give away your code is completely false. However, anyone who develops software according to a Closed Source Model is also unable to benefit from the advantages of Free Software.

In the domain of automation in particular the procedure for the development of Free Software often poses no great problem. Customers want to buy solutions to problems, not the software as such. Many problems in this field are so highly specialised that the actual performance of a company consists of adapting and tailoring software to the requirements of their customers. Investment cycles in industry are considerably longer than in the IT domain and so Free Software is also an important argument when it comes to security of investment.

If a firm discontinues a proprietary product, the customer has a problem. The best thing that can be done in this instance is to undertake an expensive migration to another generation of software. The worst case scenario is that the product is worthless and cost-intensive restructurings of entire plants have to be undertaken. But if Free Software is used, the customer can himself entrust a reliable company to maintain the status quo, since he does have access to the sources.

One example of embedded Linux: 'Spectator' powder quality control system
From the company Zeutec Opto-Elektronik GmbH. There is a 'big' industrial computer in the switchgear cabinet, which controls a compressed air-controlled sampling unit. The PC scans optical sensors and generates measurement values as a result, which are sent via an Ethernet link to a control room. The measurement device is currently in the trial stage near Cologne. The complete control of the solenoid valves, scanning of the sensors and evaluation of the data is done with software written in Perl. Standard protocols are used for communication (Telnet, HTTP, ssh).

In full bloom

What started out as tinkering with small projects such as the Linux Router Project has now become an economic factor of increasing importance. The figures in this article can show only a small number of the embedded devices developed and built in the past year.

More than 120 enterprises have now joined the 'Embedded Linux Consortium'. Nor should the 24 well-known Japanese firms in the EMBLIX consortium be forgotten. So anxious new customers no longer need to complain about the lack of support on the part of industry.

Conclusion

Linux is now an operating system to be taken seriously for industrial applications. It is robust, can be adapted to circumstances by modular means, is platform overlapping and network-capable. The same system runs from a battery-powered 386SX with 8Mb Ram via PC/104 components up to industrial PCs, desktop computers, parallel computers and mainframes. Security is a fundamental part of the system and does

Pneumatically-actuated sampling unit to inspect powdery substances.

not have to be added on later at great expense and half-heartedly. Existing cross-development tools make software testing simple and elegant, while the programmer does not have to get used to new tools, regardless of whether he is programming for the desktop or an embedded system. If necessary there is the option of complying with hard real-time requirements.

Of course, Linux is not the answer to all the problems that can arise in industrial use. But it does have the potential to unite at least part of the diversified software market for embedded systems. The fact that in qualitative terms, extremely high value products are being created can be seen from the victory parade of server and desktop applications. ■

The author

Robert Schwebel has been involved with Linux since 1994. Since completing his degree in electrical engineering he has been developing measurement systems for automation and environmental technology under Linux and drivers for hardware components.

Info

Real-time expansions: http://www.realtimelinux.org
Linux Router Project http://www.linuxrouterproject.org
Embedded Linux Consortium http://www.linux-embedded.org
Japanese Linux Consortium http://www.emblix.org
Embedded Linux News and Reports http://www.linuxdevices.com
Portal site on automation systems http://www.linux-automation.de
Comprehensive newsticker http://www.linuxdevices.com
Linux Lab Project http://www.llp.fu-berlin.de

■

Embedded Systems

What is an embedded system? There is no simple, universal answer to this since conceptions diverge depending on the sector and domains of application. For this reason, the following explanation is only an attempt at a definition.
An embedded system interacts with its (usually electromechanical) environment (embedding system). No human user is needed here, or if so, he has no need of any knowledge whatsoever about the technical innards of the embedded system to be able to use the system as a whole.
Classic embedded systems in industry are, for example, the process control computer of a CNC milling machine (Computer Numeric Control) or a modern SPS plant. In the past decade a whole range of consumer products have come onto the market, all containing electronics which are equally embedded systems – such as mobile phones, video recorders etc. Ideally, computers ought also to be embedded systems for the end user – but the definition above applies best to the Apple Macintosh, as here at least 'knowledge of the technical innards' is necessary. Although the Surfstation from Mobilcom only differs very slightly in its principle architecture from a home PC, this is a true-blue embedded system. In fact, among small and medium-sized enterprises standard PC components have been gaining more and more ground in new developments of embedded systems – assuming small to medium production runs and short product cycle times. No wonder: Commodities Off The Shelf, COTS for short, often cost just a fraction of comparable proprietary industrial solutions and at the same time are considerably more flexible (but also more prone to errors).
These embedded PCs obviously cannot be used in every situation however. Often, hostile environments and a notorious lack of space force the adoption of application-specific solutions. In the case of large production runs this is obligatory anyway.