Embedding Linux to Control Accelerators and Experiments

A scientific laboratory in Europe depends on Linux for controlling equipment used in its research.

by A. Götz, P. Mäkijärvi, B. Regad, M. Perez, P. Mangiagalli

Linux is being used at the European Synchrotron Radiation Facility to build distributed embedded controllers. The embedded systems are either PC/104-based systems, which boot from a flashdisk, or VME crates which boot from the network. The devices being controlled vary from serial lines to stepper motors to CCD cameras. The control software is written using an object-oriented toolkit that we developed, called TACO (Telescope and Accelerator Control Objects). Using TACO all control points are implemented as device objects. This article will describe how we have implemented embedded controllers using Linux and present some examples.

Introduction

What is an embedded controller? As the name indicates, an embedded controller is a mixture of control hardware and software which is embedded, i.e., integrated, into the equipment it is supposed to control. Examples of where embedded controllers can be found are abundant in daily life: printers, portable telephones, the brakes of your car, etc. The requirements for embedded controllers are usually that they be physically small, have a small memory footprint, low power consumption and low cost. Most of these requirements are dictated by the fact that controllers will be built into hardware systems and in large production volumes.

Figure 1. Aerial view of the ESRF nestled between the Drac and Isere rivers at the foot of the French Alps (Grenoble).

At the European Synchrotron Radiation Facility (ESRF, see Figure 1), our main job is to connect a wide variety of control hardware with an equally large number of different interfaces. We are faced with the task of interfacing hundreds of power supplies to control the accelerator magnets which guide the electrons, thousands of stepper motors which move and position the experiments on the beamlines and a myriad of other devices. Most of these devices contain embedded controllers which export their functionality via a dedicated interface such as a serial line, parallel interface or computer bus to the outside world. Our task consists of determining how to interface these different hardware devices in a coherent and efficient manner so that higher-level software (often not written by us) can access it easily.

The European Synchrotron Radiation Facility

The ESRF, located in Grenoble, France, is a multinational research institute supported by 15 participating countries. The ESRF is operated as a non-profit enterprise under French law. Management

is supervised by a Council whose delegates are designated by the member parties.

This large experimental facility performs basic and applied research in physics, chemistry, material and life sciences. The research is facilitated through the use of a powerful source of radiation in the X-ray range. This synchrotron radiation, used at experimental stations called beamlines, has remarkable capabilities. Pushing the technological limits, the ESRF performs novel experiments which have not been feasible before.

The construction of the ESRF started in 1988. The inauguration and opening of the first 15 beamlines to scientific users took place in September 1994. Currently, 40 beamlines are operating 24 hours a day, 7 days a week. More information about the ESRF can be found on their web site at http://www.esrf.fr/.

Our answer to the interface problem has been to use Ethernet and the TCP/IP protocol as the ubiquitous interface for all devices. We have developed an object-oriented toolkit called TACO for wrapping all devices and then exporting them to the external world (our users) as objects on the network, accessible via an application programmer's interface (API). In order to wrap the different pieces of hardware, we often build embedded controllers close to the hardware which run the TACO wrapper software and make the hardware available via the network.

We will describe two types of embedded controllers which we build using Linux--PC/104-based controllers and VME-based controllers.

Why Linux?

Why use Linux for embedded controllers? Although the choice to use Linux is obvious to many readers, the reasons are not always the same. In our case, we needed an operating system with the following features: highly configurable, excellent TCP/IP stack implementation, easy to program when it comes to writing device drivers, modern programming standard (e.g., POSIX, CORBA, Java, HTTP) support, stable, well-supported and not too expensive. Linux fulfills all these requirements and more. The fact that Linux is free and comes with the source makes it even more attractive.

Before Linux, we used commercial operating systems for embedded controllers and after two years of working with Linux, we noticed the difference. Gone are the days of flaky TCP/IP implementations, silence as an answer to our bug reports (now that we have the source, we can even fix the bugs ourselves), expensive contracts, lack of modern products and no one to talk to about our work. The choice of Linux for our embedded systems has meant that we now have the same OS from the low level to the desktop to our Beowulf cluster.

Choosing Linux does not mean that all questions have been answered, however. Questions remain, such as ``When will industry embrace Linux?'' and ``How long will Linux last?'' The answer to the first one is soon--we hope. The answer to the second is not forever, of course. As with any product/phenomenon, Linux will not last forever. At the ESRF, we renew our control systems roughly every ten years. It is important that Linux continues to exist for this time. In the worst-case scenario, we still have the source.

Embedding Linux

Having chosen Linux as our OS, we chose VME (for historical reasons) and PC/104 for our hardware. Strictly speaking, VME is not an embedded controller. It is a bus-based system which offers the ability to plug many I/O cards into a single crate. The I/O cards can communicate with the hardware without being aware of any other cards in the crate, or they can rely on other cards to do part of the work. This is one of the strengths of VME.

Why talk about VME if it is not a true embedded controller? Because we configure our VME systems like embedded controllers. They are totally diskless, i.e., they have only RAM on board, boot a generic image from the network, and in fact, resemble a general purpose I/O embedded controller (see Figure 2).

PC/104 is, however, a true embedded controller. It is a small-format PC which can be stacked together with I/O cards to form a dedicated controller. It is low-cost, low in power consumption, small and ideal for performing dedicated tasks. How do we configure Linux to boot and run on these two formats? Read on....

Figure 2. Paolo standing next to a medium-sized TacoBox with a PC-104 CPU inside being used as a dedicated SCSI controller to control a data acquisition system. The beast in the background is a robot for manipulating silicon wafers in the x-ray beam.

PC/104 alias TacoBox

The TacoBox project was started at the ESRF in 1995 with the goal of creating a low-cost, networked I/O device controller for distributed control systems. TacoBox is a very simple device with an I/O interface for the physical device to be controlled. The I/O interface is specified by the TACO I/O class as a set of commands possible to use with the physical device. Imagine a TacoBox as a black box which speaks TACO protocol as input and process I/O as output. The user interface is accessed via Ethernet using two major protocols. The configuration and (some) maintenance operations of the I/O interface can be done using an ordinary web browser.

We use the PC/104 form factor CPU and I/O boards to build TacoBoxes. A PC/104 is essentially an IBM-compatible PC in an embedded, industrial format. It is built around a 104-pin stack-through connector which is electrically compatible with the 8- and 16-bit ISA bus (PC-XT and PC-AT). The PC/104+ has an additional 32-bit bus, electrically compatible with the PCI bus. The PC/104 form factor is small, 90 x 95 mm. It is stackable, which means you can build a PC/104 system in a very small space. In a recent survey, PC/104-based systems were the third best-selling embedded bus system standard after VME and CompactPCI. The first prototypes of PC/104 ran under commercial real-time operating systems (RTOS) with limited on-board resources. When we got interested in Linux, we immediately wanted to install it on TacoBox. At first, we started with the same configuration as we used with those RTOS installations. We learned quite a bit about the different embedded Linux distributions out there. But we also learned that with just a little more money, we could get a Pentium-90 level PC/104 Single Board Computer (SBC) with 20MB of DRAM, IDE-interface, plus Ethernet (we used JUMPTec's MOPS/586). Add a hard disk and floppy drive, a SVGA card, a screen, a keyboard and a mouse and

you have an ``old-fashioned'' desktop PC. On a desktop PC, you can install Linux without any problem, thanks to some great Linux distributions. At the ESRF, SuSE Linux 5.3 was installed on our desktop machines. Within a few hours, we had a desktop PC/104.

We fit all that in a box by removing the spinning hard disk first and replacing it with a solid-state, IDE-compatible, 24MB FlashPROM hard disk. It piggybacks neatly on the SBC. You do not want to write on a FlashPROM too often, because it is slow and will not support many continuous writing operations. Therefore, the root file system on the FlashPROM is mounted read-only and all the live files and directories are placed on the initial RAM disk, the /initrd. How do you fit a SuSE Linux distribution in 24MB? You give up Perl, Apache and man pages to get to that size. Again, it is a question of money, since FlashPROM disks with up to 200MB are available. What about the SVGA, the screen and other desktop decoration? Configure a serial line system console, and rip them all out.

VME

A farm of 250 VME bus-based systems is running here. All are running diskless and booting from a UNIX machine using BOOTP/TFTP protocols. The VME Single Board Computers used at the ESRF (see Figure 3) are MVME-167, with 8, 16 or 32MB of memory, and MVME-162-522, with 8 or 16MB of memory. When we wanted to install Linux on this platform, we looked for information on the Internet and were pleasantly surprised to find Richard Hirst's site about Linux for 680x0-based VME boards. What a relief to find such professional support for what we thought would be a major job for months to come.

Figure 3. Example of diskless VME crates used as general purpose input/output boxes. The lower crate (with the box of tacos in it) is running Linux and being used to control a high-resolution encoder card. The upper crate is running a commercial OS to control a variety of input/output cards. Work is in progress to port all the software from the commercial OS to Linux while staying in VME format.

Richard's distribution is for disk-based systems, so he helped us sort out some problems with the NFS-root-based systems. A very useful package that we added on our systems is Nick Holgate's TFTPLILO package, which allows us to have just one Linux boot image for all our Linux/68k VME systems.

We keep boot images for Linux/68k VME systems, as they are used on our FTP server (ftp://ftp.esrf.fr/pub/cs/ess/linux/). This allows all our collaborators to test device drivers and other software in a similar configuration as ours. It also allows any owner of a MVME-167 or a MVME-162-522 board to try Linux/68k without any investment or other hassle.

If you have already jumped on your keyboard, you may have noticed we are using the Debian file system and the 2.0.33 kernel. The good news is that Debian now includes support for MVME-16x, MVME-17x and BVM VME bus Single Computer Boards.

Writing a device driver for a VME bus-based Linux computer requires knowing something about the VME bus and its principles of operation. Briefly, VME bus is an asynchronous bus that allows multiple bus masters, but (luckily) only one arbiter. It has seven daisy-chained interrupt lines that on most SBCs

can be re-mapped to different local interrupt levels. The access to VME bus I/O boards is entirely memory-mapped, again with some physical translation between the local memory address and the VME bus address. All this is usually managed by some complex chips, such as Vmechip2 from Motorola. Writing a device driver for VMEbus requires some understanding of the interface chip. Our ftp site contains examples of general purpose Vmechip2 device drivers. It is good idea to read through those drivers before moving ahead to more complex device drivers.

TACO

To provide a unified software interface to many different pieces of hardware with as many different kinds of hardware interfaces, we have developed an object-oriented toolkit called TACO for doing just that. It is also used at the Hartbeesthoek Radio Astronomy Observatory in South Africa (see Resources) for controlling a radio telescope, and will be used at the FRM II research reactor in Germany for controlling future experiments.

TACO unifies the software interface with the hardware and the way control software is written by providing a framework for developing control objects. Each control point is implemented in a Device class as a method. At runtime, the Device classes create as many copies of Device objects as there are pieces of hardware, and export their functionality onto the network. The Device objects are served by Device servers, which are stand-alone processes under Linux. Clients (which can be graphical GUIs or other Device objects) access the Device objects via the Device Server API (DSAPI).

The network addressing is managed by a database (implemented with **gdbm**) and the network protocol by the ONC RPC (from Sun, which is now part of glibc). RPCs make network calls look like local procedure calls. The ONC RPC is the basis of NFS. This means it can be found on all systems where NFS has been ported, i.e., just about everywhere. It requires one extra process to be running per host--the portmapper, which manages the mapping of program number to local port numbers. Performance overhead induced by an RPC over the network is on the order of a few milliseconds. The memory footprint is less than 100 kilobytes per server, linked with shared libraries (TACO API's, glibc and pthreads).

The DSAPI implements various flavours of network calls needed for doing controls--synchronous (client blocks waiting for the answer), asynchronous (client continues immediately and retrieves answer later) and events (client registers interest in an event and gets sent events asynchronously). It also implements a large number of standard types, timeouts, UDP and TCP protocols and stateless connections. Device classes can be written in C or C++. The database is used to store and retrieve device-dependent information. Interfaces to high-level scripting languages such as Tcl, LabView, SPEC and Mathlab are available. Clients can also be written in C, C++ or Java.

Because of its simple approach, TACO scales easily. It has been used for laboratory systems consisting of a single device right up to an entire synchrotron which consists of more than 10,000 devices. We use TACO to control everything from simple digital I/O to entire data acquisition systems.

We are presently working on the next generation of TACO called TANGO (TAco Next Generation Objects). In TANGO, CORBA will replace ONC RPC, and it will be possible to write Device classes

in Java as well as C++.

Examples of Embedded Controllers

We started using Linux in embedded controllers approximately one year ago. To date, we have built the following controllers.

Serial Line Controller

We have traditionally used a VME-based board with 16/18 ports to control RS-232 and RS-422 serial lines. We have thousands of serial lines today to control vacuum devices, power supplies, PLCs, etc. As the number of devices requiring serial lines increases each day (and in some crates, we have reached the maximum number of serial line VME boards), we started looking for a new system, which would be cheaper than the existing VME-based system and would be PC-based. (Many of the serial line devices are certified only on PCs.) That is how we came to the PC/104. Cost is not the only reason for changing: we also wanted to implement new technologies, like web support, for better maintenance and configuration of the serial lines. We wanted as many serial lines as the VME-based system; therefore, we looked for a PC/104 board with eight ports which we could stack. Unfortunately, the choice of boards with more than 4 ports is limited. We finally chose the Parvus Octal Serial card.

This board is a so-called ``dumb multiport serial'' board, based on two 16554 UART (which are compatible with the standard 16650 UART) plus a programmable gate array for the address and UART register mapping on the ISA bus. As this board is quite new, we encountered some problems with the on-board gate array (bugs and limitations of programming) but we also found workarounds. To integrate this board in our Linux TacoBox, we used the standard serial line driver of the kernel (2.0.35). It implements IRQ sharing; thus, we consume only one IRQ for the eight ports of a Parvus board.

Before accessing the extended serial lines through the system device descriptors **/dev/ttyS**, we have several setup steps. All are processed from a single homemade script. Therefore, to install one (or more) Parvus boards, we need to add a single line to the booting scripts. We added a call to our script in the /etc/rc.d/serial file.

The first step is to set up the Parvus card gate array. We wrote a C program to access the gate array registers. The script picks up all the parameters needed for the call to this program and for configuring the serial lines from an easy-to-understand and modify ASCII file. Then the device descriptors /dev/cuaxx and /dev/ttySxx are created. Since we use the standard driver, the major numbers are the same as the ones used for the CPU board serial lines (4 for ttySxx and 5 for cuaxx). As with any other dumb multiport card, the minor numbers start from 64. At this step, we also change the access permissions to the device descriptors to allow any user to use the serial lines.

The last step is configuring the serial ports using the helpful tool **setserial**. We set up each port of a board with a different address but with the same IRQ; for example, addresses 0x100, 0x108 and IRQ 12 for one board and addresses 0x140, 0x148 and IRQ 9 for the other board. At this point, the serial ports are fully integrated in the operating system and can be used from any program.

Figure 4. Manuel (aka RastaMan) after a heavy day of debugging the serial line TacoBox, feeling like he has serial lines coming out of his head.

For our embedded controller, we ported the existing serial-line TACO device server from our commercial OS to Linux. The new device server uses POSIX calls to interface with the serial-line driver and should therefore work with any OS supporting POSIX. The compatibility of the two device servers and the use of TACO allows us to interchange a VME-based system with a PC/104-based system without changing the client application.

Stepper Motor Controller

The experiments conducted at the ESRF beamlines require precise positioning of many motors to move goniometers, slits, translation stages, etc. Typically, a beamline has more than a hundred stepper motors to align the beamline and move the sample during the experiment (see Figure 5). Motors can be divided into two types: those which can be moved independently and those which have to be moved in sync with the data-taking process. For the independent motors, a TacoBox based on an embedded PC/104-based controller board is ideal--it is compact and can be installed close to the hardware. For the synchronized movements, VME is the preferred solution. It allows us to synchronize the motor movements with the data acquisition via the VME bus without requiring any extra cables.

Figure 5. An example of motor positioning on a beamline--a 6-legged hexapod used to position the crystal in the beam to select only a single wavelength of the beam.

In order to kill two birds with one stone, we chose a stepper motor controller which exists in PC/104 and VME format--the PC68 and VME58 from Oregon Micro Systems. These cards supports step rates of up to 1MHz, very useful for microstepping where the steps are divided by a factor of 1000 to ensure precise positioning. Both cards implement the same ASCII-based controller language. They differ in their register mappings on the bus (one is Intel I/O port-based and the other is m68k memory mapped). The differences are implemented (and hidden) at the level of the device driver. This ensures the TACO device server is identical for both cards. We were fortunate enough to get outside help from Richard

Hirst (again) to write the device driver, which is based on an initial version we found on the OMS web site written by Tony Denault of the Institute of Astronomy, Hawaii. The driver implements **ioctl** calls to read position and status and to pass commands to the board. A multi-threaded device server which supports events was developed at the ESRF (based on an initial version by Lucile Roussier of Lure Laboratory in Paris). The server uses POSIX threads on Linux/m68k and Linux/x86. All the software (device driver and server) is available under the GPL from our FTP site at ftp://ftp.esrf.fr/pub/cs/ess/linux/drivers/oms/

Figure 6. Paolo and Andy in their bug-fighting gear with the stepper motor TacoBox (second shelf from the top on the right-hand side) in its rack on the ID27 beamline clean room. The other boxes are the stepper motor power drivers and the TacoBox power supply.

The PC/104 controller consists of a JumpTEC 486 CPU board and the OMS PC68 controller stacked together (see Figure 6). The TacoBox boots Linux from flashdisk, loads the device driver module, creates as many device descriptors as needed (using major number 26), then starts the device server. On VME, an extra step is needed to program the Vmechip2 according to the ESRF addressing standard.

Once the device server is running, the client requests the server to move the motors. A minor problem we had was ensuring that the PC68 card did not clash with any I/O address already in use, e.g., the address of the network card. This is easy to determine by listing (using **cat**) the /proc/ioports file and choosing one that is not attributed. A more serious problem was with the VME version of the card and the way it handles interrupts when hitting a limit switch. In the end, we abandoned using interrupts with limits; we simply read the status register to find out if a limit had been hit.

For the stepper motor-based TacoBox, the usual PC/104 cabling problems are reduced somewhat because there is only a single flat cable for the motor controller and an Ethernet cable to connect. The entire box cost us approximately 2000 euros for four stepper motor channels and 3000 euros for eight channels (using the PC68 extension board). In our application, we use the PC68 as a simple stepper motor controller; we haven't tried the other features yet (e.g., support for relative encoders, dc motors and servo loops).

Collaborating On Device Drivers

Today, the main drawback of Linux for our applications is still the lack of device drivers for many kinds of I/O boards. PC boards are better supported than VME, but still lag far behind Windows drivers. Our aim is to install as many Linux-based controllers as possible because of their better stability, ease of programming, flexibility, lower cost, etc. In order to achieve this, we need drivers, drivers and yet more drivers.

We are only a small team working on device drivers and are therefore very interested in collaborating with other programmers on device drivers for all kinds of boards. All our drivers are developed under GPL and made available to the external world on our FTP site.

We would be interested in hearing from anyone who has written a device driver for an I/O board for PC/104 or VME. Please send e-mail to one of the authors to add your driver or name to our database,

which we will make available on our web site.

Conclusion

We hope this article has given you a taste of how to build embedded controllers using Linux and how Linux is being used in a research institute like the ESRF. All our software is available free of charge (and guarantee), with source code. At present, we have built only a few embedded controllers based on Linux, but we plan to build many more in the future. Should any real-time problems crop up, we will explore them using RTLinux.

Linux has proven to be a great OS for collaboration. All of our software comes from the Internet, and we have often had direct contact with the authors. We hope to collaborate with more device driver programmers in the future to bring the list of Linux device drivers up to be at least as long as for Windows, if not longer.

Figure 7. Is it Richard Stallman? No--it's Paolo preaching the gospel of Linux and Open Source in the clean room.

References

Acknowledgements

Andy Götz aka Mr. Linux (goetz@esrf.fr) has worked at the ESRF for ten years. His main interests are distributed control, astronomy, travel and Linux.

Petri Mäkijärvi aka Mr. TacoBox (petri@esrf.fr) has worked at the ESRF for almost as long. He is responsible for embedded systems, real time, making great web pages and recently, Beowulf clusters.

Bernard Regad aka Mr. VmeBoot (regad@esrf.fr) has been with the ESRF for six years. His main interests are configuring and booting diskless VMEs, installing Beowulf clusters and 70's music.

Manuel Perez aka Mr. RasterMan (perez@esrf.fr) is a longtime ESRF collaborator and now colleague. His main interests are building the Serial Line TacoBox, programming, cinema and Linux.

Paolo Mangiagalli aka Mr. Medea (mangiaga@esrf.fr) is the newest ESRF member. His main interests are making the MEDEA beamline the best, building TacoBoxes real cheap, and moving hexapods.
